

Handbook of Magma Functions

Part XII: Representation Theory

Version 2.30-1

Sydney

November 10, 2026

Editors

John Cannon *Wieb Bosma* *Claus Fieker* *Allan Steel*

Handbook Contributors

Eran Assaf *Geoff Bailey* *Wieb Bosma* *Gavin Brown* *Nils Bruin* *John Cannon* *Jon Carlson* *Scott Contini* *Edgar Costa* *Bruce Cox* *Brendan Creutz* *Steve Donnelly* *Tim Dokchitser* *Willem de Graaf* *Andreas-Stephan Elsenhans* *Claus Fieker* *Damien Fisher* *Volker Gebhardt* *Sergei Haller* *Michael Harrison* *Florian Hess* *Derek Holt* *David Howden* *Al Kasprzyk* *Markus Kirschmer* *David Kohel* *Axel Kohnert* *Dimitri Leemans* *Paulette Lieby* *Graham Matthews* *Scott Murray* *Eamonn O'Brien* *Dan Roozemond* *Ben Smith* *Bernd Souvignier* *William Stein* *Allan Steel* *Damien Stehlé* *Nicole Sutherland* *Don Taylor* *Bill Unger* *Alexa van der Waall* *Paul van Wamelen* *Helena Verrill* *John Voight* *Mark Watkins* *Greg White*

Production Editors

Wieb Bosma *Claus Fieker* *Allan Steel* *Nicole Sutherland*

HTML Production

Claus Fieker *Allan Steel*

Copyright © 1994–2026 The University of Sydney. All rights reserved.

Contents

99 Modules over an Algebra and Group Representations 1
- 99.1 Introduction . . . 1
- 99.2 Constructions for A-Modules . . . 3
- 99.3 Constructions for $K[G]$-Modules . . . 3
- 99.4 New Modules from Old . . . 9
- 99.5 Accessing Module Information . . . 15
- 99.6 Group Representations . . . 17
- 99.7 Module Elements . . . 18
- 99.8 Submodules and Quotient Modules . . . 20
- 99.9 Properties of a Module . . . 23
- 99.10 Structure of a Module . . . 23
- 99.11 Characters and Character Tables . . . 30
- 99.12 Constructing All Irreducible $K[G]$-Modules . . . 31
- 99.13 Lattice of Submodules . . . 33
- 99.14 Homomorphisms . . . 37
- 99.15 Projective Indecomposable Modules . . . 43
- 99.16 Cohomology and Extensions . . . 46
- 99.17 Vertex and Source of an Indecomposable Module . . . 49
- 99.18 Bimodules . . . 50
- 99.19 Enumerating All Irreducible Modules . . . 53
- 99.20 Modules over a General Algebra . . . 61
- 99.21 Bibliography . . . 63

100 Characters of Finite Groups 64
- 100.1 Creation Functions . . . 64
- 100.2 The Table of Irreducible Characters . . . 65
- 100.3 Character Ring Operations . . . 68
- 100.4 Element Operations . . . 69
- 100.5 The Schur Index . . . 72
- 100.6 Constructions for Characters . . . 75
- 100.7 Brauer Characters . . . 82
- 100.8 Database of Character Tables . . . 85
- 100.9 Bibliography . . . 89

101 Representations of Symmetric Groups 90
- 101.1 Introduction . . . 90
- 101.2 Representations of the Symmetric Group . . . 90
- 101.3 Characters of the Symmetric Group . . . 92
- 101.4 Representations of the Alternating Group . . . 92
- 101.5 Characters of the Alternating Group . . . 92
- 101.6 Bibliography . . . 93

102 Mod P Galois Representations 94
- 102.1 Introduction . . . 94
- 102.2 φ-modules and Galois Representations in Magma . . . 95
- 102.3 Examples . . . 97

103 Group Representations **99**
103.1 Introduction . . . 99
103.2 Creation of Group Representations . . . 99
103.3 Basic Properties . . . 102
103.4 Operations on Group Representations . . . 102
103.5 Elements of Group Representations . . . 103
103.6 Homomorphisms of Group Representations . . . 104
103.7 Bibliography . . . 105

References **106**

Index **108**

Chapter 99

Modules over an Algebra and Group Representations

99.1 Introduction

Let A be an algebra over a field K and let M be a vector space over K. We say that M is a *(right) A-module* if for each $a \in A$ and $m \in M$, a product $ma \in M$ is defined such that

$$(m+n)a = ma + na, m(a+b) = ma + mb,$$

$$m(ab) = (ma)b, m1 = m,$$

$$m(ka) = (ma)k = (mk)a,$$

for all $a, b \in A, m, n \in M, k \in K$.

Recall that a *representation* of an algebra A over a field K is an algebra homomorphism of A into $\mathrm{Hom}_K(M, M)$, for some K-module M. Taking M to be an A-module, and defining a mapping $\phi : M \to M$ by

$$\phi(m) := ma \quad (a \in A, m \in M)$$

then it is an easy exercise to show that ϕ is a representation of A. A *matrix representation of degree n* of the algebra A is an algebra homomorphism of A into $M_n(K)$, the complete matrix algebra of degree n over K. Suppose M has finite K-dimension n and choose a basis for M. If, for each $a \in A$, we associate the matrix corresponding to the action of a on the basis elements, we obtain a matrix representation of A. Thus, each A-module of finite K-dimension *affords* a matrix representation of the algebra A. An important special case occurs when $A = K[G]$, the group algebra of a group G. In this case the theory of A-modules coincides with the theory of group representations.

Note that in all of the above cases A has finite K-dimension.

Throughout this chapter we shall use the term *A-module* when referring to modules as defined above. For Magma V, the A-modules M currently supported are those for which A is a matrix algebra defined over a finite field, the rational field, an algebraic number field, or an Euclidean domain. However, as currently the action of A may be used only in the construction of submodules, discussion will be mainly limited to the case in which the coefficient ring of M is a field.

The situation with $K[G]$-modules is more complicated. The group G must be a finite permutation or matrix group or a (soluble) group defined by a polycyclic presentation. At present K must be a field. However, most algorithms depend upon having an effective version of the Meataxe for each type of field. At present these only exist for finite fields, the rational field, and number fields. The functionality differs for each of these fields and so we give a brief summary of the features supported for each of the three field types.

K **is a finite field:** Because of the richness of the theory of modular representations a large number of intrinsics are provided and Magma has efficient algorithms for many of the calculations. The main features include:

- Construction of a $K[G]$-module from a variety of group actions

- Induction, restriction, inflation, etc for $K[G]$-modules

- Tensor products of various kinds, direct sums of $K[G]$-modules

- Highly optimised Meataxe for splitting modules over F_q

- Condensation used when splitting tensor products and induced modules etc

- Constituents, composition series, Jacobson radical, socle, socle series

- Submodues and quotient modules, enumerate all submodules

- Indecomposable test, direct sum decomposition, complements

- Hom spaces of various kinds, endomorphism ring, isomorphism test

- Construction of all F_q-irreducible modules for a group

- Construct the projective indecomposable modules for a group

- Cohomology and extensions of a $K[G]$-module

- Vertices and sources

- Bimodules

K **is** $\mathbb{Q}$ **or** $\mathbb{Q}(\alpha)$**:** This is actually two cases as there is a version of the Meataxe for modules over $\mathbb{Q}$ and a separate version for modules over extensions of $\mathbb{Q}$. For the rational field case irreducibles can be constructed from their characters provided that the degree is moderate and the group is not huge. In the number field case the degree of the field needs to be very small and the degree of modules arising in computations is only practical up to about a thousand as the splitting of such modules rapidly becomes very expensive with increasing degree. The available operations for $K[G]$-modules in characteristic zero are sketched below.

- Construction of a $K[G]$-module from a variety of group actions

- Induction, restriction, inflation, etc for $K[G]$-modules

- Tensor products, direct sums of $K[G]$-modules

- $K[G]$-module $\Longleftrightarrow$ representation; image and kernel

- Meataxes for splitting modules over $\mathbb{Q}$ and $\mathbb{Q}(\alpha)$

- Condensation used when splitting tensor products and induced modules etc

- Constituents, composition series, Jacobson radical, socle, socle series

- Indecomposable test, direct sum decomposition, complement

- Hom(M, N), endomorphism ring, isomorphism test

- Construction of an irreducible module from its rational (complex) character

- Construction of irreducible modules (for moderate sized groups)

K **is a field not discussed above:** As no Meataxe is available in Magma for such fields only a small number of intrinsics apply.

- Construction of a $K[G]$-module from a variety of group actions

- Induction, restriction, inflation etc for $K[G]$-modules

- Tensor products, direct sums of $K[G]$-modules

- $K[G]$-module $\Longleftrightarrow$ representation; image and kernel

Note that all of the A-module intrinsics apply to $K[G]$-modules.

99.2 Constructions for A-Modules

`RModule(A): AlgMat -> ModRng`

Given a subalgebra A of $M_n(K)$, create the right A-module M with underlying vector space $K^{(n)}$, where the action of $a \in A$ is given by $m * a$, $m \in M$.

`RModule(Q): [ MtrxS ] -> ModTupRng`

Given the subalgebra A of $M_n(K)$ generated by the terms of the sequence Q, create the right A-module M with underlying vector space $K^{(n)}$, where the action of $a \in A$ is given by $m * a$, $m \in M$.

Example: Module of Dimension 6 over a Finite Field `(ex-2aced1)`

We construct the 6-dimensional module over $\mathbf{F}_2$ with an action given by the matrices

$$\begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 \end{pmatrix} \qquad \begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

```
> A := MatrixAlgebra<GF(2), 6 |
>   [ 1,0,0,1,0,1,
>     0,1,0,0,1,1,
>     0,1,1,1,1,0,
>     0,0,0,1,1,0,
>     0,0,0,1,0,1,
>     0,1,0,1,0,0 ],
>   [ 0,1,1,0,1,0,
>     0,0,1,1,1,1,
>     1,0,0,1,0,1,
>     0,0,0,1,0,0,
>     0,0,0,0,1,0,
>     0,0,0,0,0,1 ] >;
> M := RModule(A);
> M;
RModule M of dimension 6 over GF(2)
```

99.3 Constructions for $K[G]$-Modules

The following functions provide for the construction of finite-dimensional $K[G]$-modules for a group G, where the action of G is given in terms of a matrix representation of G. Note that an Euclidean Domain may appear in place of the field K.

99.3.1 General $K[G]$-Modules

`GModule(G, A): Grp, AlgMat -> ModGrp`

Let G be a group defined on r generators, let K field and let A be a subalgebra of the matrix algebra $M_n(K)$, also defined by r non-singular matrices. It is assumed that the mapping from G to A defined by $\phi(G.i) \mapsto A.i$, for $i = 1, \ldots, r$, is a group homomorphism. Let M be an n-dimensional vector space over K. The function constructs a $K[G]$-module M of dimension n, where the action of the i-th generator of G on M is given by the i-th generator of A.

`GModule(G, Q): Grp, [ GrpMatElt ] -> ModGrp`

Let G be a group defined on r generators, let K be a field and let Q be a sequence of r invertible elements of $M_n(K)$ or $GL(n, K)$. It is assumed that the mapping from G to Q defined by $\phi(G.i) \mapsto$

$Q[i]$, for $i = 1, \ldots, r$, is a group homomorphism from G into the matrix algebra A defined by the terms of Q. The function constructs a $K[G]$-module M of dimension n, where the action of G is defined by the matrix algebra A.

`TrivialModule(G, K): Grp, Fld -> ModGrp`

Create the trivial $K[G]$-module for the group G.

Example: Three-Dimensional Module for PSL(2,7) (ex-93f8b4)

We construct a 3-dimensional module for $\mathrm{PSL}(2, 7)$ over $\mathbf{F}_2$. The action of the group on M is described in terms of two elements, x and y, belonging to the ring of 3×3 matrices over $\mathbf{F}_2$.

```
> PSL27 := PermutationGroup< 8 | (2,3,5)(6,7,8), (1,2,4)(3,5,6) >;
> S := MatrixAlgebra< FiniteField(2), 3 |
>        [ 0,1,0,  1,1,1,  0,0,1 ], [ 1,1,1,  0,1,1,  0,1,0 ] >;
> M := GModule(PSL27, S);
> M: Maximal;
GModule M of dimension 3 with base ring GF(2)
Generators of acting algebra:

[0 1 0]
[1 1 1]
[0 0 1]

[1 1 1]
[0 1 1]
[0 1 0]
```

Example: Module from a Conjugation Action (ex-64e699)

We write a function which, given a matrix algebra A, together with a matrix group G acting on A by conjugation (so A is closed under action by G), computes the G-module M representing the action of G on A. We then construct a particular (nilpotent) upper-triangular matrix algebra A, a group $G = 1 + A$ which acts on A, and finally construct the appropriate G-module M.

```
> MakeMod := function(A, G)
>     // Make G-module M of G acting on A by conjugation
>     k := CoefficientRing(A);
>     d := Dimension(A);
>     S := RMatrixSpace(A, k);
>     return GModule(
>         G,
>         [
>             MatrixAlgebra(k, d) |
>             &cat[
>                 Coordinates(S, S.j^g): j in [1 .. d]
>             ] where g is G.i: i in [1 .. Ngens(G)]
>         ]
>     );
> end function;
>
> MakeGroup := function(A)
>     // Make group G from upper-triangular matrix algebra A
>     k := CoefficientRing(A);
>     n := Degree(A);
>     return MatrixGroup<n, k | [Eltseq(1 + A.i): i in [1 .. Ngens(A)]]>;
> end function;
>
> k := GF(3);
```

(continues on next page)

(continued from previous page)

```
> n := 4;
> M := MatrixAlgebra(k, n);
> A := sub<M |
>      [0,2,1,1, 0,0,1,1, 0,0,0,1, 0,0,0,0],
>      [0,1,0,0, 0,0,2,2, 0,0,0,1, 0,0,0,0]>;
> G := MakeGroup(A);
> G;
MatrixGroup(4, GF(3)) of order 3^4
Generators:
    [1 2 1 1]
    [0 1 1 1]
    [0 0 1 1]
    [0 0 0 1]

    [1 1 0 0]
    [0 1 2 2]
    [0 0 1 1]
    [0 0 0 1]
> M := MakeMod(A, G);
> M: Maximal;
GModule M of dimension 5 over GF(3)
Generators of acting algebra:

[1 1 1 2 0]
[0 1 1 0 0]
[0 0 1 0 0]
[0 0 0 1 0]
[0 0 0 0 1]

[1 2 2 2 0]
[0 1 1 0 0]
[0 0 1 0 0]
[0 0 0 1 0]
[0 0 0 0 1]
```

99.3.2 Natural $K[G]$-Modules

The following functions provide for the construction of $K[G]$-modules for a group G in one of its natural actions. Note that an Euclidean Domain may be used in place of the field K.

GModule(G, K): GrpPerm, Rng -> ModGrp

Given a finite permutation group G and a field K, create the natural permutation module for G over K.

GModule(G): GrpMat -> ModGrp

Given a matrix group G defined as a subgroup of the group of units of the ring $Mat_n(K)$, where K is field, create the natural $K[G]$-module for G.

Example: Natural Module for M11 (ex-722ddf)

Given the Mathieu group M_{11} presented as a group of 5×5 matrices over $\mathbf{F}_3$, we construct the natural $K[G]$-module associated with this representation.

```
> G := MatrixGroup<5, FiniteField(3) |
>      [ 2,1,2,1,2,  2,0,0,0,2,  0,2,0,0,0,  0,1,2,0,1,  1,0,2,2,1],
>      [ 2,1,0,2,1,  1,2,0,2,2,  1,1,2,1,1,  0,2,0,1,1,  1,1,2,2,2] >;
> Order(G);
```

(continues on next page)

(continued from previous page)

```
7920
>
> M := GModule(G);
> M : Maximal;
GModule M of dimension 5 with base ring GF(3)
Generators of acting algebra:

[2 1 2 1 2]
[2 0 0 0 2]
[0 2 0 0 0]
[0 1 2 0 1]
[1 0 2 2 1]

[2 1 0 2 1]
[1 2 0 2 2]
[1 1 2 1 1]
[0 2 0 1 1]
[1 1 2 2 2]
```

99.3.3 Permutation Modules

The following functions provide for the construction of permutation modules for a group G. Note that an Euclidean Domain may be used in place of the field K.

`PermutationModule(G, H, K): Grp, Grp, Fld -> ModGrp`

Given a group G, a subgroup H of finite index in G and a field K, create the $K[G]$-module for G corresponding to the permutation action of G on the cosets of H.

`PermutationModule(G, K): Grp, Fld -> ModGrp`

Given a permutation group G and a field K, create the natural permutation module for G over K.

`PermutationModule(G, V): Grp, ModTupFld -> ModGrp`

Given a permutation group G of degree n and an n-dimensional vector space V defined over a field K, create the natural permutation module for G over K.

`PermutationModule(G, u): Grp, ModTupFldElt -> ModGrp`

Given a permutation group G of degree n, and a vector u belonging to the vector space $V = K^{(n)}$, where K is a field, construct the $K[G]$-module corresponding to the action of G on the K-subspace of V generated by the set of vectors obtained by applying the permutations of G to the vector u.

Example: Permutation Module for M12 `(ex-3c630c)`

We construct the permutation module for the Mathieu group M_{12} over the field $\mathbf{F}_2$.

```
> M12 := PermutationGroup<12 |
>       (1,2,3,4,5,6,7,8,9,10,11),
>       (1,12,5,2,9,4,3,7)(6,10,11,8) >;
> M := PermutationModule(M12, FiniteField(2));
> M : Maximal;
GModule M of dimension 12 with base ring GF(2)
Generators of acting algebra:

[0 1 0 0 0 0 0 0 0 0 0 0]
[0 0 1 0 0 0 0 0 0 0 0 0]
[0 0 0 1 0 0 0 0 0 0 0 0]
[0 0 0 0 1 0 0 0 0 0 0 0]
[0 0 0 0 0 1 0 0 0 0 0 0]
[0 0 0 0 0 0 1 0 0 0 0 0]
```

(continues on next page)

(continued from previous page)

```
[0 0 0 0 0 0 0 1 0 0 0 0]
[0 0 0 0 0 0 0 0 1 0 0 0]
[0 0 0 0 0 0 0 0 0 1 0 0]
[0 0 0 0 0 0 0 0 0 0 1 0]
[1 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 1]

[0 0 0 0 0 0 0 0 0 0 0 1]
[0 0 0 0 0 0 0 0 1 0 0 0]
[0 0 0 0 0 0 1 0 0 0 0 0]
[0 0 1 0 0 0 0 0 0 0 0 0]
[0 1 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 1 0 0]
[1 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 1 0 0 0 0 0 0]
[0 0 0 1 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 1 0]
[0 0 0 0 0 0 0 1 0 0 0 0]
[0 0 0 0 1 0 0 0 0 0 0 0]
```

Example: Constituent of a Permutation Module for A7 `(ex-cffd45)`

We construct the constituent of the permutation module for the alternating group of degree 7 that contains the vector $(1, 0, 1, 0, 1, 0, 1)$.

```
> A7 := AlternatingGroup(7);
> V  := VectorSpace(FiniteField(2), 7);
> x  := V![1,0,1,0,1,0,1];
> M  := PermutationModule(A7, x);
> M : Maximal;
GModule of dimension 6 with base ring GF(2)
Generators of acting algebra:

[1 0 1 0 0 0]
[0 0 1 0 1 0]
[0 1 1 0 0 0]
[0 0 1 0 0 0]
[0 0 1 1 0 0]
[0 0 1 0 0 1]

[0 0 0 0 0 1]
[0 1 0 0 0 0]
[1 0 0 0 0 0]
[0 0 0 1 0 0]
[0 0 0 0 1 0]
[0 0 1 0 0 0]
```

99.3.4 Action on an Elementary Abelian Section

`GModule(G, A, B): Grp, Grp, Grp -> ModGrp, Map`

`GModule(G, A): Grp, Grp -> ModGrp, Map`

Given a group G, a normal subgroup A of G, and a normal subgroup B of A such that the section A/B is elementary abelian of order p^n, create the $K[G]$-module M corresponding to the action of G on A/B, where K is the field $\mathbf{F}_p$. If B is trivial, it may be omitted. The function returns

(a) the module M; and

(b) the homomorphism $\phi : A/B \to M$.

Example: Module for the Wreath Product A4 wr C3 (ex-e47a86)

We construct a module M for the wreath product G of the alternating group of degree 4 with the cyclic group of degree 3. The module is given by the action of G on an elementary abelian normal subgroup H of order 64.

```
> G := WreathProduct(AlternatingGroup(4), CyclicGroup(3));
> G := PCGroup(G);
> A := pCore(G, 2);
> A;
GrpPC of order 64 = 2^6
Relations:
A.1^2 = Id(A),
A.2^2 = Id(A),
A.3^2 = Id(A),
A.4^2 = Id(A),
A.5^2 = Id(A),
A.6^2 = Id(A)
> M := GModule(G, A, sub<G|>);
> M;
GModule of dimension 6 with base ring GF(2)
```

99.3.5 Action on a Polynomial Ring

`GModule(G, P, d): Grp, RngMPol, RngIntElt -> ModGrp, Map, {@ RngMPolElt @}`

Let G be a permutation group of degree n or a matrix group of degree n over a field K, $P = K[x_1, \ldots, x_n]$ a polynomial ring in n variables, and d a non-negative integer. This function creates the $K[G]$-module M corresponding to the action of G on the space of homogeneous polynomials of degree d of the polynomial ring P. The function also returns the isomorphism f between the space of homogeneous polynomials of degree d of P and M, together with an indexed set of monomials of degree d of P which correspond to the columns of M.

`GModule(G, I, J): Grp, RngMPol, RngMPol -> ModGrp, Map, {@ RngMPolElt @}`

Let G be a permutation group of degree n or a matrix group of degree n over a field K, I an ideal of a multivariate polynomial ring $P = K[x_1, \ldots, x_n]$ in n variables over a field K, and J a zero-dimensional subideal of I. This function creates the $K[G]$-module M corresponding to the action of G on the finite-dimensional quotient I/J. The function also returns the isomorphism f between the quotient space I/J and M, together with an indexed set of monomials of P, forming a (vector space) basis of I/J, and which correspond to the columns of M.

`GModule(G, Q): Grp, RngMPolRes -> ModGrp, Map, {@ RngMPolElt @}`

Let G be a permutation group of degree n or a matrix group of degree n over a field K and $Q = I/J$ a finite-dimensional quotient ring of a multivariate polynomial ring $P = K[x_1, \ldots, x_n]$ in n variables over a field K. This function creates the $K[G]$-module M corresponding to the action of G on the finite-dimensional quotient Q. The function also returns the isomorphism f between the quotient ring Q and M, together with an indexed set of monomials of P, forming a (vector space) basis of Q, and which correspond to the columns of M.

Example: Representation on Homogeneous Polynomials (ex-2611b9)

Let T be the polynomial ring in five indeterminates over $GF(5)$. We create the representation of the alternating group of degree 5 that corresponds to its action on the space H_4 of homogeneous polynomials of degree 4 of T.

```
> G := Alt(5);
> R<[x]> := PolynomialRing(GF(5), 5);
> M, f := GModule(G, R, 4);
```

(continues on next page)

(continued from previous page)

```
> M;
GModule M of dimension 70 over GF(5)
```

Thus, the action of $Alt(5)$ on H_4 yields a 70-dimensional module. We find its irreducible constituents.

```
> Constituents(M);
[
    GModule of dimension 1 over GF(5),
    GModule of dimension 3 over GF(5),
    GModule of dimension 5 over GF(5)
]
> t := x[1]^4 + x[2]^4 + x[3]^4 + x[4]^4 + x[5]^4;
> v := f(t); v;
M: (1 0 0 0
```

`GTensorProduct(M, N, H): ModGrp, ModGrp, Grp -> ModGrp, Map`

Let M and N be two $K[G]$ modules where K is a finite field and let H be a subgroup of G. This function constructs the tensor product, $M \otimes_{KH} N$ as a $N_G(H)$-module. It is equal to `FixDualMod(TensorProduct(M,N), H)`.

99.4.2 Induction, Restriction, and Inflation for $K[G]$-Modules

`Dual(M): ModGrp -> ModGrp`

Given a $K[G]$-module M, where K is a field, construct the $K[G]$-module which is the K-dual, $\mathrm{Hom}_K(M, K)$, of M.

`Induction(M, G): ModGrp, Grp -> ModGrp`

Given a $K[H]$-module M, where K is a field, and a supergroup G of H, construct the $K[G]$-module obtained by inducing M up to G.

`Induction(R, G): Map, Grp -> Map`

Given a representation R of a subgroup of G, where R is defined over a field K, construct the representation of G obtained by inducing R up to G.

`Restriction(M, H): ModGrp, Grp -> ModGrp`

Given a $K[G]$-module M, where K is a field, and a subgroup H of G, form the $K[H]$-module corresponding to the restriction of M to the subgroup H.

`Inflation(M, h): ModGrp, Map -> ModGrp`

Given a $K[H]$-module M, where K is a field, and a group homomorphism $h : G \to H$, return M as a $K[G]$-module by inflation using h.

Example: Induction and Restriction for M22 (ex-6e84be)

Starting with the permutation module M over $\mathbf{F}_2$ for the Mathieu group M_{22}, we apply the induction and restriction functions to find new irreducible modules for M_{22}.

```
> SetSeed(1);
> G := PermutationGroup< 22 |
>         (1,2,4,8,16,9,18,13,3,6,12)(5,10,20,17,11,22,21,19,15,7,14),
>         (1,18,4,2,6)(5,21,20,10,7)(8,16,13,9,12)(11,19,22,14,17),
>         (1,18,2,4)(3,15)(5,9)(7,16,21,8)(10,12,20,13)(11,17,22,14) >;
> M := PermutationModule(G, GaloisField(2));
> M;
GModule M of dimension 22 with base ring GF(2)
> CM := Constituents(M);
> CM;
[
    GModule of dimension 1 over GF(2),
    GModule of dimension 10 over GF(2),
    GModule of dimension 10 over GF(2)
]
```

We compute the restriction N of the module M to the stabilizer $L34$ of a point in M_{22} and then induce two constituents of N back up to G.

```
> L34 := Stabilizer(G, 1);
> N := Restriction(M, L34);
> N;
GModule N of dimension 22 with base ring GF(2)
> CN := Constituents(N);
> CN;
[
    GModule of dimension 1 over GF(2),
```

(continues on next page)

(continued from previous page)

```
    GModule of dimension 9 over GF(2),
    GModule of dimension 9 over GF(2)
]
> Ind1 := Induction(CN[1], G);
> Ind1;
GModule Ind1 of dimension 22 over GF(2)
> Constituents(Ind1);
[
    GModule of dimension 1 over GF(2),
    GModule of dimension 10 over GF(2),
    GModule of dimension 10 over GF(2)
]
> Ind2 := Induction(CN[2], G);
> Ind2;
GModule Ind2 of dimension 198 over GF(2)
> Constituents(Ind2);
[
    GModule of dimension 1 over GF(2),
    GModule of dimension 10 over GF(2),
    GModule of dimension 10 over GF(2),
    GModule of dimension 34 over GF(2),
    GModule of dimension 98 over GF(2)
]
```

Thus, inducing up the 1-dimensional constituent of N gives us irreducible modules for G having the same dimensions as those appearing as constituents of M. However, inducing up a 9-dimensional module gives us irreducible modules of new dimensions: 34 and 98. Hence starting out with only the permutation module for M_{22} over $\mathbf{F}_2$, we have found 5 irreducible modules for the group.

99.4.3 The Fixed-point Spaces for a $K[G]$-Module

`Fix(M): Mod -> Mod`

Given a $K[G]$-module M, K a field, construct the largest submodule of M on which G acts trivially, i.e. the fixed-point space of M. This is equal to the cohomology group $H^0(G, M)$.

`FixMod(M, H): ModGrp, Grp -> Mod`

Given a $K[G]$-module M, K a field, and a subgroup H of G, return `Fix(M_H)` as a $N_G(H)$-module, where M_H is the restriction of M to H.

`FixDual(M): ModGrp -> Modgrp, Map`

Given a $K[G]$-module M, K a field, construct the largest quotient module of M on which G acts trivially, i.e. M/N with $N := \langle m * g - m | m \in M, g \in G \rangle$. This is equal to the homology group $H_0(G, M)$. The natural map from M to M/N is also returned.

`FixDualMod(M, H): ModGrp, Grp -> Mod`

Given a $K[G]$-module M, K a field, and a subgroup H of G, return `FixDual(M_H)` as a $N_G(H)$-module, where M_H is the restriction of M to H. The natural map from M to M/N is also returned.

99.4.4 Change Ring and Base Change

`ChangeRing(M, S): ModRng, Rng -> ModRng, Map`

Given an A-module M with base ring R, together with a ring S, such that there is a natural homomorphism from R to S, construct the module N with base ring S where N is obtained from M by coercing the components of the vectors of M into N. The corresponding homomorphism from M to N is returned as a second value.

`ChangeRing(M, S, f): ModRng, Rng, Map -> ModRng, Map`

Given an A-module M with base ring R, together with a ring S, and a homomorphism $f : R \to S$, construct the module N with base ring S, where N is obtained from M by applying f to the components of the vectors of M. The corresponding homomorphism from M to N is returned as a second value.

`M ^ T: ModGrp, AlgMatElt -> ModGrp`

Given a $K[G]$-module M of dimension n, where K is a field, and a nonsingular $n \times n$ matrix T over K, construct the $K[G]$-module N which corresponds to taking the rows of T as a basis for M.

99.4.5 Writing a $K[G]$-Module over a Smaller Field

Rewriting Over a Smaller Degree Finite Field

`IsRealisableOverSmallerField(M): ModGrp -> BoolElt, ModGrp`

Given a $K[G]$-module M, where K is a finite field, return true if M can be realised over a proper subfield F of K. The equivalent $F[G]$-module is also returned. The Glasby-Howlett algorithm is used to determine the smallest field over which M can be realised.

`IsRealisableOverSubfield(M, F): ModGrp, FldFin -> BoolElt, ModGrp`

Let M be a $K[G]$-module, where K is a finite field of characteristic p, and let F be a finite field also of characteristic p. If it is possible to realise M over the subfield F of K, return true and the equivalent $F[G]$-module.

`WriteOverSmallerField(M, F): ModGrp, FldFin -> ModGrp, Map`

Given a module M of dimension d over a finite field E having degree e and a subfield F of E having degree f, write the action of M as $d * e/f$ by $d * e/f$ matrices over F and return the module and the isomorphism.

`AbsoluteModuleOverMinimalField(M, F): ModGrp, FldFin -> ModGrp`

Let M be a $K[G]$-module, where K is a finite field of characteristic p, and let F be a finite field also of characteristic p. This function returns the module obtained by writing M over the smallest possible field containing F subject to the condition that the dimension of M does not increase. The Glasby-Howlett algorithm is used to determine the smallest field over which M can be realised.

`AbsoluteModuleOverMinimalField(M): ModGrp -> ModGrp`

```
Reduce      : Verbose                    Default: 2
Cohomology  : Verbose                    Default: 2
GrunwaldWang: Verbose                    Default: 2
```

Let M be a $K[G]$-module, where K is a finite field of characteristic p. This function returns the module obtained by writing M over a field of smallest possible degree subject to the condition that the dimension of M does not increase. For modules over finite fields, a field of smallest degree is always a subfield of K, in this case, the Glasby-Howlet algorithm is used.

`AbsoluteModulesOverMinimalField(Q, F): [ ModGrp ], FldFin -> [ ModGrp ]`

Let Q be a sequence of $K[G]$-modules, where K is a finite field of characteristic p, and let F be a finite field also of characteristic p. This function returns the sequence of modules obtained by writing each module M of Q over the smallest possible field containing F subject to the condition that the dimension of M does not increase. Thus, the effect of the function is to apply the function `AbsoluteModuleOverMinimalField` to each module of Q. The Glasby-Howlett algorithm is used to determine the smallest field over which the modules M of Q can be realised.

`ModuleOverSmallerField(M, F): ModGrp, FldFin -> ModGrp`

Let M be a $K[G]$-module of dimension d, where K is a finite field of characteristic p, and let F be a subfield of K of index n. This function returns the $F[G]$-module N obtained by writing the action of M as $dn \times dn$ matrices over F.

`ModulesOverSmallerField(Q, F): SeqEnum, FldFin -> ModGrp`

Let Q be a sequence of $K[G]$-modules, where K is a finite field of characteristic p, and let F be a subfield of K of index n. This function returns the sequence R of $F[G]$-modules obtained by applying the function `ModuleOverSmallerField` to each term of Q. That is, each term N of R is formed by writing the action of the corresponding term of Q as $dn \times dn$ matrices over F.

`ModulesOverCommonField(M, N): ModGrp, ModGrp -> ModGrp, ModGrp`

Given $K[G]$-modules M and N defined over finite fields of characteristic p, change their base fields to L, where L is the smallest field containing the base fields of M and N.

Rewriting Over a Smaller Degree Number Field

`AbsoluteModuleOverMinimalField(M): ModGrp -> ModGrp`

```
Reduce      : Verbose                    Default: 2
Cohomology  : Verbose                    Default: 2
GrunwaldWang: Verbose                    Default: 2
```

Let M be a $K[G]$-module, where K is a number field. This function returns the module obtained by writing M over a field F of smallest possible degree subject to the condition that the dimension of M does not increase. The field F is found using a combination of a method due to W. Plesken and a constructive version of the Grunwald-Wang theorem.

`Minimize(M): ModGrp -> ModGrp`

`Minimize(R): Map -> Map`

```
All         : BoolElt                    Default: false
Char        : AlgChtrElt                 Default: false
FindSmallest: BoolElt                    Default: false
Reduce      : Verbose                    Default: 2
Cohomology  : Verbose                    Default: 2
GrunwaldWang: Verbose                    Default: 2
```

Given an absolutely irreducible KG-module M or an absolutely irreducible representation $R : G \to \mathrm{GL}(n, K)$, where K is a number field, this function tries to find minimal subfields k of K that afford M (R), i.e. it tries to write the module (representation) over a smaller field. In general however, there might be a number field k not contained in K of smaller degree that affords M (R).

If `All` is set to `true`, then instead of returning a single module (representation) over a minimal degree subfield of K, a list of modules (representations) over all minimal subfields of K is returned instead.

If `Char` is given, it should be set to the character of the module (representation).

If `FindSmallest` is given, the field K will be extended by some auxiliary field A such that KA will contain a minimal degree field affording R. This involves a constructive version of the Grunwald-Wang theorem and can be computationally expensive if the degree of KA is too large.

`WriteGModuleOver(M, K): ModGrp, FldAlg -> ModGrp`

`WriteRepresentationOver(R, K): Map, FldAlg -> Map`

```
Char        : AlgChtrElt                 Default: false
Subfield    : BoolElt                    Default: false
Reduce      : Verbose                    Default: 2
Cohomology  : Verbose                    Default: 2
GrunwaldWang: Verbose                    Default: 2
```

Given an absolutely irreducible $L[G]$-module M or an absolutely irreducible representation $R : G \to GL(n, L)$, where L is a number field, and a number field K that is a normal extension, try to write M (R) over K.

If `Char` is specified, it should be set to the character of this module (representation).

If `Subfield` is set to `true`, the module (representation) will be rewritten over a minimal degree subfield of K.

Example: Minimal Field of a Q8 Representation (ex-217bd4)

We will work with the G-module and character of the unique 2-dimensional character of Q_8. It is well known that, while the character is defined over Q, the corresponding representation can only be defined over fields where -1 is the sum of 2 squares.

```
> _<x> := PolynomialRing(Rationals());
> G := TransitiveGroup(8, 5);
> TransitiveGroupDescription(G);
Q_8(8)
> R := AbsolutelyIrreducibleModules(G, Rationals());
> R;
[
    GModule of dimension 1 over Rational Field,
    GModule of dimension 1 over Rational Field,
    GModule of dimension 1 over Rational Field,
    GModule of dimension 1 over Rational Field,
    GModule of dimension 2 over Cyclotomic Field
    of order 4 and degree 2
]
> R := R[5];
> W := WriteGModuleOver(R, CyclotomicField(5));
> W;
GModule W of dimension 2 over Cyclotomic Field
of order 5 and degree 4
```

So $Q(\zeta_5)$ is an example of a field affording the module but having no minimal degree subfield (of degree 2 here) affording R!.

```
> M := AbsoluteModuleOverMinimalField(W);
> M;
GModule M of dimension 2 over Number Field with
defining polynomial x^2 + 10 over the Rational Field
```

Note that the base field returned here is of degree 2 but different from $Q(\zeta_4)$ that was found initially. In general there are infinitely many minimal degree splitting fields, and the specific one returned depends on the algorithm's internal choices.

If we try to realize R over a field where -1 cannot be written as a sums of two squares we get an error:

```
> WriteGModuleOver(R, QuadraticField(3));

>> WriteGModuleOver(R, QuadraticField(3));
                    ^
Runtime error in 'WriteGModuleOver': The G-module
cannot be realised over K
```

We can try to find a minimal field containing $Q(\sqrt{3})$ by computing the local Schur-indices and then obtain a splitting field:

```
> k := QuadraticField(3);
> si := SchurIndices(Character(R), k);
> si;
[ <1st place at infinity, 2>, <2nd place at infinity, 2> ]
```

(continues on next page)

(continued from previous page)

```
> A := SplittingField(si);
> A;
FldAb, defined by (<3>, [1      2])
of structure: Z/2
```

So the splitting field is returned as an abelian extension. We can see that A is of degree 2 over k and will be ramified at most at 3 and both infinite places. In order to use it to rewrite the module, we need to convert to a number field over Q first:

```
> A := NumberField(A);
> A;
Number Field with defining polynomial $.1^2 + 1 over k
> A := AbsoluteField(A);
> A;
Number Field A with defining polynomial x^4 - 4*x^2 +
16 over the Rational Field
> M := WriteGModuleOver(R, A);
> M;
GModule M of dimension 2 over A
> M := WriteGModuleOver(R, A : Subfield);
> M;
GModule M of dimension 2 over Number Field with
defining polynomial x^2 + 1 over the Rational Field
```

99.5 Accessing Module Information

This section deals with the underlying vector space of a module M, which is a module over the algebra A.

99.5.1 The Underlying Vector Space

`M . i: ModRng, RngIntElt -> ModElt`

Given an A-module M and a positive integer i, return the i-th generator of M.

`CoefficientRing(M): ModRng -> Rng`

`BaseRing(M): ModRng -> Rng`

Given an A-module M, where A is an algebra over the field K, return K.

`Generators(M): ModRng -> { ModRngElt }`

The generators for the A-module M, returned as a set.

`Parent(u): ModRngElt -> ModRng`

Given an element u belonging to the A-module M, return M.

99.5.2 The Action Algebra

`Action(M): ModRng -> AlgMat`

`RightAction(M): ModRng -> AlgMat`

Given an A-module M, return the matrix algebra A giving the action of A on M.

`MatrixGroup(M): ModGrp -> GrpMat`

```
Check: BoolElt                    Default: true
```

Given an $R[G]$-module M, return the matrix group whose generators are the (invertible) generators of the acting algebra of M.

ActionGenerator(M, i): ModRng, RngIntElt -> AlgMatElt

The i-th generator of the (right) acting matrix algebra for the module M.

NumberOfActionGenerators(M): ModRng -> RngIntElt

Ngens(M): ModTupRng -> RngIntElt

The number of action generators (the number of generators of the algebra) for the A-module M.

Group(M): ModGrp -> Grp

Given an $R[G]$-module M, return the group G.

Example: Access Functions for a Matrix Algebra Module (ex-891476)

We illustrate the use of several of these access functions by applying them to the 6-dimensional representation of a matrix algebra defined over $\mathbf{F}_2$.

```
> F2 := GF(2);
> F := MatrixAlgebra(F2, 6);
> A := sub< F |
>   [ 1,0,0,1,0,1,
>     0,1,0,0,1,1,
>     0,1,1,1,1,0,
>     0,0,0,1,1,0,
>     0,0,0,1,0,1,
>     0,1,0,1,0,0 ],
>   [ 0,1,1,0,1,0,
>     0,0,1,1,1,1,
>     1,0,0,1,0,1,
>     0,0,0,1,0,0,
>     0,0,0,0,1,0,
>     0,0,0,0,0,1 ] >;
> T := RModule(F2, 6);
> M := RModule(T, A);
> Dimension(M);
6
> BaseRing(M);
Finite field of size 2
```

We set R to be the name of the matrix ring associated with M. Using the generator subscript notation, we can access the matrices giving the (right) action of A.

```
> R := RightAction(M);
> R.1;
[1 0 0 1 0 1]
[0 1 0 0 1 1]
[0 1 1 1 1 0]
[0 0 0 1 1 0]
[0 0 0 1 0 1]
[0 1 0 1 0 0]
> R.2;
[0 1 1 0 1 0]
[0 0 1 1 1 1]
[1 0 0 1 0 1]
[0 0 0 1 0 0]
[0 0 0 0 1 0]
[0 0 0 0 0 1]
```

We display full details of the module.

```
> M: Maximal;
Module M of dimension 6 with base ring GF(2)
Generators of acting algebra:

[1 0 0 1 0 1]
[0 1 0 0 1 1]
[0 1 1 1 1 0]
[0 0 0 1 1 0]
[0 0 0 1 0 1]
[0 1 0 1 0 0]

[0 1 1 0 1 0]
[0 0 1 1 1 1]
[1 0 0 1 0 1]
[0 0 0 1 0 0]
[0 0 0 0 1 0]
[0 0 0 0 0 1]
```

99.6 Group Representations

A $K[G]$-module corresponds to a representation of G, that is, a homomorphism $phi : G \to GL(n, K)$. While the theory of representations is largely done using the language of $K[G]$-modules it is sometime useful to switch to the language of representations. This section describes intrinsics that enable the user to move from one language to the other.

GModuleAction(M): ModGrp -> Map(Hom)

Given a $K[G]$-module M, return the action of G on M as homomorphism f of G into the matrix group $GL_n(K)$.

Representation(M): ModGrp -> Map(Hom)

Given a $K[G]$-module M, return the action of G on M as homomorphism f of G into the matrix algebra $M_n(K)$.

Example: Characters from a Representation of A7 (ex-313ec4)

The function `Representation` allows the easy calculation of group characters. We illustrate this with the 6-dimension module for the group A_7 constructed above.

```
> A7 := AlternatingGroup(7);
> M  := PermutationModule(A7, Vector(GF(11), [1,0,1,0,1,0,1]));
> phi := Representation(M);
> [ Trace(phi(c[3])) : c in Classes(A7) ];
[ 7, 3, 4, 1, 1, 2, 0, 0, 0 ]
```

Example: Constructing the Dual Module (ex-8b546c)

We present a procedure which, given a $K[G]$-module M, constructs its dual D.

```
> DualModule := function(M)
>       G := Group(M);
>       f := Representation(M);
>       return GModule(G, [ Transpose(f(G.i))^-1 : i in [1 .. Ngens(G)] ]);
> end function;
```

Kernel(M): ModGrp -> Grp

Given a $K[G]$-module M, where K is a finite field, return the kernel of the group homomorphism defined by `Representation(M)`.

GModuleOfQuotient(M, H): ModGrp, Grp -> ModGrp

Given a $K[G]$-module M, where K is a finite field, and a subgroup H of the kernel of the representation afforded by M, return M as a (G/H)-module.

ActionGenerator(M, i): ModGrp, RngIntElt -> AlgMatElt

RightActionGenerator(M, i): ModGrp, RngIntElt -> AlgMatElt

The i-th generator of the (right) acting matrix algebra for the module M. That is, the image of the i-th group generator in the corresponding representation.

ActionGenerators(M): ModGrp -> [AlgMatElt]

Return the matrices giving the action on the module M as a sequence. These are the images of the generators of the group in the corresponding representation.

NumberOfActionGenerators(M): ModGrp -> RngIntElt

Nagens(M): ModGrp -> RngIntElt

The number of action generators (the number of generators of the algebra) for the $R[G]$-module M.

ActionGroup(M): ModGrp -> GrpMat

The matrix group generated by the action generators of M.

Sections(G): GrpMat -> List

Given a matrix group G defined over a finite field K, return the action of G on each composition factor of the natural K[G]-module for G.

Example: Composition Factors of a Tensor Square (ex-984a17)

We construct the tensor square T of the natural module M of the matrix group $G = SL(3, 5)$ and then determine the action of G on each composition factor of T.

```
> G := SL(3, 5);
> M := GModule(G);
> T := TensorProduct(M, M);
> A := ActionGroup(T);
> S := Sections(A);
> #S;
2
```

There are just two composition factors of T, the symmetric square and the exterior square of M.

```
> S[1];
MatrixGroup(3, GF(5))
Generators:

    [1 0 0]
    [0 2 0]
    [0 0 3]

    [0 1 0]
    [1 0 1]
    [1 0 0]
```

99.7 Module Elements

99.7.1 Construction

elt< M | a_1, ..., a_n >: ModRng, List -> ModRngElt

Given a module M with underlying vector space $K^{(n)}$, and elements $a_1, \ldots, a_n$ belonging to K,

construct the element $m = (a_1, \ldots, a_n)$ of M. Note that if m is not an element of M, an error will result.

`M ! Q: ModRng, [RngElt] -> ModRngElt`

Given the module M with underlying vector space K^n, and a sequence $Q = [a_1, \ldots, a_n]$ with universe K, construct the element $m = (a_1, \ldots, a_n)$ of M. Note that if m is not an element of M, an error will result.

`Zero(M): ModRng -> ModRngElt`

`M ! 0: ModRng, RngIntElt -> ModRngElt`

The zero element for the A-module M.

`Random(M): ModRng -> ModRngElt`

Given a module M defined over a finite ring or field, return a random vector.

99.7.2 Deconstruction of Module Elements

`ElementToSequence(u): ModRngElt -> [RngElt]`

`Eltseq(u): ModRngElt -> [RngElt]`

Given an element u belonging to the A-module M, return u in the form of a sequence Q of elements of K.

99.7.3 Action of the Algebra on the Module

`u * a: ModRngElt, AlgElt -> ModRngElt`

Given a vector u belonging to an A-module M, and an element $a \in A$ return the image of u under the action of a.

`u * g: ModGrpElt, GrpElt -> ModGrpElt`

Given a vector u belonging to an $K[G]$-module M, and an element g belonging to the group G, return the image of u under the action of $K[G]$ on the module M.

99.7.4 Arithmetic with Module Elements

`u + v: ModRngElt, ModRngElt -> ModRngElt`

Sum of the elements u and v, where u and v lie in the same A-module M.

`- u: ModRngElt -> ModRngElt`

Additive inverse of the element u.

`u - v: ModRngElt, ModRngElt -> ModRngElt`

Difference of the elements u and v, where u and v lie in the same A-module M.

`k * u: RngElt, ModRngElt -> ModRngElt`

Given an element u in an A-module M, where A is a K-algebra and an element $k \in K$, return the scalar product $k * u$ as an element of M.

`u * k: ModRngElt, RngElt -> ModRngElt`

Given an element u in an A-module M, where A is a K-algebra and an element $k \in K$, return the scalar product $u * k$ as an element of M.

`u / k: ModRngElt, RngElt -> ModRngElt`

Given an element u in an A-module M, where A is a K-algebra and a non-zero element $k \in K$, return the scalar product $u * (1/k)$ as an element of M.

99.7.5 Indexing

`u[i]: ModRngElt, RngIntElt -> RngElt`

Given an element u belonging to a submodule M of the R-module $R^{(n)}$ and a positive integer i, $1 \leq i \leq n$, return the i-th component of u (as an element of the ring R).

`u[i] := x: ModRngElt, RngIntElt, RngElt -> ModRngElt`

Given an element u belonging to a submodule M of the R-module $T = R^{(n)}$, a positive integer i, $1 \leq i \leq n$, and an element x of the ring R, redefine the i-th component of u to be x. The parent of u is changed to T (since the modified element u need not lie in M).

99.7.6 Properties of Module Elements

`IsZero(u): ModRngElt -> BoolElt`

Returns `true` if the element u of the A-module M is the zero element.

`Support(u): ModRngElt -> { RngIntElt }`

A set of integers giving the positions of the non-zero components of the vector u.

99.8 Submodules and Quotient Modules

99.8.1 Construction

`sub<M | L>: ModRng, List -> ModRng`

Given an A-module M, construct the submodule N generated by the elements of M specified by the list L. Each term L_i of the list L must be an expression defining an object of one of the following types:

(a) A sequence of n elements of R defining an element of M;

(b) A set or sequence whose terms are elements of M;

(c) A submodule of M;

(d) A set or sequence whose terms are submodules of M.

The generators stored for N consist of the elements specified by terms L_i together with the stored generators for submodules specified by terms of L_i. Repetitions of an element and occurrences of the zero element are removed (unless N is trivial).

The constructor returns the submodule N as an A-module together with the inclusion homomorphism $f : N \to M$.

`ImageWithBasis(X, M): ModMatRngElt, ModRng -> ModRng`

```
Check: BoolElt                    Default: true
```

Given a basis matrix X for a A-submodule of the A-module M, return the submodule N of M such that the morphism of N into M is X.

`Morphism(M, N): ModRng, ModRng -> ModMatRngElt`

If the A-module M was created as a submodule of the module N, return the inclusion homomorphism $\phi : M \to N$ as an element of $\mathrm{Hom}_A(M, N)$. Thus, ϕ gives the correspondence between elements of M (represented with respect to the standard basis of M) and elements for N.

Example: Submodule of a Permutation Module for L(3,4) (ex-260f18)

We construct a submodule of the permutation module for $L(3, 4)$ in its representation of degree 21.

```
> G := PSL(3, 4);
> M := PermutationModule(G, GF(2));
> M;
```

(continues on next page)

(continued from previous page)

```
GModule M of dimension 21 over GF(2)
> x := M![0,0,0,1,0,1,0,0,0,1,1,0,0,0,1,0,1,1,0,0,1];
> N := sub< M | x >;
> N:Maximal;
GModule N of dimension 9 over GF(2)
Generators of acting algebra:

[1 0 0 0 1 0 1 0 1]
[0 1 0 1 1 1 0 0 0]
[0 0 1 1 1 1 1 0 1]
[0 0 0 0 0 1 1 0 0]
[0 0 0 1 0 0 1 0 0]
[0 0 0 0 1 0 1 0 0]
[0 0 0 1 1 1 0 0 0]
[0 0 0 0 1 1 0 0 1]
[0 0 0 1 0 1 0 1 1]

[0 0 0 0 0 1 0 1 1]
[1 0 0 0 0 0 0 0 1]
[0 1 1 0 0 1 0 0 1]
[0 0 0 0 0 1 0 0 0]
[0 0 1 0 0 1 0 0 0]
[0 0 1 0 1 1 0 0 1]
[0 0 1 1 0 0 0 0 1]
[0 0 1 0 0 0 0 0 1]
[0 0 0 0 0 0 1 0 0]
```

Note that as a $\mathbf{F}_2$-module V has dimension 1, while as a $K[G]$-module it has dimension 9. The submodule N is defined on a reduced basis so we use `Morphism` to see N embedded in M.

```
> phi := Morphism(N, M);
> [ phi(x) : x in Basis(N) ];
[
    M: (1 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 1 0 1 0 1),
    M: (0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 1 1 1 1 1),
    M: (0 0 1 0 0 0 0 0 0 1 0 1 1 0 1 0 1 1 0 1 0),
    M: (0 0 0 1 0 0 0 0 0 0 1 0 1 1 0 1 0 1 1 0 1),
    M: (0 0 0 0 1 0 0 0 0 0 1 1 0 1 1 1 0 0 0 0 1 1),
    M: (0 0 0 0 0 1 0 0 0 1 0 0 1 1 1 1 1 0 1 0 0),
    M: (0 0 0 0 0 0 1 0 0 0 1 0 0 1 1 1 1 1 0 1 0),
    M: (0 0 0 0 0 0 0 1 0 0 0 1 0 0 1 1 1 1 1 0 1),
    M: (0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 1 0 1 0 1 1)
]
```

99.8.2 Membership and Equality

The operators described below refer to the underlying vector space.

`u in M: ModRngElt, ModRng -> BoolElt`

Returns `true` if the element u lies in the A-module M.

`N subset M: ModRng, ModRng -> BoolElt`

Returns `true` if the A-module N is contained in the A-module M.

`N eq M: ModRng, ModRng -> BoolElt`

Returns `true` if the A-modules N and M are equal, where N and M are contained in a common A-module.

99.8.3 Operations on Submodules

`M + N: ModRng, ModRng -> ModRng`

Sum of the submodules M and N, where M and N belong to a common A-module.

`M meet N: ModRng, ModRng -> ModRng`

Intersection of the submodules M and N, where M and N belong to a common A-module.

99.8.4 Quotient Modules

`quo<M | L>: ModRng, List -> ModRng`

Given an A-module M, construct the quotient module $P = M/N$ as an A-module, where N is the submodule generated by the elements of M specified by the list L. Each term L_i of the list L must be an expression defining an object of one of the following types:

(a) A sequence of n elements of R defining an element of M;

(b) A set or sequence whose terms are elements of M;

(c) A submodule of M;

(d) A set or sequence whose terms are submodules of M.

The generators constructed for N consist of the elements specified by terms L_i together with the stored generators for submodules specified by terms of L_i. The constructor returns the quotient module P as an A-module together with the natural homomorphism $f : M \to P$.

`Morphism(M, N): ModRng, ModRng -> ModMatRngElt`

If the A-module N was created as a quotient module of the module M, return the natural homomorphism $\phi : M \to N$ as an element of $\mathrm{Hom}_R(M, N)$. Thus ϕ gives the correspondence between elements of M and elements of N (represented with respect to the standard basis for N).

Example: Quotient of a Permutation Module for L(3,4) (ex-26899c)

We construct a quotient module of the permutation module for $L(3, 4)$ considered above.

```
> G := PSL(3, 4);
> M := PermutationModule(G, GF(2));
> x := M![0,0,0,1,0,1,0,0,0,1,1,0,0,0,1,0,1,1,0,0,1];
> N := sub< M | x >;
> N;
GModule N of dimension 9 over GF(2)
> Q, phi := quo< M | x >;
> Q;
GModule Q of dimension 12 over GF(2)
```

We locate the kernel of the epimorphism ϕ and check that it is the same as N.

```
> K :=  Kernel(phi);
GModule Ker of dimension 9 over GF(2)
> K eq N;
true
```

Given an element x in the codomain Q of the epimorphism ϕ, the value returned as the preimage of x is a representative element of the coset of the kernel that is the actual preimage of x. Since we are working in a module over a finite field, we can explicitly construct the full preimage `PreIm` of x.

```
> x := Q![0,0,0,1,1,0,0,0,0,1,0,0];
> PreIm := { x@@phi + k : k in K };
```

(continues on next page)

(continued from previous page)

```
> #PreIm;
512
```

99.9 Properties of a Module

The following intrinsics have differing conditions on their application so the user should check before using a given intrinsic.

IsIrreducible(M): ModRng -> BoolElt, ModRng, ModRng

Given an A-module M, defined over a finite field or a number field, the intrinsic returns **true** if and only if M is irreducible. If M is reducible, a proper submodule N of M together with the corresponding quotient module $Q = M/N$, are also returned.

IsAbsolutelyIrreducible(M): ModRng -> BoolElt, AlgMatElt, RngIntElt

Given an $K[G]$-module M where K is a finite field, the intrinsic return **true** if and only if M is absolutely irreducible. If M is reducible, a matrix algebra generator for the endomorphism algebra E of M (a field), as well as the dimension of E, are also returned.

IsDecomposable(M): ModRng -> BoolElt, ModRng, ModRng

Given an A-module M defined over a finite field or a number field, the intrinsic returns **true** if and only if M is decomposable. If M is decomposable and defined over a finite field, the function also returns proper submodules S and T of M such that $M = S \oplus T$.

IsSemisimple(M): ModGrp -> BoolElt

Given a $K[G]$-module M defined over a finite field or a number field, return **true** if M is semisimple and **false** otherwise. The function returns a second value listing the ranks of the primitive idempotents of the algebra. This is also a list of the multiplicities of composition factors in a composition series for M.

IsProjective(M): ModGrp -> BoolElt

Given an $K[G]$-module M, where K is a field, the intrinsic returns **true** if and only M is a projective $K[G]$-module.

IsFree(M): ModGrp -> BoolElt

Given an $K[G]$-module M, where K is a field, the intrinsic returns **true** if and only M is a free $K[G]$-module.

IsSelfDual(M): ModGrp -> BoolElt

Given an A-module M, return whether M is self-dual, that is, whether M is isomorphic to the dual of M.

IsPermutationModule(M): ModRng -> BoolElt

Given an $K[G]$-module M, the intrinsic returns **true** if and only if the generators of the matrix algebra giving the action of G are permutation matrices.

99.10 Structure of a Module

Some of the functions described in this section assume that the base ring is a **finite field**.

99.10.1 Splitting a Module

It is frequently desirable to determine whether an A-module M is irreducible and if not to return a proper submodule N of M together with the quotient module $Q = M/N$. This is achieved using the *Meataxe* algorithm. By repeated applications of the Meataxe, all of the irreducible constituents of M can be found.

`Meataxe(M): ModRng -> ModRng, ModRng, AlgMatElt`

Given an A-module M whose base ring either a finite field or number field, find a proper submodule N of M or else prove that M is irreducible. If a splitting of M is found, three values are returned:

(a) An A-module N corresponding to the induced action of A on S;

(b) An A-module P corresponding to the induced action of A on the quotient space M/N;

(c) Let ϕ, ν, and π denote the representations of A afforded by modules M, N, and P, respectively. The third value returned is an invertible matrix T which conjugates the matrices of $\phi(A)$ into reduced form. Specifically, if $a \in A$, then

$$T * \phi(a) * T^{-1} = \begin{pmatrix} \nu(a) & 0 \\ * & \pi(a) \end{pmatrix}$$

If M is proved to be irreducible, the function simply returns M. The fact that M is irreducible is recorded as part of the data structure for M.

`IsIrreducible(M): ModRng -> BoolElt, ModRng, ModRng`

Given an A-module M defined over a finite field or number field, this intrinsic returns `true` if and only if the A-module M is irreducible. If M is reducible, a proper submodule S of M together with the corresponding quotient module $Q = M/S$, are also returned.

`IsAbsolutelyIrreducible(M): ModRng -> BoolElt, AlgMatElt, RngIntElt`

Given an A-module M defined over a finite field, this intrinsic returns `true` if and only if the A-module M is absolutely irreducible. Also returned are a matrix algebra generator for the endomorphism algebra E of M (a field), as well as the dimension of E.

`AbsolutelyIrreducibleModule(M): ModRng -> ModRng`

Given an A-module M defined over a finite field K that is irreducible but not absolutely irreducible over K, this intrinsic returns an irreducible module N that is a constituent of the module M considered as a module over the splitting field for M. Note that the module N, while not unique, is absolutely irreducible.

`MinimalField(M): ModRng -> FldFin`

Let A be a matrix algebra over a finite field K. Given an A-module M over K, return the smallest subfield of K, over which M can be realised.

Example: Applying the Meataxe to O5(3) (ex-0a0c7d)

Consider the group $O_5(3)$ given as a permutation group of degree 45. We construct the permutation module, and we apply the Meataxe manually to find an irreducible constituent. (The splitting obtained by each application of the meataxe is random, and running this input repeatedly will produce different results.)

```
> SetSeed(1);
> O53 := PermutationGroup<45 |
>     (2,3)(4,6)(7,9)(8,11)(12,16)(13,14)(15,19)(18,22)(20,25)(21,26)(27,33)
>       (28,35) (29,34)(31,38)(36,43)(39,41),
>     (1,2,4,7,10,14,16,3,5)(6,8,12,17,21,27,34,41,44)(9,13,18,23,29,37,33,40,43)
>       (11,15,20)(19,24,30,25,31,22,28,36,38)(26,32,39)(35,42,45)>;
>
> P := PermutationModule(O53, GF(2));
> A := P;
> while not IsIrreducible(A) do
>     A, B := Meataxe(P); A; B;
> end while;
GModule A of dimension 14 over GF(2)
GModule B of dimension 31 over GF(2)
> A;
```

(continues on next page)

(continued from previous page)

```
GModule A of dimension 14 over GF(2)
> G := MatrixGroup(A); // Get matrix group from representation
> G: Minimal;
MatrixGroup(14, GF(2))
> time #G;
25920
> assert #G eq #O53 or Dimension(A) eq 1 and #G eq 1; // Group is simple
```

99.10.2 Composition Series

CompositionSeries(M): ModRng -> [ModRng], [ModRng], AlgMatElt

Given an A-module M defined over a finite field or number field, construct a composition series by repeatedly applying the Meataxe. The function returns three values:

(a) The composition series as a sequence of A-modules;

(b) The composition factors as a sequence of A-modules in the order determined by the composition series (a);

(c) A transformation matrix t such that for each $a \in A$, $t * a * t^{-1}$ is in reduced form.

CompositionFactors(M): ModRng -> [ModRng]

Given a matrix algebra A defined over a finite field or number field and an A-module M, construct the composition factors by repeatedly applying the Meataxe. The composition factors are returned in the form of a sequence of modules in the order determined by a composition series for M. If M is irreducible, the function returns a sequence containing M alone.

Constituents(M): ModRng -> [ModRng], [RngIntElt]

Given a matrix algebra A defined over a finite field or number field and an A-module M, construct the constituents of M, i.e., a sequence C of representatives for the isomorphism classes of composition factors of M. A sequence I of indices is also returned, so that i-th element of C is the $I[i]$-th composition factor of M.

ConstituentsWithMultiplicities(M): ModRng -> [<ModRng, RngIntElt>], [RngIntElt]

Given a matrix algebra A defined over a finite field or number field and an A-module M, return the constituents of M as a sequence C containing each constituent together with its multiplicity. A sequence I of indices is also returned, so that i-th term of C is the $I[i]$-th composition factor of M.

Example: Composition Series for O5(3) (ex-90fff5)

We continue with the $O_5(3)$ example from the previous section. We notice that the constituent of dimension of 8 is not absolutely irreducible, so we lift it to over an extension field.

```
> O53 := PermutationGroup<45 |
>     (2,3)(4,6)(7,9)(8,11)(12,16)(13,14)(15,19)(18,22)(20,25)(21,26)(27,33)
>       (28,35) (29,34)(31,38)(36,43)(39,41),
>     (1,2,4,7,10,14,16,3,5)(6,8,12,17,21,27,34,41,44)(9,13,18,23,29,37,33,40,43)
>       (11,15,20)(19,24,30,25,31,22,28,36,38)(26,32,39)(35,42,45)>;
>
> P := PermutationModule(O53, GaloisField(2));
> Constituents(P);
[
    GModule of dimension 1 over GF(2),
    GModule of dimension 6 over GF(2),
    GModule of dimension 8 over GF(2),
    GModule of dimension 14 over GF(2)
]
```

(continues on next page)

(continued from previous page)

```
> ConstituentsWithMultiplicities(P);
[
    <GModule of dimension 1 over GF(2), 3>,
    <GModule of dimension 6 over GF(2), 1>,
    <GModule of dimension 8 over GF(2), 1>,
    <GModule of dimension 14 over GF(2), 2>
]
> S, F := CompositionSeries(P);
> S, F;
[
    GModule of dimension 14 over GF(2),
    GModule of dimension 20 over GF(2),
    GModule of dimension 21 over GF(2),
    GModule of dimension 29 over GF(2),
    GModule of dimension 30 over GF(2),
    GModule of dimension 31 over GF(2),
    GModule P of dimension 45 over GF(2)
]
[
    GModule of dimension 14 over GF(2),
    GModule of dimension 6 over GF(2),
    GModule of dimension 1 over GF(2),
    GModule of dimension 8 over GF(2),
    GModule of dimension 1 over GF(2),
    GModule of dimension 1 over GF(2),
    GModule of dimension 14 over GF(2)
]
> IndecomposableSummands(P);
[
    GModule of dimension 1 over GF(2),
    GModule of dimension 44 over GF(2)
]
> C := Constituents(P);
> C;
[
    GModule of dimension 1 over GF(2),
    GModule of dimension 6 over GF(2),
    GModule of dimension 8 over GF(2),
    GModule of dimension 14 over GF(2)
]
> [IsAbsolutelyIrreducible(M): M in C];
[ true, true, false, true ]
> DimensionOfEndomorphismRing(C[3]);
2
> L := GF(2^2);
> E := ChangeRing(C[3], L);
> E;
GModule E of dimension 8 over GF(2^2)
> CE := CompositionFactors(E);
> CE;
[
    GModule of dimension 4 over GF(2^2),
    GModule of dimension 4 over GF(2^2)
]
> IsAbsolutelyIrreducible(CE[1]);
true
```

(continues on next page)

(continued from previous page)

```
> IsIsomorphic(CE[1], CE[2]);
false
```

99.10.3 Minimal and Maximal Submodules

Note that intrinsics `MinimalSubmodules` and `MinimalSubmodules` apply only in the case of finite fields as there are an infinite number of such submodules over fields of characteristic zero.

`JacobsonRadical(M): ModRng -> ModRng, Map`

Given a $K[G]$-module M defined over a finite field or a number field K, the Jacobson radical of M is returned.

`MinimalSubmodule(M): ModRng -> ModRng`

Given a $K[G]$-module M defined over a finite field or number field K, return a single minimal (or irreducible) submodule of M; if M is itself irreducible, M is returned.

`MinimalSubmodules(M): ModRng -> [ ModRng ], BoolElt`

```
Limit: RngIntElt                    Default: 0
```

Given a $K[G]$-module M defined over a finite field, return a sequence containing the minimal submodules of M. If the parameter `Limit` is given a positive integer value L, at most L submodules are calculated, and the second return value indicates whether all of the submodules are returned.

`MinimalSubmodules(M, F): ModRng, ModRng -> [ ModRng ], BoolElt`

```
Limit: RngIntElt                    Default: 0
```

Given a $K[G]$-module M defined over a finite field and an irreducible $K[G]$-module N, return a sequence containing those minimal submodules of M, each of which is isomorphic to N. If the parameter `Limit` is given a positive integer value L, at most L submodules are calculated, and the second return value indicates whether all of the submodules are returned.

`MaximalSubmodules(M): ModRng -> [ ModRng ], BoolElt`

```
Limit: RngIntElt                    Default: 0
```

Given a $K[G]$-module M defined over a finite field, return a sequence containing the maximal submodules of M. If a limit L is provided, only up L submodules are calculated, and the second return value indicates whether all of the submodules are returned.

99.10.4 Socle Series

`Socle(M): ModRng -> ModRng, Map`

Given a $K[G]$-module M defined over a finite field or number field K, return its socle, i.e. the direct sum of the minimal submodules of M.

`SocleRecursive(M): ModRng -> ModRng`

Given a $K[G]$-module M defined over a finite field or number field K, return its socle. This uses an algorithm due to Brooksbank and Luks.

`SocleSeries(M): ModRng -> [ ModRng ], [ ModRng ], AlgMatElt`

Given a $K[G]$-module M defined over a finite field or number field K, return a socle series S M, together with the socle factors corresponding to the terms of S and a matrix T giving the transformation of M into (semi-simple) reduced form. The socle series, as returned, does not include the trivial module but does include M.

`SocleFactors(M): ModRng -> [ ModRng ]`

Given a $K[G]$-module M defined over a finite field or number field K, return the factors corresponding to the terms of a socle series for M. The factors are returned in the form of a sequence of A-modules in the order determined by a socle series for M. If M is irreducible, the function returns a sequence containing M alone.

`SocleLayerFactors(M): ModGrp -> SeqEnum`

Given a KG-module M (usually a projective indecomposable module) defined over a finite field, this intrinsic determines the composition factors of the socle layers of M. Here a layer is the quotient of two adjacent terms of the socle series of M. At present computation of the socle series employs a naive algorithm but this will be replaced by one that employs condensation.

`DisplaySocleStructure(SL): SeqEnum`

The intrinsic, `DisplaySocleStructure`, takes the output of `SocleLayerFactors` and displays it in "Christmas tree" style.

Example: Minimal Submodules for O5(3) (ex-eb70be)

We continue with the $O_5(3)$ example from the previous section.

```
> O53 := PermutationGroup<45 |
>   (2,3)(4,6)(7,9)(8,11)(12,16)(13,14)(15,19)(18,22)(20,25)(21,26)(27,33)
>     (28,35) (29,34)(31,38)(36,43)(39,41),
>   (1,2,4,7,10,14,16,3,5)(6,8,12,17,21,27,34,41,44)(9,13,18,23,29,37,33,40,43)
>     (11,15,20)(19,24,30,25,31,22,28,36,38)(26,32,39)(35,42,45)>;
>
> P := PermutationModule(O53, FiniteField(2));
> MaximalSubmodules(P);
[
    GModule of dimension 31 over GF(2),
    GModule of dimension 44 over GF(2)
]
> JacobsonRadical(P);
GModule of dimension 30 over GF(2)
> MinimalSubmodules(P);
[
    GModule of dimension 1 over GF(2),
    GModule of dimension 14 over GF(2)
]
> Soc := Socle(P);
> Soc;
GModule Soc of dimension 15 over GF(2)
> SocleSeries(P);
[
    GModule of dimension 15 over GF(2),
    GModule of dimension 22 over GF(2),
    GModule of dimension 30 over GF(2),
    GModule of dimension 31 over GF(2),
    GModule P of dimension 45 over GF(2)
]
> SocleFactors(P);
[
    GModule of dimension 15 over GF(2),
    GModule of dimension 7 over GF(2),
    GModule of dimension 8 over GF(2),
    GModule of dimension 1 over GF(2),
    GModule of dimension 14 over GF(2)
]
```

99.10.5 Decomposition and Complements

The functions in this section currently apply only in the case in which A is an algebra over a finite field.

`IsDecomposable(M): ModRng -> BoolElt, ModRng, ModRng`

Given a $K[G]$-module M defined over a finite field or number field K, return `true` if M is decomposable and `false` otherwise. If M is decomposable, the function also returns proper submodules S and T of M such that $M = S \oplus T$.

`IsSemisimple(M): ModGrp -> BoolElt`

Given a $K[G]$-module M defined over a finite field or number field K, return `true` if M is semisimple and `false` otherwise. The function returns a second value listing the ranks of the primitive idempotents of the algebra. This is also a list of the multiplicities of composition factors in a composition series for M.

`DirectSumDecomposition(M): ModRng -> [ ModRng ]`

`IndecomposableSummands(M): ModRng -> [ ModRng ]`

`Decomposition(M): ModRng -> [ ModRng ]`

Given a $K[G]$-module M defined over a finite field or number field K, return a sequence Q of indecomposable summands of M. Each element of Q is an indecomposable submodule of M and M is equal to the (direct) sum of the terms of Q. If M is indecomposable, the sequence Q consists of M alone.

`RelativeDecomposition(M, T): ModRng, ModRng -> ModRng, ModRng`

Given an A-module M defined over a finite field, return the direct sum decomposition of M into A-modules N and K where $M = N + K$, and K is minimal with respect to containing the A-module T. This uses an algorithm due to Brooksbank and Luks.

`HasComplement(M, S): ModGrp, ModGrp -> BoolElt, ModGrp`

`HasComplement(M, S): ModRng, ModRng -> BoolElt, ModRng`

`IsDirectSummand(M, S): ModGrp, ModGrp -> BoolElt, ModGrp`

`IsDirectSummand(M, S): ModRng, ModRng -> BoolElt, ModRng`

Given a $K[G]$-module M defined over a finite field or number field K, and a submodule S of M, determine whether S has a $K[G]$-invariant complement in M. If this is the case, the value `true` is returned together with a submodule T of M such that $M = S \oplus T$; otherwise the value `false` is returned.

`Complements(M, S): ModGrp, ModGrp -> [ ModGrp ]`

`Complements(M, S): ModRng, ModRng -> [ ModRng ]`

Given a $K[G]$-module M defined over a finite field or number field K, and a submodule S, return all $K[G]$-invariant complements of S in M.

Example: Decomposition of a Matrix Algebra Module (ex-6acaf1)

```
> A := MatrixAlgebra<GF(2), 6 |
>   [ 1,0,0,1,0,1,
>     0,1,0,0,1,1,
>     0,1,1,1,1,0,
>     0,0,0,1,1,0,
>     0,0,0,1,0,1,
>     0,1,0,1,0,0 ],
>   [ 0,1,1,0,1,0,
>     0,0,1,1,1,1,
>     1,0,0,1,0,1,
>     0,0,0,1,0,0,
>     0,0,0,0,1,0,
>     0,0,0,0,0,1 ] >;
```

(continues on next page)

(continued from previous page)

```
> M := RModule(RSpace(GF(2), 6), A);
> M;
RModule M of dimension 6 over GF(2)
> IsDecomposable(M);
false
> MM := DirectSum(M, M);
> MM;
RModule MM of dimension 12 over GF(2)
> l, S, T := IsDecomposable(MM);
> l;
true;
> S;
RModule S of dimension 6 over GF(2)
> HasComplement(MM, S);
true
> Complements(MM, S);
[
    RModule of dimension 6 over GF(2),
    RModule of dimension 6 over GF(2)
]
> IndecomposableSummands(MM);
[
    RModule of dimension 6 over GF(2),
    RModule of dimension 6 over GF(2)
]
> Q := IndecomposableSummands(MM);
> Q;
[
    RModule of dimension 6 over GF(2),
    RModule of dimension 6 over GF(2)
]
> Q[1] meet Q[2];
RModule of dimension 0 over GF(2)
> Q[1] + Q[2];
RModule MM of dimension 12 over GF(2)
```

99.11 Characters and Character Tables

Facilities for working with group characters can be found in Chapter *Characters of Finite Groups*. In this section we repeat a small number of character intrinsics that are used frequently when computing with $K[G]$-modules.

99.11.1 Ordinary Characters

In this section the various intrinsics for computing the table of absolutely irreducible complex characters for a finite group are described.

Character(M): ModGrp -> AlgChtrElt

Given a $K[G]$-module M where K is the field of rationals or a number field, the character for M over the field K is returned.

CharacterTable(G): Chtr -> SeqEnum

The table of irreducible complex characters for the group G is constructed.

SymmetricCharacterTable(n): RngIntElt -> SeqEnum

`AlternatingCharacterTable(n): RngIntElt -> SeqEnum`

The table of irreducible complex characters for the symmetric (alternating) group of degree n is constructed.

`RationalCharacterTable(G): Chtr -> SeqEnum`

The table of irreducible rational characters for the group G is constructed.

99.11.2 Brauer Characters

In this section the intrinsic for computing the table of absolutely irreducible Brauer characters of a finite group are described.

`BrauerCharacterTable(G, p): Chtr -> SeqEnum`

Construct the table of irreducible Brauer characters in characteristic p for the group G. For soluble groups this is deduced from the ordinary character table. For non-soluble groups the absolutely irreducible p-modular representations are constructed.

`Character(M): ModGrp -> AlgChtrElt`

Given a $K[G]$-module M, where K is a finite field of characteristic p a prime, the p-modular Brauer character of M is constructed.

99.12 Constructing All Irreducible $K[G]$-Modules

99.12.1 Constructing Irreducible $K[G]$-Modules (Characteristic p)

In this section we describe how to obtain irreducible $K[G]$-modules for a group G where K is a finite field. In this case, it is currently necessary to construct all irreducible modules in order to obtain a particular irreducible. This section gives brief descriptions of the more useful intrinsics. Much more information is given in the section *Enumerating All Irreducible Modules*.

`IrreducibleModules(G, K : parameters): Grp, Fld -> SeqEnum`

Let G be a permutation group, a matrix group, or a group defined by a power-commutator presentation and let K be a finite field. This intrinsic constructs all of the irreducible $K[G]$-modules for G over K. Thus, some of the irreducibles returned may not be absolutely irreducible. Much more information about the enumeration of all irreducibles can be found in Section *Enumerating All Irreducible Modules*.

`AbsolutelyIrreducibleModules(G, K : parameters): Grp, Fld -> SeqEnum`

Let G be a permutation group, a matrix group, or a group defined by a power-commutator presentation and let K be a finite field. This intrinsic constructs all of the absolutely irreducible $K[G]$-modules. Each irreducible is returned over a field which is the minimal field for which it is absolutely irreducible. Much more information about the enumeration of all irreducibles can be found in Section *Enumerating All Irreducible Modules*.

`AbsolutelyIrreducibleModulesSoluble(G, p): Grp, RngIntElt -> SeqEnum`

Given a finite *soluble group* G and a prime p dividing the order of G, return the absolutely irreducible $K[G]$-modules for G in characteristic p. The method works its way up a central series finding the modules at each level using induction and restriction.

`AbsolutelyIrreducibleModulesSchur(G, K): GrpPC, Fld -> List`

Let G be a finite *soluble group* defined by a pc-presentation and let K be a finite field. The function constructs all absolutely irreducible representations of G over appropriate extensions or subfields of the field K and returns them as a list. The Glasby-Howlett algorithm is used to determine a minimal field over which an irreducible module may be realised.

Example: Irreducible Modules for PSL(3,4) in Characteristic 3 `(ex-040cef)`

The irreducible modules for $PSL(3,4)$ in characteristic 3 are constructed.

```
> G := PSL(3, 4);
> irrs := IrreducibleModules(G, GF(3));
> irrs;
[
    GModule of dimension 1 over GF(3),
    GModule of dimension 15 over GF(3),
    GModule of dimension 15 over GF(3),
    GModule of dimension 15 over GF(3),
    GModule of dimension 19 over GF(3),
    GModule of dimension 90 over GF(3),
    GModule of dimension 126 over GF(3)
]
```

99.12.2 Constructing Irreducible $K[G]$-Modules (Characteristic Zero)

In this section we describe how to obtain irreducible $K[G]$-modules for a group G where K is a characteristic zero field. In contrast to the positive characteristic case, Magma can construct just the irreducible that affords a given character. This section gives brief descriptions of the more useful intrinsics. Much more information is given the section *Enumerating All Irreducible Modules*.

`GModule(x): Chtr -> ModGrp`

Let G be a finite group and let x either a rational character or a complex character. This intrinsic constructs the irreducible $K[G]$-module that affords the character x.

`IrreducibleModules(G, K): Grp, Fld -> SeqEnum`

Let G be a finite group and let K be the rational field or a simple number field (including quadratic and cyclotomic fields). This intrinsic constructs all of the irreducible $K[G]$-modules for G over K. Thus, some of the irreducibles returned may not be absolutely irreducible. If the absolute irreducibles over a finite field are required then the intrinsic `AbsolutelyIrreducibleModules` should be used.

`AbsolutelyIrreducibleModules(G): Grp -> SeqEnum`

Let G be a finite group. This intrinsic constructs all of the absolutely irreducible G-modules in characteristic zero, which are written over the rational field or a simple number field. Each irreducible is also returned over a field which is the minimal field for it to be absolutely irreducible.

`AbsolutelyIrreducibleModulesSchur(G, K): GrpPC, Fld -> List`

Let G be a finite *soluble group* defined by a pc-presentation and let K be the rational field or a cyclotomic field. The order of a cyclotomic field must divide the exponent of G. The function constructs all absolutely irreducible representations of G over appropriate extensions or subfields of the field K and returns them as a list. The field over which an irreducible module is given may not be minimal. See Section *Enumerating All Irreducible Modules* for more information.

Example: Irreducible Modules for PSL(3,4) over the Rationals `(ex-b6fa49)`

The irreducible modules for $G = \mathrm{PSL}(3,4)$ over the field $\mathbb{Q}$ are constructed. Then the absolutely irreducible modules for G are constructed.

```
> G := PSL(3, 4);
> irrs := IrreducibleModules(G, Rationals());
> irrs;
[
    GModule of dimension 1 over Rational Field,
    GModule of dimension 20 over Rational Field,
    GModule of dimension 35 over Rational Field,
    GModule of dimension 35 over Rational Field,
    GModule of dimension 35 over Rational Field,
    GModule of dimension 64 over Rational Field,
```

(continues on next page)

(continued from previous page)

```
    GModule of dimension 90 over Rational Field,
    GModule of dimension 126 over Rational Field
]
> absirrs := AbsolutelyIrreducibleModules(G);
> absirrs;
[
    GModule of dimension 1 over Rational Field,
    GModule of dimension 20 over Rational Field,
    GModule of dimension 35 over Rational Field,
    GModule of dimension 35 over Rational Field,
    GModule of dimension 35 over Rational Field,
    GModule of dimension 45 over Number Field with defining
    polynomial x^2 - x + 2 over the Rational Field,
    GModule of dimension 45 over Number Field with defining
    polynomial x^2 - x + 2 over the Rational Field,
    GModule of dimension 63 over Number Field with defining
    polynomial x^2 - x - 1 over the Rational Field,
    GModule of dimension 63 over Number Field with defining
    polynomial x^2 - x - 1 over the Rational Field,
    GModule of dimension 64 over Rational Field
]
```

99.12.3 The Cartan Matrix

`CartanMatrix(G, K): Grp, FldFin -> AlgMatElt`

Let G be a finite group and let K be a finite field. This intrinsic constructs the Cartan matrix C for G over K. Let k be the number of irreducible $K[G]$-modules. The Cartan matrix C for G over K is a $k \times k$ matrix of integers, in which the entry C_{ij} is equal to the number of times that the j-th irreducible $K[G]$-module is a constituent of the i-th projective indecomposable $K[G]$-module. This can be computed quickly from the Brauer characters of the irreducible $K[G]$-modules. (Note that, unlike the absolute Cartan matrix discussed below, C need not be symmetric.)

`AbsoluteCartanMatrix(G, K): Grp, FldFin -> AlgMatElt`

Let G be a finite group and let K be a finite field. This intrinsic constructs the Cartan matrix C for G over an extension L of K that is large enough to ensure that all irreducible $L[G]$-modules are absolutely irreducible. Its rows and columns correspond to the $K[G]$-modules returned by `AbsolutelyIrreducibleModules(G,K)`. The matrix C is symmetric and has integer entries. It is equal to the Cartan matrix for G in the characteristic p of K, as defined in textbooks on modular representation theory.

`DecompositionMatrix(G, K): Grp, FldFin -> AlgMatElt`

Let G be a finite group and let K be a finite field. This intrinsic constructs the decomposition matrix C for G in the characteristic p of K, The entry D_{ij} is equal to the number of times that the j-th absolutely irreducible $K[G]$-module occurs as a constituent of the i-th ordinary irreducible G-module over the complex numbers reduced modulo p. Note that $D^{\mathrm{T}}D$ is equal to the absolute Cartan matrix.

99.13 Lattice of Submodules

Let M be an A-module. Magma can construct the lattice L of all submodules of M if this is not too large. Various properties of the lattice L may then be examined. The elements of L are called *submodule-lattice elements* and are numbered from 1 to n where n is the cardinality of L. Once the lattice has been constructed, the result of various lattice operations, such as meet and intersection, are available without the need for any module-theoretic calculation. Certain information about M and its submodules may then be obtained by analyzing L. Given an element of L, one can easily create the submodule N of M

corresponding to it and one can also create the element of L corresponding to any submodule of M. The functions is this section apply only in the case in which A is an algebra over a finite field.

99.13.1 Creating Lattices

SubmoduleLattice(M): ModRng -> SubModLat, BoolElt

```
Limit           : RngIntElt                    Default: 0
CodimensionLimit: RngIntElt                    Default:
```

Given a $K[G]$-module M defined over F_q, construct the lattice L of submodules of M. If the parameter `Limit` is set to n, at most n submodules are calculated, and the second return value indicates whether the returned lattice L is the full lattice of submodules of M.

SubmoduleLatticeAbort(M, n): ModRng, RngIntElt -> BoolElt, SubModLat

Given a $K[G]$-module M defined over F_q and a positive integer n, construct the lattice L of submodules of M, provided that the number of submodules does not exceed n. In this case the value `true` and the lattice L are returned. If M has more than n submodules, the function aborts and returns the value `false`.

SetVerbose("SubmoduleLattice", i): MonStgElt, RngIntElt

Control verbose printing for the submodule lattice algorithm. The level i can be 2 for maximal printing or 1 for moderate printing. The algorithm works down a composition series of the module and a summary is printed for each level.

Submodules(M): ModRng -> [ModRng]

```
CodimensionLimit: RngIntElt                    Default: Dimension(M)
```

Given a $K[G]$-module M defined over F_q, return a sequence containing all the submodules of M sorted by dimension.

Example: Creating a Submodule Lattice (ex-05bd83)

We create the lattice of submodules for the A-module $\mathbf{F}_3[\mathbb{Z}_6]$ with level 1 verbose printing turned on.

```
> M := PermutationModule(CyclicGroup(6), GF(3));
> SetVerbose("SubmoduleLattice", 1);
> L := SubmoduleLattice(M);
Submodule Lattice; Dimension: 6, Composition length: 6
Starting level 4; Current number of modules: 2
Starting level 3; Current number of modules: 3
Starting level 2; Current number of modules: 6
Starting level 1; Current number of modules: 9
Starting level 0; Current number of modules: 12
Change basis time: 0.010
Jacobson radical time: 0.060
Complement time: 0.070
Total time: 0.250
> #L;
16
```

99.13.2 Operations on Lattices

In the following, L is the lattice of submodules for a module M.

 # L: SubModLat -> RngIntElt

The cardinality of L, i.e. the number of submodules of M.

`L ! i: SubModLat, RngIntElt -> SubModLatElt`

Create the i-th element of the lattice L. The number i is insignificant (i.e. the elements of L are not numbered in any special way), but this allows one to uniquely identify each element of the lattice L.

`L ! S: SubModLat, ModRng -> SubModLatElt`

Create the element of the lattice L corresponding to the submodule S of M.

`Bottom(L): SubModLat -> SubModLatElt`

Create the bottom of the lattice L, i.e. the element of L corresponding to the zero-submodule of M. If the lattice was created with a limit on the number of submodules and the lattice is partial, the bottom of the lattice may not be the zero submodule.

`Random(L): SubModLat -> SubModLatElt`

Create a random element of L.

`Top(L): SubModLat -> SubModLatElt`

Create the top of the lattice L, i.e. the element of L corresponding to M.

99.13.3 Operations on Lattice Elements

In the following, L is the lattice of submodules for a module M. Elements of L are identified with the integers $[1..\#L]$ but not in any particular order.

`IntegerRing() ! e: RngInt, SubModLatElt -> RngIntElt`

The integer corresponding to lattice element e.

`e + f: SubModLatElt, SubModLatElt -> SubModLatElt`

The sum of lattice elements e and f, i.e. the lattice element corresponding to the sum of the modules corresponding to e and f.

`e meet f: SubModLatElt, SubModLatElt -> SubModLatElt`

The intersection of lattice elements e and f.

`e eq f: SubModLatElt, SubModLatElt -> SubModLatElt`

Returns `true` if and only if lattice elements e and f are equal.

`e subset f: SubModLatElt, SubModLatElt -> SubModLatElt`

Returns `true` if and only if e is under f in the lattice L, i.e. the submodule corresponding to e is a submodule of the submodule corresponding to f.

`MaximalSubmodules(e): SubModLatElt -> { SubModLatElt }`

The maximal submodules of e, returned as a set of lattice elements.

`MinimalSupermodules(e): SubModLatElt -> { SubModLatElt }`

The minimal supermodules of e, returned as a set of lattice elements.

`Module(e): SubModLatElt -> ModRng`

The submodule of M corresponding to the element e of the lattice L.

99.13.4 Properties of Lattice Elements

`Dimension(e): SubModLatElt -> RngIntElt`

The dimension of the submodule of M corresponding to e.

`JacobsonRadical(e): SubModLatElt -> SubModLatElt`

The Jacobson radical of e, i.e. the lattice element corresponding to the Jacobson radical of the submodule corresponding to e.

`Morphism(e): SubModLatElt -> ModMatRngElt`

The morphism from the module corresponding to e to M.

Example: Operations in a Submodule Lattice (ex-42abf7)

We create the lattice of submodules for the A-module $\mathbf{F}_3[\mathbb{Z}_6]$.

```
> SetSeed(1);
> M := PermutationModule(CyclicGroup(6), GF(3));
> L := SubmoduleLattice(M);
> #L;
16
> T := Top(L);
> B := Bottom(L);
> T;
16
> B;
1
> // Check that element of L corresponding to M is T
> L ! M;
1
> (L ! M) eq T;
true
> // Check that module corresponding to B is zero-submodule of M
> Module(B);
GModule of dimension 0 with base ring GF(3)
```

We next find the minimal supermodules (immediate parents) of B in L and then determine the actual A-submodules to which they correspond.

```
> S := MinimalSupermodules(B);
> S;
{ 2, 3 }
> Module(L ! 2);
GModule of dimension 1 with base ring GF(3)
> Module(L ! 3);
GModule of dimension 1 with base ring GF(3)
> Dimension(L ! 2);
1
> Morphism(L ! 2);
[1 1 1 1 1 1]
> Morphism(L ! 3);
[1 2 1 2 1 2]
> // Set A to the sum of these elements
> A := L!2 + L!3;
> A;
5;
> // Note that A has dimension 2 and its morphism is the sum of the previous
> Dimension(A);
2
> Morphism(A);
[1 0 1 0 1 0]
[0 1 0 1 0 1]
> MaximalSubmodules(A);
{ 2, 3}
> S!2 subset A;
true
```

We now find the maximal submodules of L, and examine one, S, in detail.

```
> MaximalSubmodules(T);
```

(continues on next page)

(continued from previous page)

```
{ 14, 15 }
> A := L ! 14;
> Dimension(A);
5
> Morphism(A);
[1 0 0 0 0 1]
[0 1 0 0 0 2]
[0 0 1 0 0 1]
[0 0 0 1 0 2]
[0 0 0 0 1 1]
> S := Module(A);
> S;
GModule S of dimension 5 with base ring GF(3)
```

Finally, we compute the Jacobson radical of S directly, and also obtain it from the lattice, checking that the two methods match.

```
> J := JacobsonRadical(S);
> J;
GModule J of dimension 3 with base ring GF(3)
> L ! J;
8
> JacobsonRadical(A);
8
```

99.14 Homomorphisms

Let M and N be A-modules where A is an algebra defined over a field K. Then $\mathrm{Hom}_A(M, N)$ consists of all K-homomorphisms from M to N which commute with the action of A. The type of such (matrix) homomorphisms, called *A-homs*, is `ModMatGrpElt`.

99.14.1 Creating Homomorphisms and Hom Spaces

`hom< M -> N | X >: ModRng, ModRng, ModMatElt -> Map`

Given A-modules M and N, create the (map) homomorphism from M to N given by matrix X.

`Hom(M, N): ModRng, ModRng -> ModMatRng`

Given a matrix algebra A defined over a field and A-modules M and N, construct the vector space of homomorphisms, $\mathrm{Hom}_K(M, N)$.

`GHom(M, N): ModGrp, ModGrp -> ModMatGrp`

Given $K[G]$ modules M and N, where K is a finite field or number field, construct the space of all KG-homomorphisms from M to N. Note that `GHom(M,N) = Fix(HomMod(M,N))`.

`GHomOverCentralizingField(M, N): ModGrp, ModGrp -> ModMatGrp`

`GHomOverCentralizingField(M, N): ModRng, ModRng -> ModRng`

Let A be a matrix algebra defined over a finite field or number field. Given A-modules M and N, construct $\mathrm{Hom}_{L[G]}(M, N)$ as a subspace of $\mathrm{Hom}_K(M, N)$, where L is the centralizing field of M.

`AHom(M, N): ModRng, ModRng -> ModMatRng`

Let A be a matrix algebra defined over a finite field or number field. Given A-modules M and N, construct the vector space of homomorphisms, $\mathrm{Hom}_A(M, N)$, as a submodule of $\mathrm{Hom}_K(M, N)$.

`HomMod(M, N): ModGrp, ModGrp -> ModGrp`

Let M and N be two $K[G]$-modules where K is a finite field or a number field. This function constructs the $K[G]$-module of all K-homomorphisms from M to N. The action of G on the module is given by $(m)(\alpha * g) := ((m * g^{-1})\alpha) * g$ for $m \in M$, $g \in G$, and $\alpha \in$`Hom(M,N)`.

`H ! f: ModMatRng, Map -> ModMatRngElt`

Given the matrix space H, which is $\mathrm{Hom}_A(M, N)$ for A-modules M and N, together with a homomorphism f from M to N, create the matrix corresponding to the map f.

`IsModuleHomomorphism(X): ModMatFldElt -> BoolElt`

Given a matrix X belonging to $\mathrm{Hom}_K(M, N)$, where M and N are A-modules, return `true` if X is an A-homomorphism.

Example: Hom Space of Modules for a Soluble Group (ex-1d057a)

We construct a 12-dimensional module M and a 9-dimensional submodule P of M for a soluble group of order 648 over $\mathbf{F}_3$. We then construct $H = \mathrm{Hom}_A(M, N)$ and perform map operations on elements of H.

```
> G := PermutationGroup< 12 |
>         (1,6,7)(2,5,8,3,4,9)(11,12),
>         (1,3)(4,9,12)(5,8,10,6,7,11) >;
> K := GF(3);
> P := PermutationModule(G, K);
> M := sub< P | [1,0,0,0,0,1,0,0,1,0,0,1] >;
> M;
GModule M of dimension 9 over GF(3)
> H := AHom(P, M);
> H: Maximal;
KMatrixSpace of 12 by 9 GHom matrices and dimension 2 over GF(3)
Echelonized basis:
[1 1 1 0 0 0 0 0 0]
[1 1 1 0 0 0 0 0 0]
[1 1 1 0 0 0 0 0 0]
[0 0 0 1 1 0 0 0 0]
[0 0 0 1 1 0 0 0 0]
[0 0 0 1 1 0 0 0 0]
[0 0 0 0 0 1 1 0 0]
[0 0 0 0 0 1 1 0 0]
[0 0 0 0 0 1 1 0 0]
[0 0 0 0 0 0 0 1 1]
[0 0 0 0 0 0 0 1 1]
[0 0 0 0 0 0 0 1 1]

[0 0 0 1 1 1 1 1 1]
[0 0 0 1 1 1 1 1 1]
[0 0 0 1 1 1 1 1 1]
[1 1 1 0 0 1 1 1 1]
[1 1 1 0 0 1 1 1 1]
[1 1 1 0 0 1 1 1 1]
[1 1 1 1 1 0 0 1 1]
[1 1 1 1 1 0 0 1 1]
[1 1 1 1 1 0 0 1 1]
[1 1 1 1 1 1 1 0 0]
[1 1 1 1 1 1 1 0 0]
[1 1 1 1 1 1 1 0 0]
> // We write down a random homomorphism from M to P.
> f := 2*H.1 + H.2;
> f;
[2 2 2 1 1 1 1 1 1]
[2 2 2 1 1 1 1 1 1]
[2 2 2 1 1 1 1 1 1]
[1 1 1 2 2 1 1 1 1]
```

(continues on next page)

(continued from previous page)

```
[1 1 1 2 2 1 1 1 1]
[1 1 1 2 2 1 1 1 1]
[1 1 1 1 1 2 2 1 1]
[1 1 1 1 1 2 2 1 1]
[1 1 1 1 1 2 2 1 1]
[1 1 1 1 1 1 1 2 2]
[1 1 1 1 1 1 1 2 2]
[1 1 1 1 1 1 1 2 2]
> Ker := Kernel(f);
> Ker;
GModule Ker of dimension 8 with base ring GF(3)
```

If we print the morphism associated with Ker, we see generators for Ker as a submodule of P.

```
> Morphism(Ker, P);
[1 0 2 0 0 0 0 0 0 0 0 0]
[0 1 2 0 0 0 0 0 0 0 0 0]
[0 0 0 1 0 2 0 0 0 0 0 0]
[0 0 0 0 1 2 0 0 0 0 0 0]
[0 0 0 0 0 0 1 0 2 0 0 0]
[0 0 0 0 0 0 0 1 2 0 0 0]
[0 0 0 0 0 0 0 0 0 1 0 2]
[0 0 0 0 0 0 0 0 0 0 1 2]
> // Examine the image of f and its morphism to P.
> Im := Image(f);
> Im;
GModule Im of dimension 4 with base ring GF(3)
> Morphism(Im, P);
[1 1 1 0 0 0 0 0 0 0 0 0]
[0 0 0 1 1 1 0 0 0 0 0 0]
[0 0 0 0 0 0 1 1 1 0 0 0]
[0 0 0 0 0 0 0 0 0 1 1 1]
```

Example: Endomorphisms of a G-Homomorphism Module (ex-b3e6b8)

We construct a G-homomorphism module H_1 for a G-module and then the homomorphism module $H = \mathrm{Hom}(H_1, H_1)$ with right matrix action which is equivalent to (the right representation of) the endomorphism module of H.

```
> P := GModule(CyclicGroup(11), GF(3));
> F := Constituents(P);
> F;
[
    GModule of dimension 1 over GF(3),
    GModule of dimension 5 over GF(3),
    GModule of dimension 5 over GF(3)
]
> H1 := GHom(P, F[2]);
> H1;
KMatrixSpace of 2 by 3 matrices and dimension 1 over Rational Field
> H := Hom(H1, H1, "right");
> H: Maximal;
KMatrixSpace of 5 by 5 matrices and dimension 5 over GF(3)
Echelonized basis:

[1 0 0 0 0]
[0 1 0 0 0]
```

(continues on next page)

(continued from previous page)

```
[0 0 1 0 0]
[0 0 0 1 0]
[0 0 0 0 1]

[0 1 0 0 0]
[1 1 1 2 1]
[2 0 2 1 1]
[2 1 0 0 0]
[0 2 1 0 0]

[0 0 1 0 0]
[2 0 2 1 1]
[2 2 2 2 2]
[2 0 1 0 2]
[1 0 1 2 1]

[0 0 0 1 0]
[2 1 0 0 0]
[2 0 1 0 2]
[2 2 1 2 2]
[2 1 1 0 1]

[0 0 0 0 1]
[0 2 1 0 0]
[1 0 1 2 1]
[2 1 1 0 1]
[2 1 0 0 2]
```

99.14.2 Isomorphism and Similarity

Isomorphism

`IsIsomorphic(M, N): ModRng, ModRng -> ModRng, ModRng, BoolElt, AlgMatElt`

Let A be a matrix algebra over a finite field or a number field. This intrinsic returns `true` if the A-modules M and N are isomorphic, `false` otherwise. If M and N are isomorphic, the function also returns a matrix T such that $M^T = N$. Note that the action generators of M and N must match, so the function effectively determines whether there is an invertible matrix T such that `T^-1*ActionGenerator(M, i)*T` equals `ActionGenerator(N, i)` for each i. This test uses a fast algorithm due to Brooksbank and Luks.

`SummandIsomorphism(M, N): ModRng, ModRng -> ModRng, ModRng, Map, Map`

Let A be a matrix algebra over a finite field or a number field and let M and N be A-modules. This intrinsic determines the unique maximal, isomorphic summands of module M and N. These are returned together with the inverse isomorphisms between them. The algorithm used is due to Brooksbank and Luks.

Similarity of Cyclic Algebras and their Modules

Magma supports similarity testing of modules over cyclic associative rings and cyclic groups. Module similarity over general rings and groups is graph isomorphism hard. The algorithms used are based on [BW15].

`IsCyclic(R): AlgAss -> BoolElt, AlgAssElt`

Let A be a matrix algebra over a finite field and let M be an A-module. This intrinsic decides if the algebra is generated by a single element and, if so, return such a generator.

`IsSimilar(M, N): ModRng, ModRng -> BoolElt, Map`

Let A be a matrix algebra over a finite field or a number field and let M and N be A-modules.

This intrinsic decides if modules M and N are similar. Note that one of the two modules must have a cyclic coefficient ring.

Example: Similarity versus Isomorphism of Modules (ex-693618)

In Magma modules of a group or algebra are defined by the action of a fixed generating set of the algebra. Therefore isomorphism of modules in Magma assumes the given modules have been specified by the same generating set. This can lead to a stricter interpretation of isomorphism than perhaps intended in some situations. Consider the following example comparing two 1-dimensional vector spaces over the field $\mathbf{F}_9$.

```
> SetSeed(1);
> R := MatrixAlgebra(GF(3),2);
> A := sub<R| [R!1, R![0,1,2,0]]>;
> B := sub<R| [R!1, R![1,1,2,1]]>;
> A eq B;   // Both are a field GF(9).
true
> M := RModule(A); // A 1-dim. GF(9) vector space.
> N := RModule(B); // A 1-dim. GF(9) vector space.
> IsIsomorphic(M,N);
false
> MinimalPolynomial(A.2);
$.1^2 + 1
> MinimalPolynomial(B.2);
$.1^2 + $.1 + 2
```

Isomorphism of the two modules M and N failed because the two vector spaces are specified by different generators of $\mathbf{F}_9$, as confirmed by the minimum polynomials of the generators. Module similarity allows the comparison of modules specified by different generating sets, so in this example theses to vector spaces can be proven equivalent.

```
> IsSimilar(M,N);
true
[2 0]
[0 2]
```

Similarity can be used to compare modules given by algebras that are conjugate, but perhaps not equal.

```
> p := RandomIrreduciblePolynomial(GF(101), 10);
> q := RandomIrreduciblePolynomial(GF(101), 10);
> X := CompanionMatrix(p);
> Y := CompanionMatrix(q);
> A := sub<Parent(X)|[X]>;      // Finite field of size 101^10
> B := sub<Parent(Y)|[Y]>;      // Finite field of size 101^10
> M := RModule(A);              // 1-dim vector space over A.
> N := RModule(B);              // 1-dim vector space over B.
> IsIsomorphic(M,N);            // Not isomorphic as A and B are distinct.
false
> cyc, f := IsSimilar(M,N);     // But similar as A is isomorphic to B.
> // f conjugates A into B?
> forall { a : a in Generators (A) | f * a * f^-1 in B };
true
> // and f is a semilinear transform M->N ?
> forall{ i : i in [1..Ngens (M)] | forall { j : j in [1..Ngens (A)] | (Vector
> (M.i * A.j) * f) eq (Vector(M.i)*f)*(f^(-1)*A.j*f) } };
true
```

Similarity is presently available for cyclic algebras. This can be tested and a generator returned.

```
> M := RandomMatrix(GF(9), 100, 100);
> A := sub< Parent(M) | [ M^(Random(50)) : i in [1..10]] >;
> Ngens(A);
> IsCyclic(A);
true
...
```

99.14.3 The Endomorphism Ring

EndomorphismAlgebra(M): ModRng -> AlgMat

EndomorphismRing(M): ModRng -> AlgMat

```
Direct: BoolElt                    Default: false
```

Given an A-module M over K, where K is a finite field or a number field, return the endomorphism ring $\mathrm{End}_A(M)$ of M as a subring of the complete matrix ring $K^{(n\times n)}$.

CentreOfEndomorphismRing(M): ModRng -> AlgMat

Given an A-module M with base ring K, where K is a finite field or a number field, return the centre of the endomorphism ring $\mathrm{End}_A(M)$ of M as a subring of the complete matrix ring $K^{(n\times n)}$. This is equivalent to `Centre(EndomorphismRing(M))` but will often be much faster.

AutomorphismGroup(M): ModRng -> GrpMat

Given an A-module M with base ring K, where K is a finite field or a number field, return the automorphism group $\mathrm{Aut}(M)$ of M as a subgroup G of the general linear group $GL(n, K)$. In fact, G is the group of units of $\mathrm{End}(M)$.

Example: Endomorphism Ring of a Permutation Module (ex-d24d44)

We construct the endomorphism ring for a permutation module over $\mathbf{F}_3$ for a soluble group of order 648.

```
> G := PermutationGroup< 12 |
>         (1,6,7)(2,5,8,3,4,9)(11,12),
>         (1,3)(4,9,12)(5,8,10,6,7,11) >;
> P := PermutationModule(G, GF(3));
> time End := EndomorphismAlgebra(P);
Time: 0.000
> End;
Matrix Algebra of degree 12 and dimension 3 over GF(3)
```

Thus, the permutation module P has 27 endomorphisms.

```
> time Aut := AutomorphismGroup(P);
Time: 0.010
> Aut;
MatrixGroup(12, GF(3))
Generators:
    [1 0 0 1 1 1 1 1 1 1 1 1]
    [0 1 0 1 1 1 1 1 1 1 1 1]
    [0 0 1 1 1 1 1 1 1 1 1 1]
    [1 1 1 1 0 0 1 1 1 1 1 1]
    [1 1 1 0 1 0 1 1 1 1 1 1]
    [1 1 1 0 0 1 1 1 1 1 1 1]
    [1 1 1 1 1 1 1 0 0 1 1 1]
    [1 1 1 1 1 1 0 1 0 1 1 1]
    [1 1 1 1 1 1 0 0 1 1 1 1]
    [1 1 1 1 1 1 1 1 1 1 0 0]
```

(continues on next page)

(continued from previous page)

```
    [1 1 1 1 1 1 1 1 1 0 1 0]
    [1 1 1 1 1 1 1 1 1 0 0 1]

    [2 1 1 0 0 0 0 0 0 0 0 0]
    [1 2 1 0 0 0 0 0 0 0 0 0]
    [1 1 2 0 0 0 0 0 0 0 0 0]
    [0 0 0 2 1 1 0 0 0 0 0 0]
    [0 0 0 1 2 1 0 0 0 0 0 0]
    [0 0 0 1 1 2 0 0 0 0 0 0]
    [0 0 0 0 0 0 2 1 1 0 0 0]
    [0 0 0 0 0 0 1 2 1 0 0 0]
    [0 0 0 0 0 0 1 1 2 0 0 0]
    [0 0 0 0 0 0 0 0 0 2 1 1]
    [0 0 0 0 0 0 0 0 0 1 2 1]
    [0 0 0 0 0 0 0 0 0 1 1 2]

    [0 1 1 0 0 0 0 0 0 0 0 0]
    [1 0 1 0 0 0 0 0 0 0 0 0]
    [1 1 0 0 0 0 0 0 0 0 0 0]
    [0 0 0 0 1 1 0 0 0 0 0 0]
    [0 0 0 1 0 1 0 0 0 0 0 0]
    [0 0 0 1 1 0 0 0 0 0 0 0]
    [0 0 0 0 0 0 0 1 1 0 0 0]
    [0 0 0 0 0 0 1 0 1 0 0 0]
    [0 0 0 0 0 0 1 1 0 0 0 0]
    [0 0 0 0 0 0 0 0 0 0 1 1]
    [0 0 0 0 0 0 0 0 0 1 0 1]
    [0 0 0 0 0 0 0 0 0 1 1 0]
> #Aut;
18
> IsAbelian(Aut);
true
> AbelianInvariants(Aut);
[ 3, 6 ]
```

Thus, the permutation module P has 18 automorphisms. The automorphism group is isomorphic to the abelian group $\mathbb{Z}_2 \times \mathbb{Z}_3 \times \mathbb{Z}_3$.

99.15 Projective Indecomposable Modules

For a finite group G and a finite field K, the projective indecomposable $K[G]$-modules are in one-one correspondence with the irreducible $K[G]$-modules, where the projective indecomposable module P corresponding to the irreducible module I has the property that Socle(P) and P/JacobsonRadical(P) are both isomorphic to I.

Magma functions for the construction of the irreducible modules are described in Subsection *Enumerating All Irreducible Modules*. The functions described in this section may be used to construct the corresponding projective indecomposables for finite groups of moderate order – up to around 10^9, depending on the example. Large-dimensional projective indecomposable modules can sometimes be constructed by using *condensation* techniques, which allow many of the necessary computations to be carried out in condensed modules, which may have significantly smaller dimension than their standard uncondensed equivalents. As with the computation of irreducible modules, these methods work best for permutation groups and PC-groups.

The verbose flag `"KGModule"` may be set to 1 or 2 to show details of the computations.

`ProjectiveIndecomposableDimensions(G, K): Grp, FldFin -> SeqEnum`

Let G be a finite group and K a finite field. This intrinsic returns the K-dimensions of the projective indecomposable $K[G]$-modules corresponding to the irreducible $K[G]$-modules as constructed by the Intrinsic `IrreducibleModules(G, K)`. (These dimensions can be computed quickly from the Brauer characters of the irreducible modules, using the Cartan matrix).

`ProjectiveIndecomposableModule(I: parameters): ModGrp -> ModGrp`

```
condensation: BoolElt                    Default: false
```

Let G be a finite group and let K be a finite field. This intrinsic constructs and returns the projective indecomposable $K[G]$-module P corresponding to the irreducible $K[G]$-module I. Note that Socle(P) and P/JacobsonRadical(P) are both isomorphic to I.

If set to `true`, then an attempt is made to find a subgroup of G which allows computations to be carried out using condensed versions of the modules involved.

`ProjectiveIndecomposableModules(G, K: parameters): Grp, FldFin -> SeqEnum`

```
condensation: BoolElt                    Default: false
```

Let G be a finite group and let K be a finite field. This intrinsic constructs and returns the complete list of projective indecomposable $K[G]$-modules corresponding to the irreducible $K[G]$-modules as returned by the `IrreducibleModules(G,K)` intrinsic.

If set to `true`, then an attempt is made to find a subgroup of G which allows computations to be carried out in condensed versions of the modules involved.

`PIMBlocks(CM): AlgMatElt -> SeqEnum`

`PIMBlocks(I, P): SeqEnum, SeqEnum -> SeqEnum`

Let G be a finite group and let K be a finite field. This intrinsic partitions the projective indecomposable KG-modules (PIMs) into blocks. There are two versions of `PIMBlocks`, distinguished by argument types. The first version works off the Cartan matrix which if available is very fast. The second version has as arguments sequences I and P containing the irreducible modules and PIMs for G, respectively. This version computes the constituents of each projective indecomposable. Consequentally, as the dimension of the PIMs increases, this can be very time consuming. In each case the blocks are returned as a sequence of sequences of the numbers of those PIMs that lie in given block.

`ProjectiveCover(M): ModGrp -> ModGrp, ModMatGrpElt`

Given a $K[G]$-module M, where K is a finite field, compute the projective cover P of M together with a $K[G]$-module epimorphism $P \to M$ returned as a matrix.

If P_i is the projective indecomposable $K[G]$-module corresponding to the irreducible $K[G]$-module I_i and M/JacobsonRadical(M) is isomorphic to $\oplus_{j=1}^{t} I_{i_j}$ then P is isomorphic to $\oplus_{j=1}^{t} P_{i_j}$.

`InjectiveHull(M): ModGrp -> ModGrp, ModMatGrpElt`

Given a $K[G]$-module M, where K is a finite field, compute the injective hull I of M together with a $K[G]$-module monomorphism $M \to I$ returned as a matrix.

If P_i is the projective indecomposable $K[G]$-module corresponding to the irreducible $K[G]$-module I_i and $Socle(M)$ is isomorphic to $\oplus_{j=1}^{t} I_{i_j}$ then I is isomorphic to $\oplus_{j=1}^{t} P_{i_j}$.

Example: Projective Indecomposables for Alt(8) (ex-a07bd5)

We compute the projective indecomposable modules for the alternating group of degree 8 over the field of order 2.

```
> G := Alt(8);
> K := GF(2);
> IrreducibleModules(G, K);
[
```

(continues on next page)

(continued from previous page)

```
    GModule of dimension 1 over GF(2),
    GModule of dimension 4 over GF(2),
    GModule of dimension 4 over GF(2),
    GModule of dimension 6 over GF(2),
    GModule of dimension 14 over GF(2),
    GModule of dimension 20 over GF(2),
    GModule of dimension 20 over GF(2),
    GModule of dimension 64 over GF(2)
]
> ProjectiveIndecomposableDimensions(G, K);
[ 448, 192, 192, 320, 320, 192, 192, 64 ]
> time proj := ProjectiveIndecomposables(G, K);
Time: 22.070
> proj;
[
    GModule of dimension 448 over GF(2),
    GModule of dimension 192 over GF(2),
    GModule of dimension 192 over GF(2),
    GModule of dimension 320 over GF(2),
    GModule of dimension 320 over GF(2),
    GModule of dimension 192 over GF(2),
    GModule of dimension 192 over GF(2),
    GModule of dimension 64 over GF(2)
]
> CompositionFactors(proj[1]);
[
    GModule of dimension 1 over GF(2),
    GModule of dimension 20 over GF(2),
    GModule of dimension 20 over GF(2),
    GModule of dimension 1 over GF(2),
    GModule of dimension 14 over GF(2),
    GModule of dimension 6 over GF(2),
    GModule of dimension 1 over GF(2),
    GModule of dimension 20 over GF(2),
    GModule of dimension 20 over GF(2),
    GModule of dimension 1 over GF(2),
    GModule of dimension 14 over GF(2),
    GModule of dimension 4 over GF(2),
    GModule of dimension 4 over GF(2),
    GModule of dimension 14 over GF(2),
    GModule of dimension 6 over GF(2),
    GModule of dimension 1 over GF(2),
    GModule of dimension 4 over GF(2),
    GModule of dimension 4 over GF(2),
    GModule of dimension 20 over GF(2),
    GModule of dimension 20 over GF(2),
    GModule of dimension 1 over GF(2),
    GModule of dimension 14 over GF(2),
    GModule of dimension 6 over GF(2),
    GModule of dimension 1 over GF(2),
    GModule of dimension 6 over GF(2),
    GModule of dimension 14 over GF(2),
    GModule of dimension 6 over GF(2),
    GModule of dimension 1 over GF(2),
    GModule of dimension 20 over GF(2),
    GModule of dimension 4 over GF(2),
```

(continues on next page)

(continued from previous page)

```
    GModule of dimension 20 over GF(2),
    GModule of dimension 1 over GF(2),
    GModule of dimension 1 over GF(2),
    GModule of dimension 1 over GF(2),
    GModule of dimension 14 over GF(2),
    GModule of dimension 1 over GF(2),
    GModule of dimension 20 over GF(2),
    GModule of dimension 1 over GF(2),
    GModule of dimension 6 over GF(2),
    GModule of dimension 4 over GF(2),
    GModule of dimension 20 over GF(2),
    GModule of dimension 14 over GF(2),
    GModule of dimension 4 over GF(2),
    GModule of dimension 4 over GF(2),
    GModule of dimension 1 over GF(2),
    GModule of dimension 20 over GF(2),
    GModule of dimension 6 over GF(2),
    GModule of dimension 20 over GF(2),
    GModule of dimension 1 over GF(2),
    GModule of dimension 6 over GF(2),
    GModule of dimension 14 over GF(2),
    GModule of dimension 1 over GF(2)
]
```

99.16 Cohomology and Extensions

99.16.1 Cohomology

The facilities developed for group cohomology are described at length in Chapter Cohomology and Extensions. In this section descriptions of some basic intrinsics are repeated for convenience.

`CohomologyModule(G, M): GrpPerm, ModGrp -> ModCoho`

`CohomologyModule(G, M): GrpPC, ModGrp -> ModCoho`

Given a finite group G and a $K[G]$-module M, where K is a finite field, this intrinsic returns a *cohomology module* for the action of G. The group G may be a finite permutation group, a finite matrix group, a conditioned PC-group, or any finitely presented group. Much of the Magma cohomology machinery requires a cohomology module to be defined in order to compute cohomology groups etc.

`CohomologyGroup(CM, n): ModCoho, RngIntElt -> ModTupRng`

Given a cohomology module CM for the group G acting on the module M and a non-negative integer n taking one of the values 0, 1, or 2, this function returns the cohomology group $H^n(G, M)$.

`CohomologicalDimension(CM, n): ModCoho, RngIntElt -> RngIntElt`

Given a cohomology module CM for the group G acting on the module M defined over a finite field K and a non-negative integer n taking one of the values 0, 1, or 2, this function returns the dimension of $H^n(G, M)$ over K. Note that this function may only be applied to the module returned by a call to `CohomologyModule(G, M)`, where M is a module over a finite field K. When $n = 2$, this function is faster and may be applied to much larger examples than `CohomologyGroup(CM, n)` but, unlike that function, it does not enable the user to compute with explicit extensions and two-cocycles.

`CohomologicalDimension(M, n): ModGrp, RngIntElt -> RngIntElt`

For $K[G]$-module M (with K a finite field and G a finite group), compute and return the K-dimension of the cohomology group $H^n(G, M)$ for $n \geq 0$. For $n = 0$ and 1, this is carried out by using the functions described in Chapter Cohomology and Extensions. For $n \geq 2$, it is done recursively using projective covers and dimension shifting to reduce to the case $n = 1$. (In

particular, for $n = 2$, the method is different from that employed by the corresponding function for a cohomology module described in Chapter Cohomology and Extensions.)

`CohomologicalDimensions(M, n): ModGrp, RngIntElt -> RngIntElt`

For $K[G]$-module M (with K a finite field and G a finite group), compute and return the sequence of K-dimensions of the cohomology groups $H^k(G, M)$ for $1 \leq k \leq n$. On account of the recursive method used, this is quicker than computing them individually.

Example: Cohomology of a PSL(3,3) Module (ex-047dc8)

The simple group PSL$(3, 3)$ has an absolutely irreducible module of degree 7 over GF(3). We will compute its first and second cohomology groups.

```
> G := PSL(3, 3);
> Irrs := AbsolutelyIrreducibleModules(G, GF(3));
> M := Irrs[6];
> M;
GModule of dimension 7 over GF(3)
> CM := CohomologyModule(G, M);

> CohomologyGroup(CM, 1);
Full Vector space of degree 1 over GF(3)

> CohomologyGroup(CM, 2);
Full Vector space of degree 2 over GF(3)
```

All other absolutely irreducible modules for PSL$(3, 3)$ over GF(3) have trivial first and second cohomology groups.

Example: Cohomological Dimension over Alt(8) (ex-68a11a)

We compute the cohomology of the irreducible modules for Alt(8) over the field of order 2 computed in the previous example.

```
> G := Alt(8);
> K := GF(2);
> irr :=  IrreducibleModules(G, K);
> time [ CohomologicalDimension(I, 1) : I in irr ];
[ 0, 0, 0, 1, 1, 1, 1, 0 ]
Time: 0.000
> time [ CohomologicalDimension(I, 2) : I in irr ];
[ 1, 1, 1, 0, 2, 0, 0, 0 ]
Time: 0.100
> time [ CohomologicalDimension(I, 3) : I in irr ];
[ 2, 1, 1, 1, 1, 1, 1, 0 ]
Time: 2.400
> time [ CohomologicalDimension(I, 4) : I in irr ];
[ 2, 1, 1, 2, 3, 2, 2, 0 ]
Time: 11.730
> time CohomologicalDimensions(irr[1], 6);
[ 0, 1, 2, 2, 3, 6 ]
Time: 14.240
```

99.16.2 Extensions of Modules

For $K[G]$-modules M and N where K is a finite field, the K-vector $\mathrm{Ext}(M, N)$ of equivalence classes of $K[G]$-module extensions

$$0 \to N \to L \to M \to 0$$

of N by M can be computed, and the corresponding extensions L constructed.

Ext(M, N): ModGrp, ModGrp -> ModTupFld

Given $K[G]$-modules M and N where K is a finite field, construct the K-vector space $\mathrm{Ext}(M, N)$ of equivalence classes of $K[G]$-module extensions of N by M.

Extension(M, N, e, r): ModGrp, ModGrp, ModTupFldElt, Map -> ModGrp, ModMatGrpElt, ModMatGrpElt

Given $K[G]$-modules M and N, where K is a finite field, construct a $K[G]$-module extension L of N by M corresponding to the element e of E, where E and ϕ must be the vector space and map returned by a previous call of Ext(M,N). The insertion $N \to L$ and projection $L \to M$ are also returned.

MaximalExtension(M, N, E, r): ModGrp, ModGrp, ModTupFld, Map -> ModGrp

Let M and N be $K[G]$-modules, where K is a finite field, and further let E be the vector space and r the map returned by a previous call of Ext(M, N). Construct the largest possible $K[G]$-module extension L of a direct sum of copies of N by M, such that none of the submodules of L that are isomorphic to N has a complement in L.

MaximalExtension(M, N): ModGrp, ModGrp -> ModGrp

This is the same as MaximalExtension(M, N, E, r), but Ext(M,N) is called from within the function.

MaximalExtension(~M, N): ModGrp, ModGrp

A version of MaximalExtension(M, N) that replaces M by the result.

Example: Extensions of Alt(5) Modules (ex-fa426a)

```
> G := Alt(5);
> I := IrreducibleModules(G, GF(2));
> I;
[
    GModule of dimension 1 over GF(2),
    GModule of dimension 4 over GF(2),
    GModule of dimension 4 over GF(2)
]
> M1 := rep{M: M in I | Dimension(M) eq 1};
> M4 := rep{M: M in I | Dimension(M) eq 4 and not IsAbsolutelyIrreducible(M)};
> M4; assert not IsAbsolutelyIrreducible(M4);
GModule M of dimension 4 over GF(2)
> E, phi := Ext(M4, M1);
> E;
Full Vector space of degree 2 over GF(2)
> Extension(M4, M1, E.1, phi);
GModule of dimension 5 over GF(2)

[0 0 0 0 1]

[1 0 0 0]
[0 1 0 0]
[0 0 1 0]
[0 0 0 1]
[0 0 0 0]
> E := MaximalExtension(M4, M1);
> E;
GModule E of dimension 6 over GF(2)
> CompositionFactors(E);
[
    GModule of dimension 1 over GF(2),
    GModule of dimension 1 over GF(2),
```

(continues on next page)

(continued from previous page)

```
    GModule of dimension 4 over GF(2)
]
```

LowDimensionalModules(G, K, n): Grp, Fld, RngIntElt -> SeqEnum

Let G be a permutation group, a matrix group, or a group defined by a power-commutator presentation and let K be a finite field. This intrinsic constructs a representative of each isomorphism class of $K[G]$-modules of dimension at most n. It computes them by systematically constructing extensions of modules, starting with the irreducibles

99.17 Vertex and Source of an Indecomposable Module

For a finite group G and a finite field K, a vertex of an indecomposable $K[G]$-modules M is a subgroup V such that there exists an indecomposable $K[V]$-module S such that M is a direct summand of the induced module S^G, and V is minimal subject to this condition. The module S in the above is called a source for M. A vertex is defined up to conjugacy in G, and, for fixed vertex V, a source is defined up to conjugacy in the normalizer of V. The projective indecomposable modules of G are exactly those with vertex the trivial subgroup. If K has characteristic p, then a vertex is a p-subgroup of G.

These concepts were introduced by J.A. Green in [Gre59]. The algorithms used by Magma follow those given in [DKZ08].

Vertex(M : parameters): ModGrp -> Grp

```
H: Grp                    Default:
```

Let G be a finite group, K be a finite field, and M an indecomposable $K[G]$-module. This intrinsic returns a representative V of the conjugacy class of vertices for M, where V a subgroup of G.

The parameter H may be used to give a subgroup of G that contains the vertex of M, such as a defect group for the block of M. By default the value of H is set to a Sylow p-subgroup of G.

Source(M : parameters): ModGrp -> ModGrp, ModGrp

```
H: Grp                    Default:
```

Let G be a finite group, K be a finite field, and M an indecomposable $K[G]$-module. This intrinsic returns a source module S for M. The action group for S is a vertex for M. The Green correspondent of M is the second return value. The parameter H is same as for the **Vertex** intrinsic above, and is passed to it.

Example: Vertices and Sources in M22.2 (ex-5a661a)

We compute the vertices and sources of the absolutely irreducible 2-modular G-modules for the group $M_{22}.2$. We can compare the results with those of [DK09].

```
> G := PermutationGroup("M22d2",1);
> irrs := AbsolutelyIrreducibleModules(G, GF(2));
> irrs;
[
  GModule of dimension 1 over GF(2),
  GModule of dimension 10 over GF(2),
  GModule of dimension 10 over GF(2),
  GModule of dimension 34 over GF(2),
  GModule of dimension 98 over GF(2),
  GModule of dimension 140 over GF(2)
]
> time verts := [Vertex(m):  m in irrs];
Time: 2.330
```

(continues on next page)

(continued from previous page)

```
> [#v : v in verts];
[ 256, 256, 256, 256, 256, 128 ]
```

The orders of the vertices tally with those given in [DK09]. A Sylow 2-subgroup of G has order 256, and a vertex for the 140 dimensional module is a Sylow subgroup of the socle M_{22}. This time we work out the sources.

```
> time srcs := [Source(m) : m in irrs];
Time: 2.690
> srcs;
[
  GModule of dimension 1 over GF(2),
  GModule of dimension 10 over GF(2),
  GModule of dimension 10 over GF(2),
  GModule of dimension 34 over GF(2),
  GModule of dimension 98 over GF(2),
  GModule of dimension 140 over GF(2)
]
```

The article cited gives these results except for the last, where the source is stated to have dimension 70. The article assumes an algebraically closed field however, which is not the case with $GF(2)$.

```
> m := irrs[#irrs];
> m4 := ChangeRing(m, GF(4));
> time Source(m4);
GModule of dimension 70 over GF(2^2)
Time: 6.590
```

This resolves our differences: though the irreducible module over $GF(2)$ is absolutely irreducible, the restriction to a vertex is indecomposable, but not absolutely indecomposable, and a field extension of degree 2 yields an absolutely indecomposable source module.

99.18 Bimodules

99.18.1 Introduction

For a field K and groups G and H, a $(KG,\ KH)$-bimodule is a vector space over K equipped with a left G-action and a right H-action.

In the Magma implementation, (KG, KH)-bimodules are regarded as being equivalent to standard right $K(G \times H)$-modules, where

- the right action of H in the bimodule is the same as the right action of H in the equivalent $K(G \times H)$-module; and
- the left action of G in the bimodule is related to the right action of G in the equivalent $K(G \times H)$-module by $g^{-1} * v = v * g$ for all $g \in G$ and v in the common underlying K-vector space.

Most of the functionality for standard right KG-modules, such as forming sub- and quotient modules, defining homomorphisms, etc, should also work for (KG, KH)-modules, where results are calculated using the equivalent $K(G \times H)$-module. But note that there is no requirement for the groups G and H to have the same type.

99.18.2 Construction

`Bimodule(M, N): ModGrp, ModGrp -> LRModGrp`

Inputs M and N should be KG- and KH-modules of the same dimension, such that the actions of G and H commute; i.e. $(v * g) * h = (v * h) * g$ for all $g \in G$, $h \in H$, and v in the common underlying K-vector space V of M and N. An error will result if this condition is not satisfied.

The corresponding bimodule is returned in which $g^{-1} * v = v * g$ for all $g \in G$ and $v \in V$.

Example: Bimodule from Sym(3) and a Cyclic Group (ex-31c812)

```
> G:=Sym(3);
> H:=CyclicGroup(2);
> M:=PermutationModule(G,GF(3));
> S:=ScalarMatrix(3, GF(3)!2);
> N:=GModule(H,[S]);
> B:=Bimodule(M,N);
> B;
Bimodule B of dimension 3 over GF(3)
> M.1 * G.1;
M: (0 1 0)
> G.1^-1 * B.1;
B: (0 1 0)
> GHom(B,B);
KMatrixSpace of 3 by 3 matrices and dimension 2 over GF(3)
> #Submodules(B);
4
```

It is often more convenient to construct a bimodule directly from the equivalent $K(G \times H)$-module.

`Bimodule(G, H, M): Grp, Grp, ModGrp -> LRModGrp`

Inputs G and H should be groups, and M should be a $(G \times H)$-module. The equivalent (KG, KH)-bimodule is constructed.

Example: Bimodule from a Direct Product (ex-5e5954)

```
> X := DirectProduct(Sym(4), Alt(5));
> I := IrreducibleModules(X, GF(3));
> M12 := I[9];
> M12;
GModule M12 of dimension 12 over GF(3)
> B := Bimodule(Sym(4), Alt(5), M12);
> B;
Bimodule B of dimension 12 over GF(3)
> IsIrreducible(B);
true
```

`LeftBimodule(M): ModGrp -> LRModGrp`

M should be a KG-module. The corresponding (KG, KT)-bimodule is returned in which T is the trivial group and $g^{-1} * v = v * g$ for all $g \in G$ and $v \in V$.

`RightBimodule(M): ModGrp -> LRModGrp`

M should be a KH-module. The corresponding (KT, KH)-bimodule is returned in which T is the trivial group.

99.18.3 Extracting the Left and Right Modules From a Bimodule

`LeftOppositeModule(B): LRModGrp -> ModGrp`

Extract the equivalent right KG-module M from the (KG, KH)-bimodule B, where $g^{-1} * v = v * g$ for $g \in G$, and v in the common K-vector space V of B and M.

`RightModule(B): LRModGrp -> ModGrp`

Extract the equivalent right KH-module M from the (KG, KH)-bimodule B.

Example: Left and Right Modules of a Bimodule (ex-022d8f)

We continue with the previous example, and check the property $g^{-1} * v = v * g$ for all $g \in G$ and $v \in V$.

```
> ML := LeftOppositeModule(B);
> ML;
GModule ML of dimension 12 over GF(3)
> G := Group(ML);
> G;
Symmetric group G acting on a set of cardinality 4
Order = 24 = 2^3 * 3
    (1, 2, 3, 4)
    (1, 2)
> V := VectorSpace(B);
> forall{<i,j> : i in [1..2], j in [1..12] |
>                                V!(G.i^-1*B.j) eq V!(ML.j*G.i)};
true
```

99.18.4 Permutation Bimodules

`PermutationBimodule(G, H, m, K): Grp, Grp, Map, Fld -> LRModGrp`

The last argument K should be a field, G and H should be groups, and m should be a homomorphism from a subgroup A of G to H.

Let Y be the subgroup $\{(a, m(a)) : a \in A\}$ of $G \times H$, and let M be the $K(G \times H)$-permutation module defined by the image of the action of $G \times H$ on the right cosets of Y. The (KG, KH)-bimodule corresponding to M is returned.

Example: Permutation Bimodules (ex-791051)

```
> G := Sym(3);
> H := CyclicGroup(6);
> PG := Sylow(G,3);
> PH := Sylow(H,3);
> m := hom<PG -> PH | [PH.1] >;
> K := GF(3);
> B := PermutationBimodule(G,H,m,K);
> B;
Bimodule B of dimension 12 over GF(3)
> GHom(B,B);
KMatrixSpace of 12 by 12 matrices and dimension 8 over GF(3)
```

99.18.5 Tensor Products of Bimodules

`TensorProduct(B1, B2): LRModGrp, LRModGrp -> LRModGrp`

The arguments B_1 and B_2 are, respectively, (KG, KH)- and (KH, KJ)-bimodules for the groups G, H, J. The (KG, KJ)-bimodule $B_1 \otimes_{KH} B_2$ is returned.

Example: Induced Modules via Tensor Products (ex-f17ecc)

Let H be a subgroup of a group G, and M be a right KH-module. Then $M \otimes_{KH} KG$ is (by definition) equal to the induced module M^G.

This is not a sensible way to construct induced modules in practice, but we can use it as an example of the use of the tensor product of bimodules. We regard M as a (KT, KH)-bimodule, where T is the trivial group, and KH as a (KH, KG)-bimodule.

```
>  G := Sym(5);
>  H := Stabiliser(G,5);
>  K := GF(5);
>  M := IrreducibleModules(H,K)[4];
>  M;
```

(continues on next page)

(continued from previous page)

```
GModule M of dimension 3 over GF(5)
> B1 := RightBimodule(M);
> //make KG as right G-module
> eG := [ g: g in G ];
> mats := [];
> for i in [1..Ngens(G)] do
>   perm := Sym(#G)![Position(eG, g*G.i) : g in eG ];
>   Append(~mats, PermutationMatrix(K,perm));
> end for;
> RM := GModule(G,mats);
> //and as left H-module
> mats := [];
> for i in [1..Ngens(H)] do
>   perm := Sym(#G)![Position(eG, H.i*g) : g in eG ];
>   Append(~mats, PermutationMatrix(K,perm)^-1);
> end for;
> LM := GModule(H,mats);
> B2 := Bimodule(LM,RM);
> T := TensorProduct(B1,B2);
> T;
Bimodule T of dimension 15 over GF(5)
> IsIsomorphic(RightModule(T), Induction(M,G));
true
```

99.19 Enumerating All Irreducible Modules

The construction of all irreducible K[G]-modules for a finite group G is of major interest. If G is soluble there is a very effective method that dates back to Schur. This proceeds by working up a composition series for G and constructing the irreducibles for each subgroup by inducing or extending representations from the previous subgroup. This works equally well over finite fields and over fields of characteristic zero. If G is non-soluble the situation is more difficult. Starting with a faithful representation, by a theorem of Burnside, it is possible to construct all representations by splitting tensor powers of the faithful representation. This approach will be called the *Burnside algorithm*. An algorithm based on this idea developed by John Cannon and Derek Holt uses the Meataxe to split representations and works well over small finite fields. An algorithm for enumerating all rational irreducible representations has been developed by Allan Steel and has been in use for some years. In order to compute the ordinary irreducible modules Allan Steel, in his PhD thesis, developed a method which constructs the irreducible modules recursively by taking irreducibles of subgroups and lifting them to G using induction and extension. Heavy use of character tables is made to guide the procedure. The implemented algorithm was made available in Magma for the first time in March, 2021.

99.19.1 Irreducible Modules over F_q for Arbitrary Groups

The functions described in this section construct all irreducible representations of a finite group. The choice of algorithm depends upon the type of group and the kind of field given. The individual algorithms may be invoked directly by means of intrinsic functions described in subsequent sections.

IrreducibleModules(G, K : parameters): Grp, Fld -> SeqEnum

AbsolutelyIrreducibleModules(G, K : parameters): Grp, Fld -> SeqEnum

```
Alg   : MonStgElt                    Default: %"Schur | Burnside"
DimLim: RngInt                       Default: 2000
```

Let G be a finite group and let K be a field. If G is soluble then K may be either a finite field, the rational field, or a cyclotomic field whose order divides the exponent of G. If G is non-soluble, then currently K is restricted to being a finite field or the rational field. These functions construct all of

the irreducible or absolutely irreducible G-modules over K. Since V2.16, if K is the rational field, then by default a new algorithm is used. Otherwise, if G is soluble, Schur's algorithm is normally used while if G is non-soluble, the Burnside method is used.

The parameter `Alg` may be used to override the usual choice of algorithm if desired. Note that Schur's algorithm may only be applied to soluble groups, while, for the time being, Burnside's method requires that the field K is finite.

The Schur algorithm is usually very fast and is often able to find the complex representations more quickly than it is possible to compute the character table. The speed of the Burnside algorithm is determined firstly by the maximal degree of the irreducible modules and secondly by the number of irreducible modules. It will start to become quite slow if G has modules of dimension in excess of 1000. In order to prevent the Burnside method from wasting huge amounts of time, the algorithm takes a parameter which controls the degree of the largest module that will be consider for splitting.

The parameter `DimLim` only affects the Burnside algorithm where it is used to limit the dimension of the modules which will be considered for splitting. If this limit prevents all irreducibles being found, a warning message is output and those irreducibles that have been found will be returned. This possibility allows the user to determine a sample of low degree modules without using excessive time.

Example: Irreducible Modules of a Group of Order 416 (ex-1afbd4)

We take a group of order $416 = 2^5 13$ and compute its irreducible modules over various fields.

```
> G := SmallGroup(416, 136);
> G;
GrpPC : G of order 416 = 2^5 * 13
PC-Relations:
    G.1^2 = Id(G),
    G.2^2 = G.5,
    G.3^2 = Id(G),
    G.4^2 = G.5,
    G.5^2 = Id(G),
    G.6^13 = Id(G),
    G.3^G.1 = G.3 * G.5,
    G.3^G.2 = G.3 * G.4,
    G.4^G.2 = G.4 * G.5,
    G.4^G.3 = G.4 * G.5,
    G.6^G.1 = G.6^12
```

We first compute the $K[G]$-modules for the finite fields $K = GF(p)$, where p runs through the primes dividing the order of G.

```
> IrreducibleModules(G, GF(2));
[
    GModule of dimension 1 over GF(2),
    GModule of dimension 12 over GF(2)
]
> IrreducibleModules(G, GF(13));
[
    GModule of dimension 1 over GF(13),
    GModule of dimension 1 over GF(13),
    GModule of dimension 1 over GF(13),
    GModule of dimension 1 over GF(13),
    GModule of dimension 1 over GF(13),
    GModule of dimension 1 over GF(13),
    GModule of dimension 1 over GF(13),
    GModule of dimension 1 over GF(13),
    GModule of dimension 2 over GF(13),
```

(continues on next page)

(continued from previous page)

```
    GModule of dimension 2 over GF(13),
    GModule of dimension 4 over GF(13)
]
```

We now compute the $K[G]$-modules where K is the rational field.

```
> time L := IrreducibleModules(G, Rationals());
Time: 16.170
> L;
[
    GModule of dimension 1 over Rational Field,
    GModule of dimension 1 over Rational Field,
    GModule of dimension 1 over Rational Field,
    GModule of dimension 1 over Rational Field,
    GModule of dimension 1 over Rational Field,
    GModule of dimension 1 over Rational Field,
    GModule of dimension 1 over Rational Field,
    GModule of dimension 1 over Rational Field,
    GModule of dimension 2 over Rational Field,
    GModule of dimension 2 over Rational Field,
    GModule of dimension 8 over Rational Field,
    GModule of dimension 12 over Rational Field,
    GModule of dimension 12 over Rational Field,
    GModule of dimension 12 over Rational Field,
    GModule of dimension 12 over Rational Field,
    GModule of dimension 24 over Rational Field,
    GModule of dimension 96 over Rational Field
]
```

Finally, we compute the $K[G]$-modules taking K to be the splitting field over the rationals and verify that the number of irreducibles is equal to the number of conjugacy classes of G.

```
> Exponent(G);
104
> mods := IrreducibleModules(G, CyclotomicField(104));
> #mods;
53;
> #Classes(G);
53
> [ Dimension(N) : N in mods ];
[ 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
  2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 4,
  4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4 ]
> X := CharacterTable(G);
> [ Degree(x) : x in X ];
[ 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
  2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 4,
  4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4 ]
```

The Burnside algorithm finds all irreducible $K[G]$-modules with a faithful $K[G]$-module P and looks for the distinct irreducibles among the tensor powers of P. Both irreducible and absolutely irreducible modules may be found. At present the algorithm is restricted to finite fields. In more detail the algorithm starts with a some faithful permutation module over the given field, splits it into irreducibles using the meataxe, and constructing further modules to split as tensor products of those already found. A warning is printed if all irreducible modules are not found.

AbsolutelyIrreducibleModulesBurnside(G, K : parameters): Grp, FldFin -> [ModGrp]

```
DimLim: RngIntElt                    Default: 2000
```

Given a finite group G and a finite field K, this function constructs the absolutely irreducible $K[G]$-modules over extensions of K. Currently, the group G is restricted to a permutation group. The maximum dimension of a module considered for splitting is controlled by the parameter `DimLim`, which has default value 2000.

`IrreducibleModulesBurnside(G, K : parameters): Grp, FldFin -> [ ModGrp ]`

```
DimLim: RngIntElt                    Default: 2000
```

Given a finite group G and a finite field K, this function constructs the irreducible $K[G]$-modules over K. Currently, the group G is restricted to a permutation group. The maximum dimension of a module considered for splitting is controlled by the parameter `DimLim`, which has default value 2000.

`AbsolutelyIrreducibleConstituents(M): ModGrp -> [ ModGrp ]`

Given an irreducible module M, return M if it is already absolutely irreducible, else return the absolutely irreducible modules obtained by finding the constituents of M after extending the base field of M to a splitting field.

Example: Irreducible Modules for M11 (ex-7aff4e)

We find all irreducible modules for M_{11} over $GF(2)$, and all absolutely irreducible modules of characteristic 2. The Burnside algorithm is used by default.

```
> load m11;
Loading "/home/magma/libs/pergps/m11"
M11 - Mathieu group on 11 letters - degree 11
Order 7 920 = 2^4 * 3^2 * 5 * 11;  Base 1,2,3,4
Group: G
> IrreducibleModules(G, GF(2));
[
  GModule of dimension 1 over GF(2),
  GModule of dimension 10 over GF(2),
  GModule of dimension 44 over GF(2),
  GModule of dimension 32 over GF(2)
]
> AbsolutelyIrreducibleModules(G, GF(2));
[
  GModule of dimension 1 over GF(2),
  GModule of dimension 10 over GF(2),
  GModule of dimension 44 over GF(2),
  GModule of dimension 16 over GF(2^2),
  GModule of dimension 16 over GF(2^2)
]
```

99.19.2 Irreducible Modules over F_q for Soluble Groups

This collection of functions allows the user to find all irreducible modules of a finite soluble group. The group is first replaced by an isomorphic group defined by a power-conjugate presentation. The irreducibles are then found using Schur's method of working up the composition series for G defined by the pc-presentation.

`IrreducibleModules(G, K : parameters): Grp, Fld -> SeqEnum`

`AbsolutelyIrreducibleModulesSchur(G, K: parameters): GrpPC, Rng -> List[GMod ule]`

`AbsolutelyIrreducibleModulesSchur(G, k, i: parameters): GrpPC, Rng, RngIntElt -> List[GModule]`

```
AbsolutelyIrreducibleModulesSchur(G, k, L: parameters): GrpPC, Rng, List[GModule] ->
                                  List[GModule]
AbsolutelyIrreducibleModulesSchur(G, k, L, i: parameters): GrpPC, Rng, RngIntElt ->
                                  List[GModule]
```

```
Process       : BoolElt                    Default: true
GaloisAction  : MonStgElt                  Default: "Yes"
MaxDimension  : RngIntElt                  Default:
ExactDimension: SetEnum                    Default:
```

Let G be a finite soluble group and let K be one the following types of field: a finite field, the rational field, or a cyclotomic field. The order of a cyclotomic field must divide the exponent of G. The function constructs all absolutely irreducible representations of G over appropriate extensions or subfields of the field K. The modules returned are non-isomorphic and consist of all distinct modules, subject to the conditions imposed. In the case when K is a finite field, the Glasby-Howlett algorithm is used to determine the minimal field over which an irreducible module may be realised. If K has characteristic 0, the field over which an irreducible module is given may not be minimal.

The irreducible modules are found using Schur's method of climbing the composition series for G defined by the pc-presentation.

If the parameter `Process` is set true then the list is a list of pairs comprising an integer and a representation. This list or any sublist of it is a suitable value for the argument L in the last versions of the function, and in this case only the representations in L will be extended up the series. This allows the user to inspect the representations produced along the way and cull any that are uninteresting.

Possible values are "Yes", "No", and "Relative". The default is "Yes" for intermediate levels and "No" for the whole group. The value "Yes" means that it only lists one irreducible from each orbit of the action of the absolute Galois group $Gal(K/\text{primefield}(K))$. Setting this parameter to "No" turns this reduction off (thus listing all inequivalent representations), while setting it to "Relative" uses the group $Gal(K/k)$.

Restrict the irreducible to those of dimension $\leq$ MaxDimension. Default is no restriction.

If `ExactDimension` is assigned a set S of positive integers, attention is restricted to irreducible having dimensions lying in the set S. The default is equivalent to taking the set of all positive integers.

If both `MaxDimension` and `ExactDimension` are assigned values, then irreducible having dimensions that are either bounded by `MaxDimension` or contained in `ExactDimension` are produced.

```
IrreducibleModulesSchur(G, K: parameters): GrpPC, Rng -> List[GModule]
IrreducibleModulesSchur(G, K, i: parameters): GrpPC, Rng, RngIntElt -> List[GModule]
IrreducibleModulesSchur(G, K, L: parameters): GrpPC, Rng, List[GModule] ->
                        List[GModule]
IrreducibleModulesSchur(G, K, L, i: parameters): GrpPC, Rng, List[GModule], RngIntElt
                        -> List[GModule]
```

Compute irreducible modules for G over the given field K. All arguments and parameters are as for the absolutely irreducible case.

The computation proceeds by first computing the absolutely irreducible representations subject to the given conditions, then rewriting over the field K, with a consequent change of dimension of the representation.

Example: Representations of a Dihedral Group of Order 20 (ex-a55c65)

We compute representations of the dihedral group of order 20.

```
> G := DihedralGroup(GrpPC, 10);
> FactoredOrder(G);
[ <2, 2>, <5, 1> ]
```

First some modular representations with characteristic 2.

```
> r := IrreducibleModulesSchur(G, GF(2));
> r;
[*
    GModule of dimension 1 over GF(2),
    GModule of dimension 4 over GF(2)
*]
> r := AbsolutelyIrreducibleModulesSchur(G, GF(2));
> r;
[*
    GModule of dimension 1 over GF(2),
    GModule of dimension 2 over GF(2^2),
    GModule of dimension 2 over GF(2^2)
*]
> r := AbsolutelyIrreducibleModulesSchur(G, GF(2) : GaloisAction:="Yes");
> r;
[*
    GModule of dimension 1 over GF(2),
    GModule of dimension 2 over GF(2^2)
*]
```

The irreducible representation of dimension 4 is not absolutely irreducible, as over $GF(4)$ it splits into two Galois-equivalent representations.

Finding irreducible modules over the complex field is straightforward, despite not being able to use the complex field as the field argument. We could instead specify the cyclotomic field having order equal to the exponent of G, but it is preferable to ask for all absolutely irreducible modules over the rationals.

```
> r := AbsolutelyIrreducibleModulesSchur(G, Rationals());
> r;
[*
   GModule of dimension 1 over Rational Field,
   GModule of dimension 1 over Rational Field,
   GModule of dimension 1 over Rational Field,
   GModule of dimension 1 over Rational Field,
   GModule of dimension 2 over Cyclotomic Field of order 5 and degree 4,
   GModule of dimension 2 over Cyclotomic Field of order 5 and degree 4,
   GModule of dimension 2 over Cyclotomic Field of order 5 and degree 4,
   GModule of dimension 2 over Cyclotomic Field of order 5 and degree 4
*]
> Representation(r[6])(G.2);
[zeta_5^3        0]
[       0 zeta_5^2]
```

99.19.3 Irreducible Modules over $\mathbb{Q}$ for Arbitrary Groups

Magma V2.16 contains a new algorithm for constructing irreducible rational representations of an arbitrary finite group. This algorithm is now selected by default in the function `IrreducibleModules(G, K)` when the field K is $\mathbb{Q}$. Given an irreducible complex character for a group G, the sum of its Galois orbit gives an *irreducible rational character* for G. The Schur index of an irreducible rational character χ is defined to be the Schur index of an irreducible complex character whose Galois orbit sum is χ. We call the sequence of all possible irreducible rational characters for G (sorted by degree) the *rational character*

table of G. The irreducible modules returned by `IrreducibleModules(G, RationalField())` always match the rational character table.

`IrreducibleModules(G, Q : parameters): Grp, FldRat -> SeqEnum, SeqEnum`

```
MaxDegree       : RngIntElt                    Default: 0
Characters      : [AlgChtrElt]                 Default:
CharacterDegrees: [RngIntElt]                  Default:
UseInduction    : BoolElt                      Default: true
```

Given a finite group G, this function returns a sequence L of all irreducible rational $K[G]$-modules, and also the rational character table of G as a sequence C, which matches L. The character of the i-th module $L[i]$ is always $s_i \cdot C[i]$, where s_i is the Schur index of $C[i]$. Thus the dimension of the irreducible module $L[i]$ is $s_i \cdot \mathrm{Deg}(C[i])$.

Setting the parameter `MaxDegree` to a positive integer D instructs the algorithm not to spend effort on constructing irreducible modules whose corresponding irreducible rational characters have degree greater than D. Note that irreducible modules whose character degrees greater than D may be returned in any case, if they are easily constructed (often as side effects of operations used to construct smaller-dimensional modules).

Setting the parameter `Characters` to a set or sequence S of characters for the group G instructs the algorithm to attempt to construct only the irreducible modules whose characters are in S. Each character in S may either be an irreducible rational character, or a (complex) ordinary character χ, in which case the irreducible rational character corresponding to χ (the orbit sum of χ) is used. As for the parameter `MaxDegree`, irreducible modules whose characters are not in S may be returned in any case, if they are easily constructed or are needed as intermediate modules to construct the desired modules.

Setting the parameter `CharacterDegrees` to a set or sequence I of positive integers instructs the algorithm to attempt to construct only the irreducible modules whose character degrees are in S. This is equivalent to setting the parameter `Characters` to `[c: c in RationalCharacterTable(G) | Degree(c) in I]`.

By default, as the algorithm proceeds, it automatically searches for subgroups H_i of G such that irreducible rational H_i-modules may be induced up to G to yield G-modules from which irreducible G-modules may be computed. In general, this method is very effective (and often yields modules for G with small entries) but can be very slow for some groups (particularly when G has many subgroups). Thus setting the parameter `UseInduction` to `false` will force the algorithm not to use induction.

`RationalCharacterTable(G): Grp -> SeqEnum, SeqEnum`

Given a finite group G, return the rational character table of G as a sequence C of the irreducible rational characters of G (sorted by degree), and also a index sequence I, such that for each i, $C[i]$ is the sum of the Galois orbit of $T[I[i]]$, where T is the ordinary (complex) character table of G.

Example: Rational Irreducible Modules for PSL(3,3) `(ex-c96e06)`

We compute all irreducible rational modules for $\mathrm{PSL}(3,3)$ and note that the characters of the resulting modules match the entries in the rational character table.

```
> G := PSL(3, 3); #G;
5616
> T := CharacterTable(G); [Degree(x): x in T];
[ 1, 12, 13, 16, 16, 16, 16, 26, 26, 26, 27, 39 ]
> C := RationalCharacterTable(G); C;
[
    ( 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 ),
    ( 12, 4, 3, 0, 0, 1, 0, 0, -1, -1, -1, -1 ),
    ( 13, -3, 4, 1, 1, 0, -1, -1, 0, 0, 0, 0 ),
    ( 26, 2, -1, -1, 2, -1, 0, 0, 0, 0, 0, 0 ),
```

(continues on next page)

(continued from previous page)

```
    ( 27, 3, 0, 0, -1, 0, -1, -1, 1, 1, 1, 1 ),
    ( 39, -1, 3, 0, -1, -1, 1, 1, 0, 0, 0, 0 ),
    ( 52, -4, -2, -2, 0, 2, 0, 0, 0, 0, 0, 0 ),
    ( 64, 0, -8, 4, 0, 0, 0, 0, -1, -1, -1, -1 )
]
> time L := IrreducibleModules(G, RationalField());
Time: 0.760
> L;
[
    GModule of dimension 1 over Rational Field,
    GModule of dimension 12 over Rational Field,
    GModule of dimension 13 over Rational Field,
    GModule of dimension 26 over Rational Field,
    GModule of dimension 27 over Rational Field,
    GModule of dimension 39 over Rational Field,
    GModule of dimension 52 over Rational Field,
    GModule of dimension 64 over Rational Field
]
> [Character(M): M in L];
[
    ( 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 ),
    ( 12, 4, 3, 0, 0, 1, 0, 0, -1, -1, -1, -1 ),
    ( 13, -3, 4, 1, 1, 0, -1, -1, 0, 0, 0, 0 ),
    ( 26, 2, -1, -1, 2, -1, 0, 0, 0, 0, 0, 0 ),
    ( 27, 3, 0, 0, -1, 0, -1, -1, 1, 1, 1, 1 ),
    ( 39, -1, 3, 0, -1, -1, 1, 1, 0, 0, 0, 0 ),
    ( 52, -4, -2, -2, 0, 2, 0, 0, 0, 0, 0, 0 ),
    ( 64, 0, -8, 4, 0, 0, 0, 0, -1, -1, -1, -1 )
]
```

Example: Irreducible Rational Modules for M23 `(ex-011efd)`

For large groups, one can use the parameter `MaxDegree` to compute the irreducible modules of reasonable dimension.

```
> load m23;
Loading "/home/magma/libs/pergps/m23"
M23 - Mathieu group on 23 letters - degree 23
Order 10 200 960 = 2^7 * 3^2 * 5 * 7 * 11 * 23;  Base 1,2,3,4,5,6
Group: G
> T := CharacterTable(G); [Degree(x): x in T];
[ 1, 22, 45, 45, 230, 231, 231, 231, 253, 770, 770,
896, 896, 990, 990, 1035, 2024 ]
> C := RationalCharacterTable(G); C;
[
    ( 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 ),
    ( 22, 6, 4, 2, 2, 0, 1, 1, 0, 0, 0, -1, -1, -1, -1, -1, -1 ),
    ( 90, -6, 0, 2, 0, 0, -1, -1, -2, 2, 2, 1, 1, 0, 0, -2, -2 ),
    ( 230, 22, 5, 2, 0, 1, -1, -1, 0, -1, -1, 1, 1, 0, 0, 0, 0 ),
    ( 231, 7, 6, -1, 1, -2, 0, 0, -1, 0, 0, 0, 0, 1, 1, 1, 1 ),
    ( 253, 13, 1, 1, -2, 1, 1, 1, -1, 0, 0, -1, -1, 1, 1, 0, 0 ),
    ( 462, 14, -6, -2, 2, 2, 0, 0, -2, 0, 0, 0, 0, -1, -1, 2, 2 ),
    ( 1035, 27, 0, -1, 0, 0, -1, -1, 1, 1, 1, -1, -1, 0, 0, 0, 0 ),
    ( 1540, -28, 10, -4, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, -1 ),
    ( 1792, 0, -8, 0, 2, 0, 0, 0, 0, -1, -1, 0, 0, 2, 2, -2, -2 ),
    ( 1980, -36, 0, 4, 0, 0, -1, -1, 0, 0, 0, -1, -1, 0, 0, 2, 2 ),
```

(continues on next page)

(continued from previous page)

```
    ( 2024, 8, -1, 0, -1, -1, 1, 1, 0, 0, 0, 1, 1, -1, -1, 0, 0 )
]
> Q := RationalField();
> time L := IrreducibleModules(G, Q: MaxDegree := 253);
Time: 23.400
> L;
[
    GModule of dimension 1 over Rational Field,
    GModule of dimension 22 over Rational Field,
    GModule of dimension 90 over Rational Field,
    GModule of dimension 230 over Rational Field,
    GModule of dimension 231 over Rational Field,
    GModule of dimension 253 over Rational Field,
    undef,
    undef,
    undef,
    undef,
    GModule of dimension 1980 over Rational Field
]
```

Note that the module of dimension 1980 is included (it was easily constructed as the tensor product of modules of dimension 20 and 90).

99.20 Modules over a General Algebra

99.20.1 Introduction

This section describes the functionality for modules over general algebras in Magma and is independent of the earlier chapters. A left-module over an algebra A is a module M together with a bilinear map $A \times M \to M$. A right-module over A is a module M together with a bilinear map $M \times A \to M$. Magma provides functionality for both kinds of modules.

99.20.2 Construction of Algebra Modules

`Module(A, m): Alg, Map[SetCart, ModRng] -> ModAlg`

For an algebra A this function creates a module over A. If the module will be a left-module then m is a map from the Cartesian product $A \times M$ to M. If the module will be a right-module then m is a map from $M \times A$ to M. Here M has to be an R-module, where R is the coefficient field of A.

Example: Natural Module of a Matrix Algebra (ex-dc2928)

We create the right-module over the full matrix algebra of 3×3 - matrices acting on its natural module.

```
> A:= MatrixAlgebra(Rationals(), 3);
> V:= RModule(Rationals(), 3);
> m:= map< CartesianProduct(V, A) -> V | t :-> t[1]*t[2] >;
> Module(A, m);
Right Module of Full Matrix Algebra of degree 3 over Rational Field
```

99.20.3 The Action of an Algebra Element

`a ^ v: AlgElt, ModAlgElt -> ModAlgElt`

Given an element v of a left-module over an algebra A, and an element a of A computes the result of letting a act on v.

`v ^ a: ModAlgElt, AlgElt -> ModAlgElt`

Given an element v of a right-module over an algebra A and an element a of A computes the result of letting a act on v.

`ActionMatrix(M, a): ModAlg, AlgElt -> AlgMatElt`

Given a module M over an algebra A and an element a of A returns the matrix of the action of a on M. If M is a left-module then the i-th column of this matrix contains the coordinates of the image of a acting on the i-th basis element of M. If A is a right-module then the rows contain these coordinates.

Example: Action Matrices of Algebra Elements (ex-72e8cf)

```
> A:= MatrixAlgebra(Rationals(), 3);
> V:= RModule(Rationals(), 3);
> m:= map< CartesianProduct(V, A) -> V | t :-> t[1]*t[2] >;
> M:=Module(A, m);
> M.1^A.1;
M: (1 0 0)
> ActionMatrix(M, A.2);
[0 1 0]
[0 0 1]
[1 0 0]
```

99.20.4 Related Structures of an Algebra Module

`Algebra(M): ModAlg -> Alg`

This returns the algebra over which the algebra module M is defined.

`CoefficientRing(M): ModAlg -> Fld`

Returns the ground field of the algebra module M.

`Basis(M): ModAlg -> SeqEnum`

Returns a sequence containing the basis vectors of the algebra module M.

99.20.5 Properties of an Algebra Module

`IsLeftModule(M): ModAlg -> BoolElt`

This returns `true` if the algebra module M is a left-module, and `false` if it is a right module.

`IsRightModule(M): ModAlg -> BoolElt`

This returns `true` if the algebra module M is a right-module, and `false` if it is a left module.

`Dimension(M): ModAlg -> RngIntElt`

The dimension of the algebra module M.

99.20.6 Creation of Algebra Modules from other Algebra Modules

`DirectSum(Q): SeqEnum -> ModAlg, SeqEnum, SeqEnum`

Given a sequence Q of algebra modules (all defined over the same algebra, and all left (respectively right) modules), returns the module M that is the direct sum of the modules in Q. Furthermore, two sequences of mappings are returned. The i-th element of the first sequence is the embedding of the i-th element of Q into M. The i-th element of the second sequence is the projection of M onto the i-th element of Q.

`SubalgebraModule(B, M): Alg, ModAlg -> ModAlg`

Given an algebra module M over the algebra A, and a subalgebra B of A, return M as a B-module.

`ModuleWithBasis(Q): SeqEnum -> ModAlg`

Given a sequence Q containing the elements of a particular basis of an algebra module M, create an algebra module that is isomorphic to M, but with basis Q. (Or, more precisely, the basis vectors of the module V that is returned are in bijection with Q. The action of an algebra element on the i-th basis vector of V is computed by computing it on the i-th vector in Q and expressing the result as a linear combination of the elements of Q. The resulting coordinates are used to form the corresponding element of V.) This can be used to compute the action of algebra elements with respect to a given basis of M.

Example: Direct Sums of Algebra Modules (ex-720705)

```
> A:= MatrixAlgebra(Rationals(), 3);
> V:= RModule(Rationals(), 3);
> m:= map< CartesianProduct(V, A) -> V | t :-> t[1]*t[2] >;
> M:=Module(A, m);
> N:=DirectSum([ M, M ]);
> ActionMatrix(N, A.1);
[1 0 0 0 0 0]
[0 0 0 0 0 0]
[0 0 0 0 0 0]
[0 0 0 1 0 0]
[0 0 0 0 0 0]
[0 0 0 0 0 0]
> W:= ModuleWithBasis([ M.1+M.2+M.3, M.2+M.3, M.3 ]);
> ActionMatrix(W, A.1);
[ 1 -1  0]
[ 0  0  0]
[ 0  0  0]
```

`sub< M | S >: ModAlg, [ModAlgElt] -> ModAlg`

`sub< M | e1, ..., en >: ModAlg, ModAlgElt, ..., ModAlgElt -> ModAlg`

Return the submodule of M containing the elements in the sequence S or the elements $e1$, ..., en.

`quo< M | S >: ModAlg, [ModAlgElt] -> ModAlg`

`quo< M | e1, ..., en >: ModAlg, ModAlgElt, ..., ModAlgElt -> ModAlg`

`quo< M | S >: ModAlg, ModAlg -> ModAlg`

Construct the quotient module of M by the submodule S of M, the submodule containing the elements in the sequence S or the elements $e1, ..., en$.

99.21 Bibliography

- [BW15] Peter A. Brooksbank and James B. Wilson. The module isomorphism problem reconsidered. *J. Algebra*, 421:541–559, 2015. URL: https://doi.org/10.1016/j.jalgebra.2014.09.004, doi:10.1016/j.jalgebra.2014.09.004.
- [DK09] Susanne Danz and Burkhard Külshammer. Vertices, sources and Green correspondents of the simple modules for the large Mathieu groups. *J. Algebra*, 322(11):3919–3949, 2009.
- [DKZ08] Susanne Danz, Burkhard Külshammer, and René Zimmermann. On vertices of simple modules for symmetric groups of small degrees. *J. Algebra*, 320(2):680–707, 2008.
- [Gre59] J.A. Green. On the indecomposable representations of a finite group. *Math. Z.*, 70:430–445, 1958/59.

Chapter 100

Characters of Finite Groups

Assume that G is a finite group of exponent m with k conjugacy classes of elements. The operators discussed here are concerned with the ring of *class functions on* G, defined to be the ring of complex-valued functions on G that are constant on conjugacy classes. This ring is made into a $\mathbb{C}$-algebra by identifying $c \in \mathbb{C}$ with the constant function that is c everywhere. In fact we will restrict ourselves to functions with values that are elements of cyclotomic fields. Elements of the ring, that is, objects of type `AlgChtrElt`, are represented by the k values (elements of some cyclotomic field $\mathbb{Q}(\zeta_n)$) on the classes. The numbering of those elements matches the numbering of the classes as returned by `Classes` applied to the underlying group `G`: Thus `X[i]` is the value of the character `X` on the i-th class, i.e. `Classes(G)[i]`.

100.1 Creation Functions

100.1.1 Structure Creation

`ClassFunctionSpace(G): Grp -> AlgChtr`

`CharacterRing(G): Grp -> AlgChtr`

Given a finite group G, create the ring of complex-valued class functions. This function will trigger the computation of the conjugacy classes of G if these are not yet known. Information about the irreducible characters is stored in the ring when it is computed.

`ClassFunctionSpace(Q): SeqEnum -> AlgChtr`

`CharacterRing(Q): SeqEnum -> AlgChtr`

Construct the ring of complex-valued class functions for a group that has classes data as given in Q. The sequence Q must be a sequence of pairs `<o, n>`, each representing one conjugacy class, where o is the order of the elements in the class, and n is the length of the class. The first class must be the class of the group identity element. This is a sequence as returned by the *ClassesData* intrinsic.

This function allows the creation of a space to work with characters without the attached group.

100.1.2 Element Creation

The elementary constructions for class functions are listed. Other useful ways of defining class functions and characters are defined in sections discussing the permutation character, the (de)composition functions, and the sections on the conjugating, restricting, and inducing of class functions.

`elt< R | a`$_1$`, ..., a`$_k$` :parameters>: AlgChtr, FldCycElt, ..., FldCycElt -> AlgChtrElt`

`R ! [ a`$_1$`, ..., a`$_k$` ]: AlgChtr, SeqEnum -> AlgChtrElt`

Character: BoolElt	Default: false

Given the ring of class functions R of a finite group G with k conjugacy classes and k elements a_i contained in some common cyclotomic field, create a class function on G for which the value on the i-th class is equal to the i-th term a_i.

If `Character` is set to `true`, then the resulting character is flagged to be a proper character.

`R ! a: AlgChtr, RngIntElt -> AlgChtrElt`

`R ! a: AlgChtr, FldRatElt -> AlgChtrElt`

`R ! a: AlgChtr, FldCycElt -> AlgChtrElt`

Define a constant class function for the ring of class functions R of the group G. Here a is allowed to be an integer, a rational field element, or a cyclotomic field element.

`Id(R): AlgChtr -> AlgChtrElt`

`Identity(R): AlgChtr -> AlgChtrElt`

`One(R): AlgChtr -> AlgChtrElt`

`PrincipalCharacter(G): Grp -> AlgChtrElt`

Given the finite group G or its ring of class functions R, create the principal character (which takes on the value 1 on every element of G).

`Zero(R): AlgChtr -> AlgChtrElt`

Given a ring of class functions R create its zero element (which is the class function that takes on the value 0 on every element of the group).

100.2 The Table of Irreducible Characters

The function `CharacterTable` can be invoked to determine the complete or a partial table of irreducible characters on a finite group G. If necessary, an existing character table can be supplemented by another call to the function. The known irreducible characters are stored in the character ring of G.

The default algorithm for character tables is Unger's Induce/Reduce algorithm, described in detail in [Ung06a]. Computing character tables assumes that the classes of the group can be computed and stored, along with a class representative for each class, as well as the power map of the group. Unger's algorithm constructs elementary subgroups of the group (where elementary means direct product of cyclic group with p-group), and constructs characters of these subgroups (using Conlon's algorithm for the character table of the p-part of the group). Following Brauer's theorem on induced characters, these characters are induced to the full group and all irreducible characters of the full group are found in the integral span of the induced characters (using LLL reduction to maintain a manageable basis of the character space found). The two potentially most time-consuming parts of the algorithm are the computation of fusion in the elementary subgroups, which is necessary for induction, and repeated use of LLL to maintain a basis of the space of generalised characters found to date. Arithmetic with generalised characters is done quickly by using a finite field representation of these characters, as in Dixon's work [Dix67], but here the prime used may be twice the length of Dixon's.

The Dixon-Schneider algorithm for computing character tables [Dix67, Sch90b] is also available. See below for details on how to access it. This was the previous default algorithm for character tables. As indicated above, Conlon's algorithm for the character table of a p-group [Con90a] is also implemented, and may be accessed directly as indicated below.

The functions in this section return character tables, which are enumerated sequences of characters that are flagged to allow printing in a special format.

`KnownIrreducibles(R): AlgChtr -> SeqEnum`

`KnownIrreducibles(G): Grp -> SeqEnum`

Given a finite group G, or a class function space R of G, return the table of irreducible characters currently stored. Such a table is a sequence of characters with specially formatted printing.

It should be noted that characters are stored with some information about whether they are characters, generalized characters, or class functions. From this information, it is often possible to deduce with little effort that a character is, in fact, irreducible. When a new irreducible is found this way, it is immediately inserted into the table of irreducible characters.

`CharacterTable(G :parameters): Grp -> SeqEnum`

```
Al          : MonStgElt                    Default: "Default"
DSSizeLimit: RngIntElt                     Default: 0
```

Construct the table of ordinary irreducible characters for the group G.

This parameter controls the algorithm used. The string `"DS"` forces use of the Dixon-Schneider algorithm. The string `"IR"` forces the use of Unger's Induce/Reduce algorithm [Ung06a]. The string `"Conlon"` forces the use of Conlon's algorithm. This last is valid only when the group is a p-group. The `"Default"` algorithm is to use Dixon-Schneider for groups of order ≤ 5000, Conlon's algorithm for larger p-groups, and Unger's algorithm for other groups. This may change in future.

When the algorithm selected is either "Default" or "IR", a positive value n for `DSSizeLimit` means that before using ithe Induce/Reduce algorithm, the full possible character space is split by some passes of the Dixon-Schneider algorithm, restricted to using class matrices corresponding to conjugacy classes with size at most n.

`CharacterTableDS(G :parameters): Grp -> SeqEnum, SeqEnum`

```
ClassMatrices   : SeqEnum                  Default: []
ClassMatrixLimit: RngIntElt                Default: Infinity()
ClassSizeLimit  : RngIntElt                Default: Infinity()
MinChars        : RngIntElt                Default: Infinity()
Modulus         : RngIntElt                Default: 0
```

Given a finite group G, construct the table of irreducible characters of G using the Dixon-Schneider algorithm [Dix67, Sch90b]. The conjugacy classes of G will be computed if necessary; if the user wishes to exert control over the computation of the classes, the function `Classes` can be invoked on G before calling `CharacterTableDS`.

There are several optional parameters for `CharacterTableDS`. Assigning a non-negative integer value m to `MinChars`, indicates that the computation is to be terminated as soon as m or more additional characters have been found. By setting `ClassMatrixLimit` to a positive integer n, the user can limit the number of class matrices used. If the calculation of all irreducible characters (or at least m of them if `MinChars` has been set) cannot be completed by using k class matrices, an incomplete character table is returned. Setting `ClassSizeLimit` to an integer n restricts the algorithm to computing class matrices corresponding to classes of size at most n. Again, if the calculation of all irreducible characters cannot be completed using class matrices from these small classes, an incomplete character table is returned. The optional argument `ClassMatrices` (a sequence of integers in the range $1 \ldots k$, where k is the number of conjugacy classes of G) can be used to specify a preference for the order in which class matrices should be used. Finally the `Modulus` argument can be used to set which prime field is used for the internal computations of the Dixon-Schneider algorithm. The value used must be a prime equivalent to 1 modulo the group exponent and greater than twice the square root of the group order.

The second return argument is the sequence of unsplit character spaces remaining at the point the algorithm terminated. This is empty when the algorithm returns a complete character table, but may be useful when an incomplete character table is returned.

`Basis(R): AlgChtr -> SeqEnum`

The function `Basis` takes the character ring R of G as input and performs the same computation to find the basis for R consisting of the irreducible characters.

Both `CharacterTable` and `Basis` return a character table, which is an enumerated sequence of elements of the character ring over G (if necessary, the ring is created) that only differs from arbitrary sequences of class functions with respect to printing.

`CharacterTableConlon(G): Grp -> SeqEnum`

Compute the character table of group G using Conlon's algorithm for p-groups. The group G must thus be a p-group for some prime p.

`LinearCharacters(G): Grp -> SeqEnum`

Given a finite group G, determine the (partial) character table containing only the linear characters.

`CharacterDegrees(G): GrpPerm -> SeqEnum`

`CharacterDegrees(G): GrpPC -> SeqEnum`

`CharacterDegrees(G): GrpMat -> SeqEnum`

`CharacterDegrees(G): GrpAb -> SeqEnum`

Returns the degrees of the ordinary irreducible characters of the finite group G. The sequence returned has form $[< d_1, c_1 >, < d_2, c_2 >, \dots]$ where c_i is the number of characters of degree d_i. For p-groups Slattery's algorithm is used, for other soluble groups Conlon's counting algorithm is used, and for insoluble groups the character table of G is computed.

`CharacterDegrees(G, z, p): GrpPC, GrpPCElt, RngIntElt -> SeqEnum`

Returns the degrees of the absolutely irreducible characters of G lying over a faithful linear character of $\langle z \rangle$, where z is a central element of G and p is zero or a prime.

`CharacterDegreesPGroup(G): GrpPC -> SeqEnum`

Returns the degrees of the ordinary irreducible characters of the finite p-group G. The sequence returned has form $[c_0, c_1, c_2 \dots]$, where c_i is the number of characters of degree p^i. Slattery's counting algorithm is used.

`RationalCharacterTable(G): GrpFin -> SeqEnum`

Returns a sequence of minimal rational characters of G. These are the sums of the Galois orbits on the character table of G.

`SaveCharacterTable(T, F): SeqEnum[AlgChtrElt], MonStgElt`

Write the character table T to the file named by the string F. The command `> T := eval Read(F);` will read T in, as a character table without group attached.

Example: Printing a Character Table (ex-1eda96)

We use the `CharacterTable` function and print the resulting table. Character tables have a special print format.

```
> G := Alt(5);
> CT := CharacterTable(G);
> CT;


Character Table of Group G
--------------------------



---------------------------
Class |   1  2  3    4    5
Size  |   1 15 20   12   12
Order |   1  2  3    5    5
---------------------------
p  =  2   1  1  3    5    4
p  =  3   1  2  1    5    4
p  =  5   1  2  3    1    1
---------------------------
X.1   +   1  1  1    1    1
X.2   +   3 -1  0   Z1 Z1#2
X.3   +   3 -1  0 Z1#2   Z1
X.4   +   4  0  1   -1   -1
X.5   +   5  1 -1    0    0
```

(continues on next page)

(continued from previous page)

```
Explanation of Character Value Symbols
--------------------------------------

# denotes algebraic conjugation
#k indicates replacing the root of unity w by w^k

Z1     = (CyclotomicField(5: Sparse := true)) ! [
RationalField() | 1, 0, 1, 1 ]

> CT[2];
( 3, -1, 0, zeta(5)_5^3 + zeta(5)_5^2 + 1, -zeta(5)_5^3 -
  zeta(5)_5^2 )
> CT[2]:Minimal;
( 3, -1, 0, Z1, Z1#2 )
```

The character table is represented as a sequence of characters. Non-rational values in a character table are printed symbolically. This same printing can be forced on an individual character by using `Minimal` printing.

Example: Character Table of an Almost Simple Group (ex-be592e)

The `CharacterTable` algorithm can handle quite large groups. We illustrate this by computing the character table of the almost simple group $P\Gamma U_5(4)$.

```
> G := PGammaU(5,4);
> G;
Permutation group G acting on a set of cardinality 17425
Order = 2^22 * 3^2 * 5^5 * 13 * 17 * 41
> time CT := CharacterTable(G);
Time: 22.220
> #CT;
160
> Degree(CT[160]);
5227500
```

100.3 Character Ring Operations

100.3.1 Related Structures

`Parent(R): AlgChtr -> Pow`

`Category(R): AlgChtr -> Cat`

`NumberOfClasses(R): AlgChtr -> RngIntElt`

`Nclasses(R): AlgChtr -> RngIntElt`

Given the ring R of class functions, return the number of conjugacy classes of the finite group underlying R.

`ClassesData(R): AlgChtr -> SeqEnum[Tup]`

Given the ring R of class functions, return a sequence of pairs, one pair `<o,n>` for each conjugacy class of the underlying group, where o is the order of the group elements in the class, and n is the length of (number of group elements in) the class.

`Group(R): AlgChtr -> Grp`

Given the ring R of class functions on a finite group G, return G. This will cause an error if R does not have a group attached. If unsure whether or not R has a group attached, check the `Group` attribute of R using `assigned R`Group`.

`PowerMap(R): AlgChtr -> Map`

Given a character ring R, return the associated class power map. This will cause an error unless R has a power map assigned, or has a group attached. The `PowerMap` is an attribute of R, as is `Group`, and its presence can be checked using `assigned R`PowerMap`.

If the power map of R is not already assigned, and there is an assigned group, this function will compute the power map using operations within the group.

`Kernel(x): AlgChtrElt -> Grp`

The kernel of the character x of G, i.e. the normal subgroup of G consisting of those elements g for which $x(g) = x(1)$.

`Centre(x): AlgChtrElt -> Grp`

The centre of the character x of G, i.e. the subgroup of G consisting of those classes C of G for which $|x(g)|$, g in C, is equal to the degree of x.

`CoefficientField(x): AlgChtrElt -> Rng`

The minimal cyclotomic field containing all values of the class function x.

`CharacterField(x): AlgChtrElt -> Rng`

The subfield of the coefficient field of the class function x that is generated by the values of x.

`DegreeOfCharacterField(x): AlgChtrElt -> RngIntElt`

The degree of the character field of the class function x as an extension of the rational numbers.

100.4 Element Operations

100.4.1 Arithmetic

In the list of arithmetic operations below x and y denote class functions in the same ring, and a denotes a scalar, which is any element coercible into a cyclotomic field. Also, j denotes an integer.

`+ y: AlgChtrElt -> AlgChtrElt`

`- y: AlgChtrElt -> AlgChtrElt`

`x + y: AlgChtrElt, AlgChtrElt -> AlgChtrElt`

`x - y: AlgChtrElt, AlgChtrElt -> AlgChtrElt`

`x * y: AlgChtrElt, AlgChtrElt -> AlgChtrElt`

`a * x: FldCycElt, AlgChtrElt -> AlgChtrElt`

`x ^ j: AlgChtrElt, RngIntElt -> AlgChtrElt`

100.4.2 Predicates and Booleans

The following Boolean-valued functions are available. Note that with the exception of `in`, `notin`, `IsReal`, and `IsFaithful`, these functions use the table of irreducible characters, which will be created if it is not yet available.

`x in y: AlgChtrElt, AlgChtrElt -> BoolElt`

Returns `true` if the inner product of class functions x and y is non-zero, otherwise `false`. If x is irreducible and y is a character, this tests whether or not x is a constituent of y.

`x notin y: AlgChtrElt, AlgChtrElt -> BoolElt`

Returns `true` if the inner product of class functions x and y is zero, otherwise `false`. If x is irreducible and y is a character, this tests whether or not x is not a constituent of y. Returns `true` if the character x is not a constituent of the character y, otherwise `false`.

`a in F: FldFunElt, FldFun -> BoolElt`

`a notin F: FldFunElt, FldFun -> BoolElt`

`x eq y: AlgChtrElt, AlgChtrElt -> BoolElt`

`x ne y: AlgChtrElt, AlgChtrElt -> BoolElt`

`IsCharacter(x): AlgChtrElt -> BoolElt`

Returns `true` if the class function x is a character, otherwise `false`. A class function is a character if and only if all inner products with the irreducible characters are non-negative integers.

`IsGeneralizedCharacter(x): AlgChtrElt -> BoolElt`

Returns `true` if the class function x is a generalized character, otherwise `false`. A class function is a generalized character if and only if all inner products with the irreducible characters are integers.

`IsIrreducible(x): AlgChtrElt -> BoolElt`

Returns `true` if the character x is an irreducible character, otherwise `false`.

`IsLinear(x): AlgChtrElt -> BoolElt`

Returns `true` if the character x is a linear character, otherwise `false`.

`IsFaithful(x): AlgChtrElt -> BoolElt`

Returns `true` if the character x is faithful, i.e. has trivial kernel, otherwise `false`.

`IsReal(x): AlgChtrElt -> BoolElt`

Returns `true` if the character x is a real character, i.e. takes real values on all of the classes of G, otherwise `false`.

`IsOne(x): AlgChtrElt -> BoolElt`

`IsMinusOne(x): AlgChtrElt -> BoolElt`

`IsZero(x): AlgChtrElt -> BoolElt`

`IsPermutationCharacter(c): AlgChtrElt -> BoolElt`

Test if a character c of G is a virtual permutation character, that is $\mathbb{C}[X] - \mathbb{C}[Y]$ for some G-sets X and Y. (The implementation relies on `BurnsideCokernel`, and does not compute the subgroup lattice.)

Example: Testing for Permutation Characters (ex-cd3c30)

In a symmetric group S_n every character is a virtual permutation character, but not in the quaternion group Q_8.

```
> C:=CharacterTable(Group("S5"));
> [IsPermutationCharacter(c): c in C];
[ true, true, true, true, true, true, true ]
> C:=CharacterTable(Group("Q8"));
> [IsPermutationCharacter(c): c in C];
[ true, true, true, true, false ]
```

100.4.3 Accessing Class Functions

In this subsection T is a character table, and x is any class function. A character table is an enumerated sequence of characters that has a special print function attached. In particular, its entries can be accessed with the ordinary sequence indexing operations.

`T[i]: TabChtr, RngIntElt -> AlgChtrElt`

Given the table T of ordinary characters of G, return the i-th character of G, where i is an integer in the range $[1...k]$.

`T[i][j]: TabChtr, RngIntElt, RngIntElt -> FldCycElt`

The value of the i-th irreducible character (from the character table T) on the j-th conjugacy class of G.

`# T: SeqEnum -> RngIntElt`

Given a character table T (or any sequence of characters), return the number of entries.

`x(g): AlgChtrElt, GrpElt -> FldCycElt`

`g @ x: GrpElt, AlgChtrElt -> FldCycElt`

The value of the class function x on the element g of G.

`x[i]: AlgChtrElt, RngIntElt -> FldCycElt`

The value of the class function x on the i-th conjugacy class of G.

`# x: AlgChtrElt -> RngIntElt`

Given a class function x on G return its length (which equals the number of conjugacy classes of the group G).

100.4.4 Conjugation of Class Functions

`x ^ g: AlgChtrElt, GrpElt -> AlgChtrElt`

Given a class function x on a normal subgroup N of the group G, and an element g of G, construct the conjugate class function x^g of x which is defined as follows: $x^g(n) = x(g^{-1}ng)$, for all n in N.

`x ^ H: AlgChtrElt, Grp -> { AlgChtrElt }`

Given a class function x on a normal subgroup N of the group G, and a subgroup H of G, construct the sequence of conjugates of x under the action of the subgroup H. The action of an element of H on x is that defined for x^g.

`GaloisConjugate(x, j): AlgChtrElt, RngIntElt -> AlgChtrElt`

Let $\mathbb{Q}(x)$ be the subfield of $\mathbb{Q}(\zeta_m)$ generated by $\mathbb{Q}$ and the values of the G-character x. This function returns the Galois conjugate x^j of x under the action of the element of the Galois group $\mathrm{Gal}(\mathbb{Q}(x)/\mathbb{Q})$ determined by the integer j. The integer j must be coprime to m.

`GaloisOrbit(x): AlgChtrElt -> { AlgChtrElt }`

Let $\mathbb{Q}(x)$ be the subfield of $\mathbb{Q}(\zeta_m)$ generated by $\mathbb{Q}$ and the values of the G-character x. This function returns the sequence of Galois conjugates of x under the action of the Galois group $\mathrm{Gal}(\mathbb{Q}(x)/\mathbb{Q})$.

`IsConjugate(x, y): AlgChtrElt, AlgChtrElt -> BoolElt, RngIntElt`

Determine if x and y are galois conjugate characters. Return `false` if not. If so, return `true`, and j such that `GaloisConjugate(x, j)` is y.

`ClassPowerCharacter(x, j): AlgChtrElt, RngIntElt -> AlgChtrElt`

Given a class function x on the group G and a positive integer j, construct the class function x^j which is defined as follows: $x^j(g) = x(g^j)$.

100.4.5 Functions Returning a Scalar

`Degree(x): AlgChtrElt -> RngIntElt`

The degree of the class function x, i.e. the value of x on the identity element of G.

`InnerProduct(x, y): AlgChtrElt, AlgChtrElt -> FldCycElt`

The inner product of the class functions x and y, where x and y are class functions belonging to the same character ring.

`Order(x): AlgChtrElt -> RngIntElt`

Given a linear character of the group G, determine the order of x as an element of the group of linear characters of G.

`Norm(x): AlgChtrElt -> FldCycElt`

Norm of the class function x (which is the inner product with itself).

`Schur(x, k): AlgChtrElt, RngIntElt -> FldCycElt`

`Indicator(x): AlgChtrElt -> FldCycElt`

Given class function x and a positive integer k, return the generalised Frobenius–Schur indicator which is defined as follows: Suppose g is some element of G, and set $T_k(g) = |\{h \in G | h^k = g\}|$. The value of `Schur(x, k)` is the coefficient a_x in the expression $T_k = \sum_{x \in \mathrm{Irr}(G)} a_x x$.

The call `Indicator(x)` is equivalent to `Schur(x,2)`.

`StructureConstant(G, i, j, k): Grp, RngIntElt, RngIntElt, RngIntElt -> RngIntElt`

The structure constant $a_{i,j,k}$ for the centre of the group algebra of the group G. If K_i is the formal sum of the elements of the i-th conjugacy class, $a_{i,j,k}$ is defined by the equation $K_i * K_j = \sum_k a_{i,j,k} * K_k$.

100.5 The Schur Index

Magma incorporates functions for computing the Schur index of an ordinary irreducible character over various number fields and local fields. The routines below are all based on the function `SchurIndices(x)`, which computes the Schur Indices of the given character over all the completions of the rationals.

The algorithm is based on calculations with characters, groups, and fields, and does not compute representations. The character table methods of §2 and §3 of [Fei83] are used as much as possible to avoid calculations in the group.

The algorithm was devised by Gabi Nebe and Bill Unger, with code written by Bill Unger. The extension to compute a Schur index over a number field was written by Claus Fieker.

`SchurIndex(x): AlgChtrElt -> RngIntElt`

`SchurIndex(x, Q): AlgChtrElt, FldRat -> RngIntElt`

`SchurIndex(x, F): AlgChtrElt, FldAlg -> RngIntElt`

The Schur index of the character x over the given field. When no field is given, the Schur index over the rationals is returned. The character x must be a complex irreducible character. The field F must be an absolute number field.

`SchurIndices(x): AlgChtrElt -> SeqEnum`

`SchurIndices(x, Q): AlgChtrElt, FldRat -> SeqEnum`

`SchurIndices(x, F): AlgChtrElt, FldAlg -> SeqEnum`

`SchurIndices(C, s, F): FldAlg, SeqEnum, FldAlg -> SeqEnum`

Compute the Schur indices of the character x over the completions of the given field. The character x must be a complex irreducible character. The field F must be an absolute number field. When no field is specified the rational field is assumed. The last form takes the character field, C, and the output from `SchurIndices(x)`, s, as well as a number field. This is sufficient to compute the Schur indices over the number field without repeating group and character computations when a number of fields are being considered for one character.

The return value is a sequence of pairs. Each pair gives a completion at which the Schur index is not 1, followed by the Schur index over the complete field. For the rational field, a completion is specified by an integer. The integer zero specifies the archimedean completion (the real numbers), while a prime p specifies the p-adic field $\mathbb{Q}_p$. When a number field is given, the completions are specified by a place of the field, an object of type `PlcNumElt`.

If the character has Schur index 1 over the given field the return value will be an empty sequence. Otherwise, the Schur index over the given field is the least common multiple of the second entries of the tuples returned.

Example: Computing Schur Indices `(ex-cbd955)`

We first look at the faithful irreducible character of the Dihedral group of order 8. It has Schur index 1.

```
> T := CharacterTable(SmallGroup(8, 3));
> T[5];
( 2, -2, 0, 0, 0 )
> SchurIndex(T[5]);
1
> SchurIndices(T[5]);
[]
```

The corresponding character of the quaternion group of order 8 has non-trivial Schur index.

```
> T := CharacterTable(SmallGroup(8, 4));
> T[5];
( 2, -2, 0, 0, 0 )
> SchurIndex(T[5]);
2
> SchurIndices(T[5]);
[ <0, 2>, <2, 2> ]
```

The Schur index is 2 over the real numbers and Q_2. For all odd primes p, the Schur index over Q_p is 1. We look at the Schur index of this character over some number fields. First we look at some cyclotomic fields.

```
> [SchurIndex(T[5], CyclotomicField(n)):n in [3..20]];
[ 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1 ]
> SchurIndices(T[5], CyclotomicField(7));
[ <Place at Prime Ideal
Two element generators:
  [2, 0, 0, 0, 0, 0]
  [1, 1, 0, 1, 0, 0], 2>, <Place at Prime Ideal
Two element generators:
  [2, 0, 0, 0, 0, 0]
  [1, 0, 1, 1, 0, 0], 2> ]
```

The cyclotomic field of order 7 gives Schur index 2. An archimedean completion of this field is necessarily the field of complex numbers, hence no infinite places give Schur index greater than 1. There are now two 2-adic completions which give Schur index 2.

```
> P<t> := PolynomialRing(Rationals());
> F := ext<Rationals()|t^3-2>;
> SchurIndex(T[5], F);
2
> SchurIndices(T[5], F);
[ <1st place at infinity, 2>, <Place at Prime Ideal
Two element generators:
  [2, 0, 0]
  [0, 1, 0], 2> ]
```

For the non-normal field F, one archimedean completion is real, the other complex. Thus the real field features in the output of `SchurIndices`, along with the 2-adic completion.

Example: Constructing a Character of Schur Index 6 (ex-ded9d7)

We will use a general construction for a character with given Schur index over the rationals to construct a character with Schur index 6. Given an integer $n \geq 1$, we select a prime p such that $p = kn + 1$ where k and n are coprime. We take an integer a such that a has order n modulo p. We then consider the metacyclic group

$$G = \langle < x, y | x^{n^2}, y^p, y^x = y^a \rangle.$$

The order of G is n^2p. The subgroup of G generated by x^n and y is cyclic, normal, and self-centralizing in G with order np. If λ is any faithful linear character of this subgroup, then λ^G is an irreducible character of G with Schur index n over the rational field. The correctness of this construction is proved in Lemma 3 of [Tur01].

The group G is constructed in two stages. The first stage is to construct G as a finitely presented group using the procedure described above. Then the (soluble) fp-group is converted to a group defined by a pc-presentation for further computation. We take $n = 6$, $p = 7$, and $a = 3$.

```
> G1 := Group<x, y|x^36, y^7, y^x = y^3>;
> G, f := SolubleQuotient(G1, 36*7);
> x := f(G1.1); y := f(G1.2);
> C := sub<G|x^6,y>;
> IsCyclic(C);
true
> IsNormal(G, C);
true
> Centralizer(G,C) eq C;
true
> exists(l){l:l in LinearCharacters(C)|IsFaithful(l)};
true;
> c := Induction(l, G);
> IsIrreducible(c);
true
> Degree(c);
6
> CharacterField(c);
Cyclotomic Field of order 3 and degree 2 in sparse
representation
> SchurIndex(c);
6
```

The construction of the previous example is used in the following two intrinsics.

SchurIndexGroup(n: parameters): RngIntElt -> GrpPC

Prime: RngIntElt	Default:

Return a group having a faithful character with Schur index n over the rational field. The construction used is as in the previous example. The parameter `Prime` may be used to supply the prime p. (The necessary conditions on p are not checked. If these conditions are not met, an error is possible.) If `Prime` is not set, then the least prime meeting the conditions is used.

CharacterWithSchurIndex(n: parameters): RngIntElt -> AlgChtrElt. GrpPC

Prime: RngIntElt	Default:

Return a character with Schur index n over the rational field. The construction used is as in the previous example. The second return value is the group of the character, equal to `SchurIndexGroup(n)`. The parameter `Prime` is as for `SchurIndexGroup`.

100.5.1 Attribute

AssertAttribute(x, "IsCharacter", b): AlgChtrElt, MonStgElt, BoolElt

Procedure that, given a class function x and a Boolean value b, stores with x the information that the value of the predicate `IsCharacter(x)` equals b.

100.6 Constructions for Characters

100.6.1 Induction, Restriction, and Lifting

`Induction(x, G): AlgChtrElt, Grp -> AlgChtrElt`

`Induction(Q, G): SeqEnum[AlgChtrElt], Grp -> SeqEnum[AlgChtrElt]`

Given a class function x on the subgroup H of the group G, construct the class function obtained by induction of x to G. Note that if x is a character of H, then `Induction(x, G)` will return a character of G.

The `Induction` command may also be used to induce a sequence of characters of a particular subgroup (such as a character table) to the given supergroup.

`LiftCharacter(c, f, G): AlgChtrElt, Map, Grp -> AlgChtrElt`

Given a class function c of the quotient group Q of the group G and the natural homomorphism $f : G \to Q$, lift c to a class function of G.

`LiftCharacters(T, f, G): [AlgChtrElt], Map, Grp -> AlgChtrElt`

Given a sequence T of class functions of the quotient group Q of the group G and the natural homomorphism $f : G \to Q$, lift T to a sequence of corresponding class functions of G. Since a character table is just a sequence of class functions which is printed in a special way, this intrinsic may also be applied to it.

`Restriction(x, H): AlgChtrElt, Grp -> AlgChtrElt`

`Restriction(Q, H): SeqEnum[AlgChtrElt], Grp -> SeqEnum[AlgChtrElt]`

Given a class function x on the group G and a subgroup H of G, construct the restriction of x to H (a class function). Note that if x is a character of G, then `Restriction(x, H)` will return a character of H.

The `Restriction` intrinsic may also be applied to a sequence of class functions.

`SubgroupFusion(G, H): Grp, Grp -> SeqEnum[RngIntElt]`

For group G with subgroup H, return a sequence giving the numbers of the conjugacy class of G containing each conjugacy class of H.

100.6.2 Symmetrization

Let λ be a partition. If we need to refer to the entries of λ, we shall do so with parenthesis, possibly also writing (λ). We write $\{\lambda\}$ for the `SymmetricCharacter` associated to λ.

Let χ be a character of a group. We then have

$$\chi^{\otimes m} = \bigoplus_{|\lambda|=m} c^{\lambda}_{(1)^m} \chi^{\{\lambda\}},$$

where $c^{\lambda}_{(1)^m}$ is a generalized Littlewood-Richardson coefficient defined as the S_λ coefficient of $S^m_{(1)}$ for the Schur symmetric functions S. For example,

$$\chi \otimes \chi \otimes \chi = \chi^{(3)} \oplus 2\chi^{(2,1)} \oplus \chi^{(1,1,1)}.$$

The character $\chi^{\{\lambda\}}$ is then called the λ-symmetrization of χ, and can be defined as

$$\chi^{\{\lambda\}}(g) = \frac{1}{n!}\Big(\sum_{\rho \in \mathrm{Sym}(n)} \{\lambda\}(\rho) \prod_{k=1}^{n} \chi(g^k)^{a_k(\rho)} \Big),$$

where $n = |\lambda|$ and $a_k(\rho)$ is the number of cycles of length k in ρ. In practice, to compute this one sums over conjugacy classes of $\mathrm{Sym}(n)$.

One can similarly define characters $[\lambda]$ or $\langle\lambda\rangle$ corresponding to orthogonal and symplectic components. Here one has (see [Lit40, §11.9], though the formula II erroneously has $\{\eta\}$ instead of $[\eta]$)

$$\{\lambda\} = \bigoplus_{\delta} \bigoplus_{\eta} c^{\lambda}_{\delta,\eta} [\eta] \quad \text{and} \quad \{\lambda\} = \bigoplus_{\beta} \bigoplus_{\eta} c^{\lambda}_{\beta,\eta} \langle\eta\rangle$$

where δ runs over partitions into even parts only, while β runs over partitions where each part appears an even number of times (conjugate partitions to the δ), and η runs over all partitions. The significant property of these is the multiplication rules (see [Lit58] (Theorems I,II)). Recalling the Littlewood-Richardson rule $\{\lambda\}\{\mu\} = \sum_\rho c^\rho_{\lambda,\mu}\{\rho\}$, we define $K^\rho_{\lambda,\mu}$ by

$$\sum_\xi \sum_{\zeta,\nu} c^\lambda_{\xi,\zeta} c^\mu_{\xi,\nu} \{\zeta\}\{\nu\} = \sum_\rho K^\rho_{\lambda,\mu}\{\rho\}$$

and we then have

$$[\lambda][\mu] = \sum_\rho K^\rho_{\lambda,\mu}[\rho] \quad \text{and} \quad \langle\lambda\rangle\langle\mu\rangle = \sum_\rho K^\rho_{\lambda,\mu}\langle\rho\rangle.$$

As noted in [New51], there are "modification rules" for $[\lambda]$ and $\langle\lambda\rangle$ when twice the number of parts of the partition exceeds the ambient dimension of the orthogonal or symplectic group. Indeed, the $[\lambda]$ or $\langle\lambda\rangle$ with no more than k parts in dimension $2k$ or $2k+1$ form a basis, with the characters from partitions with more parts being linearly dependent on them.

Given an orthogonal or respectively symplectic character χ, one can then compute the symmetrization with respect to the character $[\lambda]$ or $\langle\lambda\rangle$. See [Fra82] (or [Mur58]).

Previously the Magma implementation had hand-coded formulas for partitions of numbers no greater than 6. Furthermore, the cases of partitions of 6 had various bugs. Below we list the relevant formulas for these cases.

$$\{1\} = [1] = \langle 1\rangle$$

$$\{2\} = [2] + [] = \langle 2\rangle$$

$$\{1,1\} = [1,1] = \langle 1,1\rangle + \langle\rangle$$

$$\{3\} = [3] + [1] = \langle 3\rangle$$

$$\{2,1\} = [2,1] + [1] = \langle 2,1\rangle + \langle 1\rangle$$

$$\{1,1,1\} = [1,1,1] + [1] = \langle 1,1,1\rangle$$

$$\{4\} = [4] + [2] + [] = \langle 4\rangle$$

$$\{3,1\} = [3,1] + [2] + [1,1] = \langle 3,1\rangle + \langle 2\rangle$$

$$\{2,2\} = [2,2] + [2] + [] = \langle 2,2\rangle + \langle 1,1\rangle + \langle\rangle$$

$$\{2,1,1\} = [2,1,1] + [1,1] = \langle 2,1,1\rangle + \langle 2\rangle + \langle 1,1\rangle$$

$$\{1,1,1,1\} = [1,1,1,1] = \langle 1,1,1,1\rangle + \langle 1,1\rangle + \langle\rangle$$

$$\{5\} = [5] + [3] + [1] = \langle 5\rangle$$

$$\{4,1\} = [4,1] + [3] + [2,1] + [1] = \langle 4,1\rangle + \langle 3\rangle$$

$$\{3,2\} = [3,2] + [3] + [2,1] + [1] = \langle 3,2\rangle + \langle 2,1\rangle + \langle 1\rangle$$

$$\{3,1,1\} = [3,1,1] + [2,1] + [1,1,1] = \langle 3,1,1\rangle + \langle 3\rangle + \langle 2,1\rangle$$

$$\{2,2,1\} = [2,2,1] + [2,1] + [1] = \langle 2,2,1\rangle + \langle 2,1\rangle + \langle 1,1,1\rangle + \langle 1\rangle$$

$$\{2,1,1,1\} = [2,1,1,1] + [1,1,1] = \langle 2,1,1,1\rangle + \langle 2,1\rangle + \langle 1,1,1\rangle + \langle 1\rangle$$

$$\{1,1,1,1,1\} = [1,1,1,1,1] = \langle 1,1,1,1,1\rangle + \langle 1,1,1\rangle + \langle 1\rangle$$

$$\{6\} = [6] + [4] + [2] + [] = \langle 6\rangle$$

$$\{5,1\} = [5,1] + [4] + [3,1] + [2] + [1,1] = \langle 5,1\rangle + \langle 4\rangle$$

$$\{4,2\} = [4,2] + [4] + [3,1] + [2,2] + 2[2] + [] = \langle 4,2\rangle + \langle 3,1\rangle + \langle 2\rangle$$

$$\{4,1^2\} = [4,1,1] + [3,1] + [2,1,1] + [1,1] = \langle 4,1,1\rangle + \langle 4\rangle + \langle 3,1\rangle$$

$$\{3,3\} = [3,3] + [3,1] + [1,1] = \langle 3,3\rangle + \langle 2,2\rangle + \langle 1,1\rangle + \langle\rangle$$

$$\{3,2,1\} = [3,2,1] + [3,1] + [2^2] + [2,1^2] + [2] + [1^2] = \langle 3,2,1\rangle + \langle 3,1\rangle + \langle 2^2\rangle + \langle 2,1^2\rangle + \langle 2\rangle + \langle 1^2\rangle$$

$$\{3,1^3\} = [3,1^3] + [2,1,1] + [1^4] = \langle 3,1^3\rangle + \langle 3,1\rangle + \langle 2,1,1\rangle + \langle 2\rangle$$

$$\{2,2,2\} = [2,2,2] + [2,2] + [2] + [] = \langle 2,2,2\rangle + \langle 2,1,1\rangle + \langle 2\rangle$$

$$\{2^2,1^2\} = [2^2,1^2] + [2,1^2] + [1^2] = \langle 2^2,1^2\rangle + \langle 2^2\rangle + \langle 2,1^2\rangle + \langle 1^4\rangle + 2\langle 1^2\rangle + \langle\rangle$$

$$\{2,1^4\} = [2,1^4] + [1^4] = \langle 2,1^4\rangle + \langle 2,1,1\rangle + \langle 1^4\rangle + \langle 2\rangle + \langle 1,1\rangle$$

$$\{1^6\} = [1^6] = \langle 1^6\rangle + \langle 1^4\rangle + \langle 1^2\rangle + \langle\rangle$$

`Symmetrization(chi, p): AlgChtrElt, [ RngIntElt ] -> AlgChtrElt`

Given a character χ and partition p return the symmetrized character $\chi^{\{p\}}$. The power map for the parent of χ must be available.

`OrthogonalSymmetrization(chi, p): AlgChtrElt, [ RngIntElt ] -> AlgChtrElt`

Given an orthogonal character χ and a partition p return the character $\chi^{[p]}$. The power map for the parent of χ must be available.

`SymplecticSymmetrization(chi, p): AlgChtrElt, [ RngIntElt ] -> AlgChtrElt`

Given a symplectic character χ and a partition p return the character $\chi^{\langle p\rangle}$. The power map for the parent of χ must be available.

Example: Tensor Product Rule for Symmetrizations (ex-870f58)

This example verifies the tensor product rule for some characters and partitions.

```
> S := SFA(Integers());
> function tensprod(chi,m)
>   F := S.1^m;
>   return &+[Coefficient(F,p)*Symmetrization(chi,p) : p in Support(F)];
> end function;
> CT := CharacterTable(SU(2,7));
> for chi in CT, m in [1..6] do
>   assert tensprod(chi,m) eq chi^m;
> end for;
```

Example: Symplectic and Orthogonal Symmetrizations (ex-04434a)

Here we compute the orthogonal and symplectic symmetrizations of some characters. Note that these need not be characters (but virtual ones) in cases where the degree of the character is smaller than twice the number of parts in the partition. This is consistent with the above-mentioned modification rules.

```
> CT := CharacterTable(SU(2,5));
> chi := rep{chi: chi in CT | Degree(chi) eq 4 and IsOrthogonalCharacter(chi)};
( 4, 4, 1, 0, -1, -1, 1, -1, -1 )
> [OrthogonalSymmetrization(chi,p) : p in Partitions(6)];
[
    ( 49, 49, 1, 1, -1, -1, 1, -1, -1 ),
    ( 70, 70, -2, -2, 0, 0, -2, 0, 0 ),
    ( 42, 42, 0, 2, 2, 2, 0, 2, 2 ),
    ( 25, 25, 1, 1, 0, 0, 1, 0, 0 ),
    ( 14, 14, 2, -2, -1, -1, 2, -1, -1 ),
    ( 0, 0, 0, 0, 0, 0, 0, 0, 0 ),
```

(continues on next page)

(continued from previous page)

```
    ( 0, 0, 0, 0, 0, 0, 0, 0, 0 ),
    ( -10, -10, 2, -2, 0, 0, 2, 0, 0 ),
    ( -9, -9, 0, -1, 1, 1, 0, 1, 1 ),
    ( -1, -1, -1, -1, -1, -1, -1, -1, -1 ),
    ( 0, 0, 0, 0, 0, 0, 0, 0, 0 )
]
> CT := CharacterTable(SU(2,7));
> chi := CT[7]; chi;
( 6, -6, 0, 0, 0, -1, -1, Sqrt(2), -Sqrt(2), 1, 1 )
> [SymplecticSymmetrization(chi,p) : p in Partitions(6)];
[
    ( 462, 462, 3, -10, 3, 0, 0, -2, -2, 0, 0 ),
    ( 924, 924, -3, 4, -3, 0, 0, 0, 0, 0, 0 ),
    ( 924, 924, 0, -12, 0, 0, 0, 0, 0, 0, 0 ),
    ( 525, 525, 3, 5, 3, 0, 0, 1, 1, 0, 0 ),
    ( 385, 385, 4, 9, 4, 0, 0, 1, 1, 0, 0 ),
    ( 512, 512, -4, 0, -4, 1, 1, 0, 0, 1, 1 ),
    ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ),
    ( 84, 84, 3, -4, 3, 0, 0, 0, 0, 0, 0 ),
    ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ),
    ( -70, -70, -1, 2, -1, 0, 0, 2, 2, 0, 0 ),
    ( -14, -14, 1, -2, 1, 0, 0, 0, 0, 0, 0 )
]
```

Finally, there are intrinsics to determine whether a character is orthogonal or symplectic. For irreducible characters, this can be determined simply by the Schur indicator. In general, the fact that $SO_{2n} \cap Sp_{2n} = U_n$ implies that for all χ the character $\chi \oplus \bar{\chi}$ is both orthogonal and symplectic (orthogonal meaning that it preserves a full rank symmetric form, the same with alternating for symplectic).

When ψ is not real, then it is neither orthogonal nor symplectic. We decompose $\psi = \sum_\chi a_\chi \chi$ into irreducibles, and for ψ real this implies $a_\chi = a_{\bar{\chi}}$ for nonreal χ. For real χ we can further reduce the a_χ modulo 2. For ψ to be declared orthogonal, we then require every remaining constituent to be itself orthogonal, and similarly for symplectic.

`IsOrthogonalCharacter(chi): AlgChtrElt -> BoolElt`

`IsSymplecticCharacter(chi): AlgChtrElt -> BoolElt`

Given a character of a finite group, determine whether it is orthogonal or symplectic.

100.6.3 Permutation Character

`PermutationCharacter(G): GrpPerm -> AlgChtrElt`

Given group G represented as a permutation group, construct the character of G afforded by the defining permutation representation of G.

`PermutationCharacter(G, H): Grp, Grp -> AlgChtrElt`

Given a group G and some subgroup H of G, construct the character of G afforded by the permutation representation of G given by the action of G on the right cosets of H in G.

100.6.4 Composition and Decomposition

`Composition(T, q): [ AlgChtrElt ], [RngElt] -> AlgChtrElt`

Given a sequence or table of characters T for the group G and a sequence q of k elements of $\mathbb{Q}(\zeta_m)$ (possibly $\mathbb{Q}$), create the class function $q_1 * T_1 + \cdots + q_k * T_k$, where T_i is the i-th character in T.

`Decomposition(T, y): [AlgChtrElt], AlgChtrElt -> [ FldCycElt ], AlgChtrElt`

Given a sequence or table of class functions T for G of length l and a class function y on G, attempt to express y as a linear combination of the elements of T.

The function returns two values: a sequence $q = [q_1, \ldots, q_l]$ of cyclotomic field elements and a class function z. For $1 \leq i \leq l$, the i-th term of q is defined to be the ratio of inner products $(y, T_i)/(T_i, T_i)$, where T_i is the i-th entry of T. The sequence q determines a class function $x = q_1 \cdot T_1 + \cdots + q_l \cdot T_l$ which will equal y if T is the complete table of irreducible characters. The difference $z = y - x$ is the second return value. If the entries in T are mutually orthogonal, then z is the zero class function if and only if y is a linear combination of the T_i.

100.6.5 Finding Irreducibles

A common approach to finding the irreducible characters of a group is to start with an irreducible character and generate new characters by applying `Symmetrization`, or the orthogonal or symplectic variants. Then, by examining norms and inner products, it is often possible to identify irreducible characters or at least characters with smaller norms. There are two Magma intrinsics available to help with this task.

`RemoveIrreducibles(I, C): [ AlgChtrElt ], [ AlgChtrElt ] -> [ AlgChtrElt ], [ AlgChtrElt ]`

Remove occurrences of the irreducible characters in the sequence I from the characters in the sequence C and look for characters of norm 1 among the reduced characters. Return a sequence of new irreducibles found and the sequence of reduced characters.

`ReduceCharacters(I, C): [ AlgChtrElt ], [ AlgChtrElt ] -> [ AlgChtrElt ], [ AlgChtrElt ]`

Make the norms of the characters in the sequence C smaller by computing the differences of appropriate pairs. Return a sequence of new irreducibles found and a sequence of reduced characters.

Example: Character Table of A5 from Subgroups `(ex-51853a)`

We conclude this section with an example, showing how the above functions can be used to construct the character table for A_5 (compare Isaacs, p64), using only characters on subgroups.

```
> A := AlternatingGroup(GrpPerm, 5);
> R := CharacterRing(A);
```

The first character will be the principal character

```
> T1 := R ! 1;
> T1;
( 1, 1, 1, 1, 1 )
```

Next construct the permutation character

```
> pc := PermutationCharacter(A);
> T2 := pc - T1;
> InnerProduct(pc, T1), InnerProduct(T2, T2);
1 1
> T2;
( 4, 0, 1, -1, -1 )
```

It follows that $pc - T1$ is an irreducible character.

```
> B := Stabilizer(A, 5);
> r := RootOfUnity(3, CyclotomicField(3));
> S := CharacterRing(B);
> lambda := S ! [1, 1, r, r^2 ];
> IsLinear(lambda);
true
```

This defines a linear character on a subgroup of index 5 in A.

```
> T3 := Induction(lambda, A);
> InnerProduct(T3, T3);
1
> T3;
( 5, 1, -1, 0, 0 )
```

Finally we use characters on the cyclic subgroup of order 5:

```
> K := sub<A |  (1,2,3,4,5) >;
> Y := CharacterTable(K);
> Y;

Character Table of Group K
--------------------------


-------------------------------
Class |   1    2    3    4    5
Size  |   1    1    1    1    1
Order |   1    5    5    5    5
-------------------------------
p  =  5   1    1    1    1    1
-------------------------------
X.1   +   1    1    1    1    1
X.2   0   1   Z1 Z1#2 Z1#3 Z1#4
X.3   0   1 Z1#2 Z1#4   Z1 Z1#3
X.4   0   1 Z1#3   Z1 Z1#4 Z1#2
X.5   0   1 Z1#4 Z1#3 Z1#2   Z1


Explanation of Symbols:
-----------------------

# denotes algebraic conjugation, that is,
# k indicates replacing the root of unity w by w^k

Z1     = -1 - zeta_5 - zeta_5^2 - zeta_5^3

> mu := Induction(Y[2], A);
```

We subtract what we already know from mu and get a new irreducible. We use decomposition with respect to a sequence.

```
> _, T4 := Decomposition([T1, T2, T3], mu);
> InnerProduct(T4, T4);
1
> T4;
( 3, -1, 0, (1 + zeta_5^2 + zeta_5^3), (-zeta_5^2 - zeta_5^3) )
> T5 := GaloisConjugate(T4, 2);
> T5;
( 3, -1, 0, (-zeta_5^2 - zeta_5^3), (1 + zeta_5^2 + zeta_5^3) )
```

Compare this to the standard character table:

```
> CharacterTable(A);

Character Table of Group A
--------------------------
```

(continues on next page)

(continued from previous page)

```
--------------------------
Class |   1  2  3    4    5
Size  |   1 15 20   12   12
Order |   1  2  3    5    5
--------------------------
p  =  2   1  1  3    5    4
p  =  3   1  2  1    5    4
p  =  5   1  2  3    1    1
--------------------------
X.1   +   1  1  1    1    1
X.2   +   3 -1  0   Z1 Z1#2
X.3   +   3 -1  0 Z1#2   Z1
X.4   +   4  0  1   -1   -1
X.5   +   5  1 -1    0    0


Explanation of Symbols:
-----------------------

#  denotes algebraic conjugation, that is,
# k indicates replacing the root of unity w by w^k

Z1     =(1 + zeta_5^2 + zeta_5^3)
```

We now create a character ring, corresponding to the character ring for A_5, but without a group, using the data displayed above. We then assert the irreducible of degree 4, and produce another irreducible via asserting the power map for R and looking at a symmetrized power.

```
> Q := [<1,1>, <2,15>, <3,20>, <5, 12>, <5,12>];
> R := CharacterRing(Q);
> x := R![4,0,1,-1,-1];
> x`IsIrreducible := true;
> x;
( 4, 0, 1, -1, -1 )
> #KnownIrreducibles(R);
2
> pm := [[1],[2,1],[3,3,1],[4,5,5,4,1],[5,4,4,5,1]];
> R`PowerMap := pm;
> y := Symmetrization(x, [2]);
> InnerProduct(y, R!1);
1
> y := y - R!1;
> InnerProduct(y, x);
1
> y := y - x;
> y;
( 5, 1, -1, 0, 0 )
> Norm(y);
1
> y`IsIrreducible := true;
> #KnownIrreducibles(R);
3
```

100.7 Brauer Characters

Magma provides some tools for the construction of individual Brauer characters and has an intrinsic that can compute all of the p-modular characters. These intrinsics are described in this section.

A Brauer character modulo p in Magma is represented as a class function (that is, element of a character ring) which is zero for p-singular group elements. In this format the standard character operations of addition, multiplication, induction, and restriction all apply directly to Brauer characters as they do to other class functions.

It should be emphasised that for a Brauer character x, the operation `x[i]` returns the value of x on the ith conjugacy class of the group, and that a printed Brauer character omits values on the p-singular classes.

100.7.1 Structure Creation

`CharacterRing(G, p): Grp, RngIntElt -> AlgChtr`

Given a finite group G, create the ring of complex-valued class functions on p-regular elements of G. This function will trigger the computation of the conjugacy classes of G if these are not yet known.

`CharacterRing(R, p): AlgChtr, RngIntElt -> AlgChtr`

Given a character ring, create the ring of complex-valued class functions on p-regular elements of the domain of R.

`BrauerCharacteristic(R): AlgChtr -> RngIntElt`

The prime associated with the Brauer characters in R, returns zero for the ordinary character ring.

100.7.2 Element Creation

`R ! [ a`$_1$`, ..., a`$_k$` ]: AlgChtr, SeqEnum -> AlgChtrElt`

Given the ring of class functions R of a finite group G with k p-regular conjugacy classes and k elements a_i contained in some common cyclotomic field, create a class function on G for which the value on the i-th class is equal to the i-th term a_i.

`Id(R): AlgChtr -> AlgChtrElt`

`Identity(R): AlgChtr -> AlgChtrElt`

`One(R): AlgChtr -> AlgChtrElt`

Given the ring of class functions R, create the principal character (which takes on the value 1 on every p-regular element of G).

`Zero(R): AlgChtr -> AlgChtrElt`

Given a ring of class functions R create its zero element (which is the class function that takes on the value 0 on every element of the group).

`BrauerCharacter(x, p): AlgChtrElt, RngIntElt -> AlgChtrElt`

The Brauer character modulo the prime p obtained by setting the value of x on p-singular elements to be zero.

`BrauerCharacter(M): ModGrp -> AlgChtrElt`

The Brauer character of the G-module M over $GF(q)$.

The choice of lifting from finite field elements to complex values uses a Conway polynomial for the finite field of order q, and lifts a root of the polynomial to $\exp(2\pi i/(q-1))$.

`BrauerCharacterTable(G, p): Chtr -> SeqEnum`

Construct the table of irreducible mod p Brauer characters for the group G. For soluble groups this is deduced from the ordinary character table. For insoluble groups the absolutely irreducible mod p representations are constructed and their characters computed.

100.7.3 Arithmetic

The basic arithmetic operations for ordinary characters are also available for Brauer characters.

In the list of arithmetic operations below x and y denote Brauer characters in the same ring, and a denotes a scalar, which is any element coercible into a cyclotomic field. Also, j denotes an integer.

`+ y: AlgChtrElt -> AlgChtrElt`

`- y: AlgChtrElt -> AlgChtrElt`

`x + y: AlgChtrElt, AlgChtrElt -> AlgChtrElt`

`x - y: AlgChtrElt, AlgChtrElt -> AlgChtrElt`

`x * y: AlgChtrElt, AlgChtrElt -> AlgChtrElt`

`a * x: FldCycElt, AlgChtrElt -> AlgChtrElt`

`x ^ j: AlgChtrElt, RngIntElt -> AlgChtrElt`

100.7.4 Induction, Restriction, and Lifting

`Induction(x, G): AlgChtrElt, Grp -> AlgChtrElt`

`Induction(Q, G): SeqEnum[AlgChtrElt], Grp -> SeqEnum[AlgChtrElt]`

Given a Brauer character x on the subgroup H of the group G, construct the Brauer character obtained by induction of x to G.

The `Induction` command may also be used to induce a sequence of Brauer characters of a particular subgroup to the given supergroup.

`LiftCharacter(c, f, G): AlgChtrElt, Map, Grp -> AlgChtrElt`

Given Brauer character c of the quotient group Q of the group G and the natural homomorphism $f : G \to Q$, lift c to a class function of G.

`LiftCharacters(T, f, G): [AlgChtrElt], Map, Grp -> AlgChtrElt`

Given a sequence T of Brauer characters of the quotient group Q of the group G and the natural homomorphism $f : G \to Q$, lift T to a sequence of corresponding class functions of G.

`Restriction(x, H): AlgChtrElt, Grp -> AlgChtrElt`

`Restriction(Q, H): SeqEnum[AlgChtrElt], Grp -> SeqEnum[AlgChtrElt]`

Given a Brauer character x on the group G and a subgroup H of G, construct the restriction of x to H (a Brauer character).

The `Restriction` intrinsic may also be applied to a sequence of Brauer characters.

`Blocks(T, p): SeqEnum[AlgChtrElt], RngIntElt -> SeqEnum, SeqEnum`

When T is the full ordinary character table of a group, return the partition of T into p-blocks, where p is a given prime. The partition is returned as a sequence of sets of integers which give the blocks by the positions of the characters in T. The second return value is the corresponding sequence of defects of the blocks. The blocks are ordered first by decreasing defect, second by first character in the block.

`DefectGroup(T, b, p): SeqEnum[AlgChtrElt], SetEnum[RngIntElt], RngIntElt -> Grp`

Returns a subgroup of the group associated with the character table T which is a defect group for the p-modular block b.

`DefectGroup(x, p): AlgChtrElt, RngIntElt -> Grp`

Returns a subgroup of the group associated with the ordinary character x which is a defect group for the p-modular block containing x. Note that x must be an ordinary character of the group.

Example: 3-Modular Characters of the Higman-Sims Group (ex-ae1049)

We give an example of the use of these Brauer character functions. We consider the 3-modular characters of the Higman-Sims simple group.

```
> load hs176;
> T := CharacterTable(G);
> Blocks(T,3);
[
  { 1, 2, 5, 10, 18, 19, 21, 23, 24 },
  { 3, 4, 6, 7, 11, 12, 14, 15, 20 },
  { 8, 13, 16 },
  { 9 },
  { 17 },
  { 22 }
]
[ 2, 2, 1, 0, 0, 0 ]
```

The characters $T[8], T[13], T[16]$ are the ordinary irreducible characters in a 3-block of defect one. In such a small block the two ordinary irreducibles of minimal degree will restrict to modular irreducibles.

```
> [Degree(T[i]): i in [8, 13, 16]];
[ 231, 825, 1056 ]
> BrauerCharacter(T[8], 3);
( 231, 7, -9, 15, -1, -1, 6, 1, 1, 0, -1, -1, -1, 2, 1, 0, 0, 0, 0 )
> BrauerCharacter(T[13], 3);
( 825, 25, 9, -15, 1, 1, 0, -5, 0, -1, 1, 1, 1, 0, -1, 0, 0, 0, 0 )
> $1 + $2 eq BrauerCharacter(T[16], 3);
true
```

The projective indecomposable characters (as ordinary characters) corresponding to these Brauer irreducible characters are as follows.

```
> T[8] + T[16];
( 1287, 39, -9, 0, 15, -1, -1, 12, -3, 2, 0, 0, -1, -1, -1,
-1, 4, 1, 0, 0, 0, 0, 0, 0 )
> T[13] + T[16];
( 1881, 57, 9, 0, -15, 1, 1, 6, -9, 1, 0, 0, -2, 1, 1, 1, 2,
-1, 0, 0, 0, 0, 0, 0 )
> BrauerCharacter(T[8] + T[16],3);
( 1287, 39, -9, 15, -1, -1, 12, -3, 2, -1, -1, -1, -1, 4, 1, 0, 0, 0, 0 )
> chi := BrauerCharacter(T[5],3)^2;
> chi;
( 23716, 100, 100, 4, 36, 4, 16, 16, 1, 0, 0, 0, 0, 0, 0, 0, 0, 4, 4 )
> chi_p := BlockProjections(G, chi);
> chi_p;
[
  ( 13708, -20, -8, -20, 28, 0, 8, -2, -2, 2, 2, -4, 2, 0, 2, 2, 2, 0, 0 ),
  ( 3969, 33, 45, -15, 1, 5, -6, 9, 4, 0, -1, 1, -1, -2, 5, -2, -2, 0, 0 ),
  ( 1881, 57, 9, -15, 1, 1, 6, -9, 1, -2, 1, 1, 1, 2, -1, 0, 0, 0, 0 ),
  ( 1386, 42, 18, 42, 10, 2, -14, 6, -4, 0, -2, 2, -2, 2, -2, 0, 0, 2, 2 ),
  ( 2772, -12, 36, 12, -4, -4, 22, 12, 2, 0, 0, 0, 0, -2, -4, 0, 0, 2, 2 ),
  ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 )
]
> chi_p[3] eq BrauerCharacter(T[8],3) + 2*BrauerCharacter(T[13],3);
true
```

100.8 Database of Character Tables

Magma contains character tables of many of the groups available as an ATLASGroup(N): MonStgElt → GrpAtlas. These tables have been computed using the *CharacterTable* function, and are examples of character tables without a group. Characters are returned with parent character ring having a power map assigned. and with Frobenius-Schur indicator of each character pre-computed.

The available character tables are named using the name of the corresponding ATLASGroup(N): MonStgElt → GrpAtlas.

This facility is currently under expansion, and more tables and facilities will be available with future releases.

CharacterTable(N): MonStgElt -> SeqEnum[AlgChtrElt]

CharacterTable(A): GrpAtlas -> SeqEnum[AlgChtrElt]

Return the character table of the group A, or the group named by the string N, as a sequence of characters without a group. If there is no stored character table corresponding to the name given, a runtime error will result.

HasCharacterTable(A): GrpAtlas -> BoolElt

Return **true** if the character table for the ATLAS group A is in the Magma character table database.

CharacterTableNames() -> SetIndx[MonStElt]

Return the set of strings that are valid character table names in the database.

Example: Character Table Data for E6(2) (ex-7566cf)

We use the CharacterTable function to examine some aspects of the simple group $E_6(2)$. We start by showing that $E_6(2)$ has elements x of order 2 and y of order 3 such that xy has order 7.

```
> "E62" in CharacterTableNames();
true
> X := CharacterTable("E62");
> #X;
180
> R := Universe(X);
> assigned R`Group;
false
> cd := ClassesData(R);
> c2 := [i: i in [1..#cd] | cd[i,1] eq 2];
> c3 := [i: i in [1..#cd] | cd[i,1] eq 3];
> c7 := [i: i in [1..#cd] | cd[i,1] eq 7];
> c2, c3, c7;
[ 2, 3, 4 ]
[ 5, 6, 7 ]
[ 29, 30, 31, 32 ]
> exists(t){<i,j,k>: i in c2, j in c3, k in c7 |
>                  not IsZero(StructureConstant(R, i, j, k))};
true
> t;
<3, 6, 29>
```

This shows that elements of class 3 (element order 2) when multiplied by a suitable element of class 6 (element order 3), give an element in class 29 (element order 7).

In the proof of Ore's conjecture by Liebeck, O'Brien, Shalev & Tiep, $E_6(2)$ was a case where the conjecture was checked directly using Frobenius' character table criterion for an element to be a commutator. We can easily repeat this calculation. The function test computes a value for a given conjugacy class such that this value is non-zero if and only if the class consists of commutators.

```
> test := func< k | &+[ x[k]/x[1] : x in X ] >;
> forall{ k : k in [1..#cd] | not IsZero(test(k)) };
true
```

This shows that all conjugacy classes of $E_6(2)$ consist of commutators, verifying the conjecture for this group.

We now move on to finding characters of rational representations of $E_6(2)$. It can be shown, using no more than the character table data, that the second and third characters in this table are afforded by rational representations of the group. Given this, we can find further characters afforded by rational representations in the character table. We next verify enough conditions to show that the second and third characters are indeed afforded by rational representations, and then use this to find further characters.

```
> Degree(X[2]), Degree(X[3]);
2482 137020
> Indicator(X[2]), Indicator(X[3]);
1 1
> DegreeOfCharacterField(X[2]);
1
> DegreeOfCharacterField(X[3]);
1
> pm := PowerMap(R);
> is_integral := func<i|forall{j:j in [1..n] |
>  GCD(i,n) ne 1 or pm(i,j) eq i} where n := cd[i,1]>;
> [i : i in [1..#X] | X[2,i] eq 1 and is_integral(i)];
[ 52, 56, 66, 75, 82, 84, 93, 104, 105, 106, 126, 147, 150 ]
> GCD([cd[i,1]: i in $1]);
1
> [i : i in [1..#X] | X[3,i] eq 1 and is_integral(i)];
[ 62, 87, 90, 91, 93, 94, 96, 102, 105, 106, 108, 111, 112,
114, 123, 124, 126, 161, 162 ]
> GCD([cd[i,1]: i in $1]);
1
```

We first established that the two characters are both rational valued with indicators 1. They both take the value 1 on integral classes such that the orders of the elements in these classes are coprime. This is sufficient to allow us to conclude that the Schur index over the rational field of both these characters is 1, see [Fei83]. It follows that these are characters of rational representations of $E_6(2)$. We now consider products of these characters.

```
> x := X[2]*X[3];
> [ i : i in [1..#X] | InnerProduct(X[i], x) eq 1];
[ 2, 5, 8, 17, 21 ]
> x := X[2]^2;
> [ i : i in [1..#X] | InnerProduct(X[i], x) eq 1];
[ 1, 2, 3, 5, 6, 8 ]
> x := X[3]^2;
> [ i : i in [1..#X] | InnerProduct(X[i], x) eq 1];
[ 1, 6, 7, 12, 20, 30, 33, 36, 40 ]
```

This shows that the characters numbered $2, 3, 5, 6, 7, 8, 12, 17, 20, 21, 30, 33, 36, 40$ all have Schur index 1 over the rational numbers. In fact, all these characters are rational valued, so they are characters of rational representations. Further products give more information.

```
> J := [2,3,5,6,7,8,12,17,20,21,30,33,36,40];
> forall{i: i in J | DegreeOfCharacterField(X[i]) eq 1};
true
```

(continues on next page)

(continued from previous page)

```
> x := X[5]^2;
> [ i : i in [1..#X] | InnerProduct(X[i], x) eq 1];
[ 1, 2, 7, 12, 25, 29, 36, 49, 54, 55, 82, 96, 97, 100,
102, 151, 152, 154, 178, 180 ]
> DegreeOfCharacterField(X[151]);
2
```

The characters found above all have Schur index 1. We see that this includes some irrational characters.

As the power map is stored with the character table, we can use symmetrized powers to improve the above results.

```
> assigned R`PowerMap;
true
> x := Symmetrization(X[5], [1,1]);
> [ i : i in [1..#X] | InnerProduct(X[i], x) eq 1];
[ 13, 25, 30, 35, 36, 37, 52, 54, 66, 67, 70, 77, 82, 97,
98, 101, 131, 133, 134, 148, 149, 150, 151, 152, 178 ]
> x := Symmetrization(X[5], [2]);
> [ i : i in [1..#X] | InnerProduct(X[i], x) eq 1];
[ 1, 2, 5, 7, 12, 20, 29, 34, 35, 37, 49, 52, 55, 66, 67,
70, 96, 98, 100, 101, 102, 131, 134, 148, 149, 150, 154,
180 ]
```

Example: Schur Indices for the Lyons Group (ex-4c43a0)

We consider the characters of Lyons' sporadic simple group, and compute Schur indices using character table information. We will use the methods described in [Fei83], and check the assertion in §7 of this article that all characters of Lyons' group have Schur index 1. Theorems referenced here refer to this article.

In general we cannot deduce all local Schur indices from character table information, but here we are fortunate and can do so.

We begin by finding the integral classes of the group, and setting up a `test` function that, when it returns true, implies that the character has Schur index 1 over the rationals. We are using Theorem A, Theorem 2.7, and Theorem 2.15.

```
> X := CharacterTable("Ly");
> #X;
53
> Z := Integers();
> int_cl := {i : i in [1..#X] | CanChangeUniverse([x[i]: x in X], Z)};
> R := Universe(X);
> d := ClassesData(R);
> test := function(x)
>    if Indicator(x) eq -1 then return false; end if;
>    c := [i : i in int_cl | x[i] in {1,-1}];
>    return #c gt 0 and GCD([d[i,1]: i in c]) eq 1;
> end function;
> schur_1 := {i : i in [1..#X] | test(X[i]) };
> #schur_1;
28
> rat := {i:i in schur_1|DegreeOfCharacterField(X[i]) eq 1};
> #rat;
20
```

We have found 28 characters with Schur index 1. Of these, 20 are rational valued, hence are

characters of rational representation of Ly. We now consider inner products with products of our 20 rational characters in pairs, to find more characters having Schur index 1. Here we are using Theorem 2.1 and Corollary 2.4.

```
> unknown := {1..#X} diff schur_1;
> #unknown;
25
> new_1 := {i : i in unknown | exists{j : j,k in rat | j le k
> and (ip eq 1 or (Indicator(X[i]) ne 0 and IsOdd(ip)))
> where ip := Z!InnerProduct(X[i], X[j]*X[k]) } };
> schur_1 := schur_1 join new_1;
> unknown := {1..#X} diff schur_1;
> unknown;
{ 37 }
> chi := X[37];
> Degree(chi), Indicator(chi);
36887520 1
> DegreeOfCharacterField(chi);
1
> PrimeDivisors(GroupOrder(R) div 36887520);
[ 2, 3, 5, 7, 11 ]
```

We have shown that all but one character ($\chi = \chi_{37}$) have Schur index 1. The outstanding character χ has degree 36 887 520, indicator 1 (hence Schur index 1 over the real numbers), and is rational valued. Furthermore, for primes $p \notin \{2, 3, 5, 7, 11\}$ this character has p-adic Schur index $m_p(\chi) = 1$ (Theorem 2.10).

We now compute the Schur indices of χ over the remaining p-adic fields. We start with values of χ on group elements of order 11 (class 17). This may give us information on the cases $p \in \{2, 3, 5, 7\}$. We test if characters in the same p-block as χ take values on class 17 that lie in the p-adic field $\mathbb{Q}_p$. We use a test for this based on the Galois group of $\mathbb{Q}_p(\zeta_{11})$ over $\mathbb{Q}_p$. This group is cyclic, generated by the pth conjugacy operation: $\zeta_{11} \mapsto \zeta_{11}^p$.

```
> d[17,1];
11
> chi[17];
-1
> for p in [2,3,5,7] do
>    bl := Block(X, 37, p);
>    p, forall{i:i in bl|X[i,17] eq Conjugate(X[i,17],p)};
> end for;
2 false
3 true
5 true
7 false
> GroupFactoredOrder(R);
[ <2, 8>, <3, 7>, <5, 6>, <7, 1>, <11, 1>, <31, 1>,
<37, 1>, <67, 1> ]
```

We find that $m_3(\chi)$ and $m_5(\chi)$ are divisors of the character value -1 (Corollary 3.2), hence are both 1. When $p = 2, 7$ we see that Brauer characters in the p-block of χ take values outside $\mathbb{Q}_p$. Noting that the Sylow 7-subgroups of Ly are cyclic, Theorem 2.12 gives $m_7(\chi) > 1$. As real valued characters have all Schur indices 1 or 2, $m_7(\chi) = 2$.

It remains to determine $m_2(\chi)$ and $m_{11}(\chi)$. Theorem 2.15 shows that one of these has to equal 1 and the other has to equal 2. We consider group elements of order 21 (class 27) in the same manner as above.

```
> d[27,1];
21
> chi[27];
1
> for p in [2,11] do
>    bl := Block(X, 37, p);
>    p, forall{i:i in bl|X[i,27] eq Conjugate(X[i,27],p)};
> end for;
2 false
11 true
```

From this we conclude that $m_{11}(\chi) = 1$ (Corollary 3.2), and deduce $m_2(\chi) = 2$.

We have shown that the statement in §7 of [Fei83] that all characters of Lyons' group have Schur index 1 is not quite correct. The character of degree 36 887 520 has 2-adic and 7-adic Schur indices equal to 2, with its other p-adic Schur indices and its real Schur index being 1. All the other characters of Ly have Schur index 1.

100.9 Bibliography

- [Con90a] S. B. Conlon. Calculating characters of p-groups. *J. Symbolic Comp.*, 9:535–550, 1990.
- [Dix67] J. D. Dixon. High–speed computation of group characters. *Numerische Mathematik*, 10:446–450, 1967.
- [Fei83] Walter Feit. The computations of some Schur indices. *Israel J. Math.*, 46(4):274–300, 1983.
- [Fra82] J. S. Frame. Recursive computation of tensor power components. *Bayreuth. Math. Schr.*, pages 153–159, 1982.
- [Lit40] D. E. Littlewood. *The Theory of Group Characters and Matrix Representations of Groups.* Clarendon Press (Oxford), 1940.
- [Lit58] D. E. Littlewood. Products and plethysms of characters with orthogonal, symplectic and symmetric groups. *Canad. J. Math*, 10:17–32, 1958.
- [Mur58] F. D. Murnaghan. The orthogonal and symplectic groups. *Comm. Dublin Inst. Adv. Studies. Ser. A, no.*, 13:146, 1958.
- [New51] M. J. Newell. Modification rules for the orthogonal and symplectic groups. *Proc. Royal Irish Acad. Section A*, 54:153–163, 1951.
- [Sch90b] G. J. A. Schneider. Dixon's Character Table Algorithm Revisited. *J. Symbolic Computation*, 9:601–606, 1990.
- [Tur01] A. Turull. Schur indices of perfect groups. *Proc. Amer. Math. Soc.*, 130(2):367–370, 2001.
- [Ung06a] W.R. Unger. Computing the character table of a finite group. *J. Symbolic Comp.*, 41(8):847–862, 2006.

Chapter 101

Representations of Symmetric Groups

101.1 Introduction

This chapter describes functions available in Magma for computations concerning the non modular representation theory of the symmetric group and the alternating group. It is possible to compute different matrix representations, complete character tables using special routines, single irreducible characters, or the value of an irreducible character on an element of the group.

101.2 Representations of the Symmetric Group

For the symmetric group of degree n the irreducible representations can be indexed by partitions of weight n. For more information on partitions see Section Partitions.

101.2.1 Integral Representations

It is possible to define representing matrices of the symmetric group over the integers.

`SymmetricRepresentation(pa, pe): SeqEnum, GrpPermElt -> AlgMatElt`

`Al: MonStgElt` Default: `"JamesKerber"`

Given a partition pa of weight n and a permutation pe in a symmetric group of degree n, return an irreducible representing matrix for pe, indexed by pa, over the integers. If `Al` is set to the default `"JamesKerber"` then the method described in [JK81b] is used. If `Al` is set to `"Boerner"` the method described in the book of Boerner [Boe67] is used. If `Al` is set to `"Specht"` then the method used is a direct implementation of that used by Specht in his paper from 1935 [Spe35].

Example: Integral Representations of a Permutation `(ex-be2f6e)`

We compute a representing matrix of a permutation using two different algorithms and check whether the results have the same character.

```
> a:=SymmetricRepresentation([3,2],Sym(5)!(3,4,5) : Al := "Boerner");a;
[ 0  0  1 -1  0]
[ 1  0  0 -1  0]
[ 0  1  0 -1  0]
[ 0  0  0 -1  1]
[ 0  0  0 -1  0]
> b:=SymmetricRepresentation([3,2],Sym(5)!(3,4,5) : Al := "Specht");b;
[ 0  1  0 -1  0]
```

(continues on next page)

(continued from previous page)

```
[ 0  0  1  0 -1]
[ 1  0  0  0  0]
[ 0  0  0  0 -1]
[ 0  0  0  1 -1]
> IsSimilar(Matrix(Rationals(), a), Matrix(Rationals(), b));
true
```

The matrices are similar as they should be.

101.2.2 The Seminormal and Orthogonal Representations

The seminormal and orthogonal representations involve matrices which are not necessarily integral. The method Magma uses to construct these matrices is described in [JK81b, Section 3.3];

`SymmetricRepresentationSeminormal(pa, pe): SeqEnum, GrpPermElt -> AlgMatElt`

Given a partition pa of weight n and a permutation pe in a symmetric group of degree n, return the matrix of the seminormal representation for pe, indexed by pa, over the rationals.

`SymmetricRepresentationOrthogonal(pa, pe): SeqEnum, GrpPermElt -> AlgMatElt`

Given a partition pa of weight n and a permutation pe in a symmetric group of degree n, return the matrix of the orthogonal representation for pe, indexed by pa. An orthogonal basis is used to compute the matrix which may have entries in a cyclotomic field.

Example: Seminormal and Orthogonal Representations `(ex-5248de)`

We compare the seminormal and orthogonal representations of a permutation and note that they are similar.

```
> g:=Sym(5)!(3,4,5);
> a:=SymmetricRepresentationSeminormal([3,2],g);a;
[-1/2    0 -3/4    0    0]
[   0  1/2    0  3/4    0]
[   1    0 -1/2    0    0]
[   0  1/3    0 -1/6  8/9]
[   0    1    0 -1/2 -1/3]
> b:=SymmetricRepresentationOrthogonal([3,2],g);b;
[-1/2 0 zeta(24)_8^2*zeta(24)_3 + 1/2*zeta(24)_8^2 0 0]
[0 1/2 0 -zeta(24)_8^2*zeta(24)_3 - 1/2*zeta(24)_8^2 0]
[-zeta(24)_8^2*zeta(24)_3 - 1/2*zeta(24)_8^2 0 -1/2 0 0]
[0 -1/3*zeta(24)_8^2*zeta(24)_3 - 1/6*zeta(24)_8^2 0
 -1/6 2/3*zeta(24)_8^3 - 2/3*zeta(24)_8]
[0 2/3*zeta(24)_8^3*zeta(24)_3 + 1/3*zeta(24)_8^3
+ 2/3*zeta(24)_8*zeta(24)_3 + 1/3*zeta(24)_8 0
    -1/3*zeta(24)_8^3 + 1/3*zeta(24)_8 -1/3]
> IsSimilar(a,b);
true
```

They should both be of finite order, 3.

```
> IsOne(a^Order(g));
true
> IsOne(b^Order(g));
true
>
```

101.3 Characters of the Symmetric Group

101.3.1 Single Values

The method used to compute the value of a character on a permutation is the recursion formula of Murnaghan and Nakayama [JK81b, p. 60], except when computing the value of a character on the identity permutation a formula for the dimension of the representation indexed by a partition is used, [JK81b, p. 56].

`SymmetricCharacterValue(pa, pe): SeqEnum, GrpPermElt -> RngElt`

Computes the value of the irreducible character of the symmetric group of degree n indexed by the partition pa of weight n on the permutation pe. When computing the value of the character on the identity permutation, i.e. the degree of the character, the dimension (hook) formula is used on the partition pa.

101.3.2 Irreducible Characters

`SymmetricCharacter(pa): SeqEnum -> AlgChtrElt`

Return the character of the representation of the symmetric group of degree n indexed by the partition pa where pa has weight n.

101.3.3 Character Table

`SymmetricCharacterTable(d): RngIntElt -> SeqEnum`

Return the character table of the symmetric group of degree d.

101.4 Representations of the Alternating Group

101.5 Characters of the Alternating Group

There is a special routine which computes the character table of the alternating group, as well as routines which compute values of alternating characters and the characters themselves. These routines make use of the fact that in most cases the irreducible characters of the symmetric group, which can be computed quickly, are also irreducible in the alternating group. So the irreducible characters of the alternating group may be indexed by partitions in the same way as those of the symmetric group. As the restriction of the irreducible character indexed by the partition λ is equal to the restriction of the character indexed by the conjugate partition, we only need to take one from each of these pairs of partitions to form a full set of irreducible characters. When a partition is conjugate to itself the character of the symmetric group indexed by that partition is no longer irreducible but is the sum of two irreducibles. This method is described in [JK81b].

101.5.1 Single Values

`AlternatingCharacterValue(pa, pe): SeqEnum, GrpPermElt -> RngElt`

Return the value of the character of the alternating group of degree n indexed by the partition pa of weight n on the permutation pe. The partition pa and its conjugate should be distinct.

`AlternatingCharacterValue(pa, i, pe): SeqEnum, RngIntElt, GrpPermElt -> RngElt`

Return the value of the ith character of the alternating group of degree n indexed by the self conjugate partition pa of weight n on the permutation pe. Since there are two possible irreducible characters indexed by such partitions i must be either 1 or 2.

101.5.2 Irreducible Characters

`AlternatingCharacter(pa): SeqEnum -> AlgChtrElt`

Return the character of the alternating group of degree n indexed by the partition pa of weight n. The partition pa and its conjugate should be distinct.

`AlternatingCharacter(pa, i): SeqEnum, RngIntElt -> AlgChtrElt`

Return the ith character of the alternating group of degree n indexed by the self conjugate partition pa of weight n. Since there are two possible irreducible characters indexed by such partitions i must be either 1 or 2.

101.5.3 Character Table

`AlternatingCharacterTable(d): RngIntElt -> SeqEnum`

Returns the character table of the alternating group of degree d.

101.6 Bibliography

- [Boe67] H. Boerner. *Darstellungen von Gruppen. 2. Aufl.* Berlin-Heidelberg-New York: Springer-Verlag. XIV, 317 S. , 1967.
- [JK81b] Gordon James and Adalbert Kerber. *The representation theory of the symmetric group.* Addison-Wesley Publishing Co., Reading, Mass., 1981. ISBN 0-201-13515-9. With a foreword by P. M. Cohn, With an introduction by Gilbert de B. Robinson.
- [Spe35] Wilhelm Specht. Die irreduziblen Darstellungen der symmetrischen Gruppe. *Math. Z.*, 39:696–711, 1935.

Chapter 102

Mod P Galois Representations

102.1 Introduction

This package provides tools to work with φ-modules over $k((u))$ where k is a finite field, and representations of the absolute Galois group of $k((u))$ with coefficients in a finite field. The main functionality of the package computes the semisimplification of a given φ-module, and the semisimplification of the Galois representation that is naturally attached to it. In particular, the slopes of the φ-module, corresponding to the tame inertia weights of the Galois representation, can be computed using this package.

102.1.1 Motivation

Let K be a p-adic field and let G_K be the absolute Galois group of K. Representations of this group naturally arise from geometry, namely from the p-adic étale cohomology of a scheme over K.

The study of these representations is a central topic in arithmetic, and a motivation for creating this package is the following: let V be a $\mathbb{Q}_p$-representation of G_K, *i.e.* a $\mathbb{Q}_p$-vector space endowed with a continuous, linear action of G_K. Now let $T \subset V$ be any $\mathbb{Z}_p$-lattice stable under the action of G_K. There always exists such a lattice. Moreover, the quotient T/pT has a natural structure of $\mathbb{F}_p$-representation of G_K. This representation depends on the choice of T, but its semisimplification $(T/pT)^{ss}$ does not, according to the Brauer-Nesbitt theorem. Recall the semisimplification of a representation is the direct sum of the composition factors appearing in any Jordan-Holder sequence of this representation. Therefore, it is an interesting question to determine properties of $(T/pT)^{ss}$ in terms of V. Although the Fontaine-Laffaille theory completely addresses this question for some V, the general case remains an open question. Some computations concerning this problem can be performed in Magma using this package.

102.1.2 Definitions

Let k be a finite field of characteristic p, and let $K = k((u))$ be the field of Laurent series with coefficients in k. Let $s \geq 0$ and $b \geq 2$ be integers. We define a "Frobenius" map σ on K by the following formula:

$$\sigma\left(\sum_{i\in\mathbb{Z}} a_i u^i\right) = \sum_{i\in\mathbb{Z}} a_i^{p^s} u^{bi}.$$

A φ-module over K is the data of a finite-dimensional K-vector space D, endowed with an endomorphism $\varphi : D \to D$ that is semilinear with respect to σ. This means that for all $\lambda \in K$, $x \in D$, we have the identity $\varphi(\lambda x) = \sigma(\lambda)\varphi(x)$.

A φ-module is said to be étale if the map φ is injective. A φ-module can be described by the matrix representing the action of φ on some basis of D, and it is étale if and only if this matrix is invertible.

102.1.3 Classification of φ-modules

Some φ-modules play a crucial role in the theory because they are the simple objects in the category of étale φ-modules over the maximal unramified extension K^{ur} of K.

Let $d \geq 1$, $h \in \mathbb{Z}$, $\lambda \in \bar{k}$. We define the φ-module $D(d, s, \lambda)$ as the φ-module of dimension d whose matrix in some basis is the companion matrix of the polynomial $T^d - u^h$. We also write $D(d, h) = D(d, h, 1)$. Note that in general there are several ways to extend the action of σ on K^{ur}, but we may only distinguish the cases where σ acts as identity on k, and the case where is does not. We say that a couple (d, h) is reduced if there is no divisor d' of d (except d) such that $\frac{b^{d'}-1}{b^d-1}$ is a divisor of h. The main classification results are the following:

If $\sigma \neq id$, the simple objects of the category of étale φ-modules over K^{ur} are the $D(d, h)$ for (d, h) reduced, and if $\sigma = id$, the simple objects of the category of étale φ-modules over K^{ur} are the $D(d, h, \lambda)$ for (d, h) reduced.

By definition, the slope of a simple φ-module isomorphic to $D(d, h, \lambda)$ is the rational number $\frac{h}{b^d-1}$, up to the equivalence relation "$x \sim y \Leftrightarrow \exists m, n \in \mathbb{N}$ such that $b^m x - b^n y \in \mathbb{Z}$". With this equivalence relation, the definition does not depend on the choice of (d, h).

If D is a φ-module over K, the slopes of D are the collection of the slopes of the composition factors of $K^{ur} \otimes_K D$ (this notion does not depend on how σ is extended to K^{ur}). Note that even though the algorithms that we present can give decompositions over K, for most practical uses the knowledge of the slopes should be sufficient.

102.1.4 Connection with Galois Representations

Let us explain the link between Galois representations and φ-modules over K. In this section, we assume that σ is the classical Frobenius $x \mapsto x^p$. Let K^{sep} be a separable closure of K and let $G_K = Gal(K^{sep}/K)$ be the absolute Galois group of K.

A theorem of Katz states that there is an equivalence of categories between the étale φ-modules over K and the $\mathbb{F}_p$-representations of G_K.

Under this equivalence of categories, the φ-module $D(d, h)$ corresponds to the "fundamental character of level d" to the power h, ω_d^h, seen as a $\mathbb{F}_p$-representation. The figures of h in base p are called the tame inertia weights of the representation, because they describe the action of the tame inertia group on the representation. These weights can be recovered from the slope of the φ-module. It is worth noting that if F is a p-adic field whose residue field is k, and F_∞ is the extension of F generated by a compatible sequence of p^n-th roots of the uniformizer for all n, then G_{F_∞} is isomorphic to G_K. Moreover, the tame inertia weights of a $\mathbb{F}_p$-representation of G_F are the same as the tame inertia weights of its restriction to G_{F_∞}, seen as a representation of G_K. Hence, working with φ-modules will enable us to study representations of p-adic Galois groups.

102.2 φ-modules and Galois Representations in Magma

Let us now give an overview of the functionalities of the package.

102.2.1 φ-modules

Category

In Magma, φ-modules have type `PhiMod`. Elements of φ-modules have type `PhiModElt`.

Creation Functions

`PhiModule(M): AlgMatElt -> PhiMod`

```
F: SeqEnum                    Default: [1,p]
```

Create the φ-module whose matrix is given by M in some basis. The optional argument F describes the action of the Frobenius on coefficients: if $F = [s, b]$ then φ acts by $a \mapsto a^{p^s}$ on the residue field and maps the variable u to u^b. The default value is $[1, p]$ where p is the characteristic of the base field, corresponding to the absolute Frobenius.

`ElementaryPhiModule(S, d, h): RngSerLaur, RngIntElt, RngIntElt -> PhiMod`

```
F: SeqEnum                    Default: [1,p]
```

Create the φ-module $D(d, s)$ whose matrix is the companion matrix of $T^d - u^s$.

`PhiModuleElement(x, D): AlgMatElt, PhiMod -> PhiModElt`

Create the element of the φ-module D whose coordinates are given by the vector x.

Attributes of φ-modules

`Dimension(D): PhiMod -> RngIntElt`

The dimension of a φ-module.

`CoefficientRing(D): PhiMod -> RngSerLaur`

The coefficient ring of a φ-module.

`FrobeniusMatrix(D): PhiMod -> AlgMatElt`

Return the matrix of the action of φ on D in the current basis.

Basic Operations and Properties of φ-modules

`IsEtale(D): PhiMod -> BoolElt`

Return `true` if the action of φ on D is injective. This is only possible up to the precision of the coefficient ring of D.

`ChangePrecision(~D, prec): ~PhiMod, RngIntElt`

Change the precision of the coefficient ring of D to `prec`.

`DirectSum(D1, D2): PhiMod, PhiMod -> PhiMod`

The direct sum of two φ-modules. The coefficient rings and Frobenius action on the coefficients must be the same.

`BaseChange(~D, P): ~PhiMod, AlgMatElt`

Change the basis of D. The base change matrix is P, meaning that if G is the current matrix of φ, the new matrix will be $P^{-1}G\varphi(P)$.

`RandomBaseChange(~D): PhiMod`

Randomly change the basis of D.

`Phi(D, x): PhiMod, PhiModElt -> PhiModElt`

Compute the image of $x \in D$ under the action of φ.

Reduction of φ-modules and Galois Representations

`SemisimpleDecomposition(D): PhiMod -> AlgMatElt, AlgMatElt, SeqEnum, SeqEnum`

Compute a Jordan-Holder sequence for the φ-module D. The result G, P, sl, pol is as follows: G is the matrix of φ in a basis where it is block upper triangular, with diagonal blocks corresponding to simple φ-modules. The matrix P gives the corresponding basis. The list sl is the list of the slopes of D, and the list pol is a list of polynomials. The isomorphism class of a simple block of G is determined by the corresponding slope and polynomial.

`Slopes(D): PhiMod -> SeqEnum`

Compute the list of slopes of D (with multiplicities).

`SSGaloisRepresentation(D): PhiMod -> SSGalRep`

Compute the semisimplification of the Galois representation corresponding to D.

102.2.2 Semisimple Galois Representations

This part is dedicated to the study of representations of absolute Galois groups of fields of the form $k((u))$ with k finite, and with coefficients in finite fields. The implementation is for semisimple representations, and these are described by their tame inertia weights and polynomials giving the action of the Frobenius on the unramified part.

Category

In Magma, semisimple Galois representations have type `SSGalRep`.

Creation Functions

`SSGaloisRepresentation(E, K, w, P): FldFin, RngSerLaur, SeqEnum, SeqEnum -> SSGalRep`

Create the semisimple representation of the absolute Galois group of K with coefficients in E, tame inertia weights given by w, and action of the Frobenius described by the elements of the list P.

Basic Operations

`CoefficientRing(V): SSGalRep -> FldFin`

The coefficient ring.

`FixedField(V): SSGalRep -> RngSerLaur`

The fixed field of the absolute Galois group of which V is a representation.

`Weights(V): SSGalRep -> SeqEnum`

The tame inertia weights of V.

Representation Associated to a φ-Module

`SSGaloisRepresentation(D): PhiMod -> SSGalRep`

If D is a φ-module over a field K of Laurent series this returns the semisimplification of the representation associated to D.

102.3 Examples

Example: Slopes of a Direct Sum of Phi-Modules (ex-b604c9)

We create two φ-modules D_1, D_2, build their direct sum, and compute its slopes and corresponding representation.

```
> k<e9> := GF(3,2);
> S<u> := LaurentSeriesRing(k,20);
> D1 := ElementaryPhiModule(S,3,2);
> D1;
Phi-module of dimension 3 over Laurent series field in u over GF(3^2)
with fixed relative precision 20 with matrix
[ O(u^20)         O(u^20)        u^2 + O(u^20)]
[ 1 + O(u^20)     O(u^20)             O(u^20) ]
[ O(u^20)        1 + O(u^20)          O(u^20) ] and Frobenius [1,3]
> M := Matrix(S,2,2,[0,k.1*u,1,0]);
> D2 := PhiModule(M);
> D2;
Phi-module of dimension 2 over Laurent series field in u over GF(3^2)
with fixed relative precision 20 with matrix
[ O(u^20)       e9*u^2 + O(u^20)]
[ 1 + O(u^20)            O(u^20) ] and Frobenius [1,3]
> D := DirectSum(D1,D2);
```

(continues on next page)

(continued from previous page)

```
> Slopes(D);
[
  [2, 1],
  [3, 2]
]
> SSGaloisRepresentation(D);
Semisimple representation of the absolute Galois group of
Laurent series field in u over GF(3^2) with fixed relative
precision 20 with coefficients in Finite field of size 3 and
components [
[3, 18],
[2, 3]
]
```

Chapter 103

Group Representations

103.1 Introduction

The first version of this package was released in V2.29 (October 2025). It has been developed further in each subsequent release, and is still under development. We encourage users to send feedback regarding the package, and desirable features or improvements.

The package contains implementations of certain representations of groups on free modules of finite rank, where the groups are not assumed to be finite or even finitely generated. The actions are assumed to be on the left (V is a left G-module). The goal is to be able to construct representations of reductive groups. The primary focus is on orthogonal and unitary groups.

The main purposes of the current functionality are to serve as a basis for algebraic modular forms.

103.1.1 Definitions and Background

Let G be a group, let R be a commutative ring, and let V be a left $R[G]$-module. Then V is called a representation of G. This chapter describes function available in Magma for computations concerning such representations.

103.1.2 Categories

The Magma category for group representations is `ModRed`, while elements in these spaces have type `ModRedElt`. Morphisms between group representations are linear maps which are compatible with the group action, and have type `ModRedHom`.

The representations are built on a category of combinatorial free modules, whose Magma type is `CombFreeMod`, while elements in such modules have type `CombFreeModElt`.

103.2 Creation of Group Representations

103.2.1 General Group Representations

`GroupRepresentation``(G, M, action): Grp, CombFreeMod, MonStgElt -> ModRed`

`GroupRepresentation``(G, M, action): GrpRed, CombFreeMod, MonStgElt -> ModRed`

`params: List` Default: `[* *]`

Constructs a group representation for the group G on the combinatorial free module M with basis B, such that the action on basis elements $G \times B \to M$ is described by the map given by the string `action`.

103.2.2 Subrepresentations

`Subrepresentation(V, t): ModRed, Any -> ModRed, ModRedHom`

`SubCFModule(V, t): CombFreeMod, Any -> CombFreeMod, CombFreeModHom`

The subrepresentation (or submodule) of V whose underlying free module is generated by t.

103.2.3 Natural Representations

`TrivialRepresentation(G, R): Grp, Rng -> ModRed`

`TrivialRepresentation(G, R): GrpRed, Rng -> ModRed`

```
name: MonStgElt                    Default: "v"
```

The trivial representation for the group G over the ring R, where the basis element has name `name`.

`StandardRepresentation(G): GrpMat -> ModRed`

`StandardRepresentation(G): GrpRed -> ModRed`

```
name: MonStgElt                    Default: "x"
```

The standard representation of the matrix group G over its ring of definition R, i.e. the representation obtained by considering its given embedding in $\mathrm{GL}_n(R)$ acting on R^n by invertible linear transformations. The basis will have names $x_1, \ldots, x_n$, where x is specified by `name`.

`SpinorNormRepresentation(G, d): GrpRed, RngIntElt -> ModRed`

```
name: MonStgElt                    Default: "x"
```

The 1-dimensional representation spin_d of the orthogonal group G induced by the spinor norm and d.

`Rho(G, k, j): GrpMat, RngIntElt, RngIntElt -> ModRed`

The representation $\det^k \otimes \mathrm{Sym}^j$.

`SymSpinor(G, d, k): GrpRed, RngIntElt, RngIntElt -> ModRed`

The representation $\mathrm{spin}_d \otimes \mathrm{Sym}^k$ of the orthogonal group G.

`AltSpinor(G, d): GrpRed, RngIntElt, RngIntElt -> ModRed`

The representation $\mathrm{spin}_d \otimes \mathrm{Alt}^j$ of the orthogonal group G.

`RadicalSignCharacterSinglePrime(G, p): GrpRed, RngIntElt -> ModRed`

The character θ_p from [DPRT24]. If $G = \mathrm{O}(Q)$ is an orthogonal group defined over $\mathbb{Q}$, with Q integral, and p is a prime divisor of $\mathrm{Disc}(Q)$, returns the 1-dimensional representation of $G(\mathbb{Z}_p)$ on the determinant of the radical of Q mod $2p$.

`RadicalSignCharacter(G, d): GrpRed, RngIntElt -> ModRed`

The character $\theta_d = \prod_{p|d} \theta_p$, where θ_p is the character constructed by the function `RadicalSignCharacterSinglePrime`.

`SpinRepresentation(G, p): GrpRed, RngIntElt -> ModRed`

The spin representation of the special orthogonal group $G = \mathrm{SO}(Q)$ with coefficients in $\mathbb{F}_p$.

103.2.4 New Representations from Old

`DeterminantRepresentation(G): GrpMat -> ModRed`

```
k   : RngIntElt                    Default: 1
name: MonStgElt                    Default: "v"
```

The 1-dimensional representation, where G acts via the $g \mapsto \det(g)^k$, where the basis element has name `name`.

`SymmetricRepresentation(V, n): ModRed, RngIntElt -> ModRed`

The symmetric representation $\mathrm{Sym}^n(V)$.

`AlternatingRepresentation(V, n): ModRed, RngIntElt -> ModRed`

The alternating representation $\mathrm{Alt}^n(V)$.

`DualRepresentation(V): ModRed -> ModRed`

The dual (contragredient) representation $V^\vee$.

`TensorProduct(V, W): ModRed, ModRed -> ModRed`

The representation $V \otimes W$ with the diagonal action.

`TensorPower(V, d): ModRed, RngIntElt -> ModRed`

The tensor power representation $V^{\otimes d}$, with the diagonal action.

`Pullback(V, f, G): ModRed, MonStgElt, Grp -> ModRed`

If $f : G \to H$ is a group homomorphism, and V is a representation of H, returns the pullback of V via f to a representation of G. Does not verify that f is a group homomorphism.

103.2.5 New Combinatorially Free Modules from Old

`ExteriorPower(M, n): CombFreeMod, RngIntElt -> CombFreeMod`

`AlternatingPower(M, n): CombFreeMod, RngIntElt -> CombFreeMod`

Returns $\bigwedge^n M$.

`ExteriorAlgebra(M): CombFreeMod -> CombFreeMod, CombFreeModHom`

The underlying module of the exterior algebra of M, namely $\bigwedge^\bullet M$, together with an embedding of M as the degree 1 component.

`DirectSum(M): [ CombFreeMod ] -> CombFreeMod`

The direct sum $\bigoplus_i M_i$.

103.2.6 Highest Weight Representations

`GroupRepresentation(G, w): GrpLie, [ RngIntElt ] -> ModRed`

The algebraic representation of the group G with highest weight w. Embeds the group of Lie type G into its standard representation.

`HighestWeightRepresentation(G, w): GrpRed, [ RngIntElt ] -> ModRed`

The irreducible algebraic representation of G with highest weight w.

`HighestWeightRepresentation(G, w, p): GrpRed, [ RngIntElt ], RngIntElt -> ModRed`

If G is a reductive group defined over a number field F, return the irreducible algebraic representation of G with highest weight w, and coefficient field of characteristic p, obtained by the reduction modulo a prime P of F above p.

103.2.7 Creation of Combinatorial Free Modules

`CombinatorialFreeModule(R, S): Rng, SetIndx -> CombFreeMod`

`CombinatorialFreeModule(R, S): Rng, [ MonStgElt ] -> CombFreeMod`

```
params: List                    Default: [* *]
```

A combinatorial free module over the ring R with basis given by S, namely $M = R^S$.

103.3 Basic Properties

103.3.1 Accessing Representation Information

`Rank(V): ModRed -> RngIntElt`

`Dimension(V): ModRed -> RngIntElt`

`Ngens(V): ModRed -> RngIntElt`

The rank of the free module underlying the representation V.

`Basis(V): ModRed -> [ ModRedElt ]`

A basis for the free module underlying the representation V.

`BaseRing(V): ModRed -> Rng`

The coefficient ring, the ring over which V is defined.

`Group(V): ModRed -> Grp`

The group G that acts on V.

`CFM(V): ModRed -> CombFreeMod`

The combinatorial free module underlying V.

103.3.2 Accessing Combinatorial Free Module Information

`Rank(M): CombFreeMod -> RngIntElt`

`Dimension(M): CombFreeMod -> RngIntElt`

`Ngens(M): CombFreeMod -> RngIntElt`

The rank of M.

`Basis(M): CombFreeMod -> [ CombFreeModElt ]`

A basis for M.

`BaseRing(M): CombFreeMod -> Rng`

The ring R over which M is a module.

`Names(M): CombFreeMod -> SetIndx`

`Names(M): CombFreeMod -> [ MonStgElt ]`

The names of the basis elements.

103.3.3 Predicates

`IsTrivial(V): ModRed -> BoolElt`

Returns `true` iff V is the trivial representation.

103.4 Operations on Group Representations

`Intersection(V, W): ModRed, ModRed -> ModRed`

`V meet W: ModRed, ModRed -> ModRed`

If $V, W \subseteq U$ are both subrepresentations of U, return the intersection $V \cap W \subseteq U$.

`V eq W: ModRed, ModRed -> BoolElt`

`V eq W: CombFreeMod, CombFreeMod -> BoolElt`

Returns `true` if the representations V and W are equal; otherwise `false`

103.4.1 Base Change

`ChangeRing(V, S): ModRed, Rng -> ModRed`

The group representation $V_S = V \otimes_R S$ with base ring changed to S.

`ChangeRing(M, S): CombFreeMod, Rng -> CombFreeMod`

The module $M_S = M \otimes_R S$ with base ring changed to S.

103.4.2 Other Operations

`FixedSubspace(H, V): GrpMat, ModRed -> ModRed`

Given a subgroup $H \subseteq G$, returns the subspace V^H, the vectors of V fixed by H.

103.5 Elements of Group Representations

103.5.1 Creation of Elements

`GroupRepresentationElement(V, m): ModRed, CombFreeModElt -> ModRedElt`

An element of the group representation V whose underlying vector is m.

`CombinatorialFreeModuleElement(M, v): CombFreeMod, ModRngElt -> CombFreeModElt`

`CombinatorialFreeModuleElement(M, v): CombFreeMod, ModEDElt -> CombFreeModElt`

`CombinatorialFreeModuleElement(M, v): CombFreeMod, ModTupFldElt[Fld] -> CombFreeModElt`

An element of M whose underlying vector is v.

103.5.2 Basic Properties

`Parent(v): ModRedElt -> ModRed`

`Parent(v): CombFreeModElt -> CombFreeMod`

The parent of v.

`ActionMatrix(V, g): ModRed, GrpElt -> GrpMatElt`

The matrix describing the action of g on V.

103.5.3 Operations on Elements

`v + w: ModRedElt, ModRedElt -> ModRedElt`

`v + w: CombFreeModElt, CombFreeModElt -> CombFreeModElt`

Given $v, w \in V$, return $v + w \in V$.

`v - w: ModRedElt, ModRedElt -> ModRedElt`

`v - w: CombFreeModElt, CombFreeModElt -> CombFreeModElt`

Given $v, w \in V$, return $v - w \in V$.

`a * v: RngElt, ModRedElt -> ModRedElt`

`a * v: RngElt, CombFreeModElt -> CombFreeModElt`

Given $a \in R$ and $v \in V$, return $av \in V$.

`g * v: GrpElt, ModRedElt -> ModRedElt`

Given $g \in G$ and $v \in V$, returns $g(v) = g \cdot v$.

`m * v: AlgMatElt, ModRedElt -> ModRedElt`

Given $v \in V \simeq R^n$ and $m \in M_n(R) \simeq \mathrm{End}(V)$, return $m(v) \in V$.

`v ^ w: CombFreeModElt, CombFreeModElt -> CombFreeModElt`

Given $v, w \in \bigwedge^\bullet M$, return $v \wedge w$.

103.5.4 Comparisons and Membership

`v eq w: ModRedElt, ModRedElt -> BoolElt`

Returns `true` if the elements v and w of a representation V are equal; otherwise `false`.

`v in V: ModRedElt, ModRed -> BoolElt`

`v in V: CombFreeModElt, CombFreeMod -> BoolElt`

Returns `true` if v is in the representation V; otherwise `false`.

103.5.5 Other Operations

`Eltseq(v): ModRedElt -> []`

`Eltseq(v): CombFreeModElt -> []`

Given an element v of a representation V, returns a sequence representing v.

`ChangeRing(v, S): CombFreeModElt, Rng -> CombFreeModElt`

Given an element $v \in M$, where M is a combinatorial R-module, return $v \otimes 1 \in M \otimes_R S$.

103.6 Homomorphisms of Group Representations

103.6.1 Creation of Homomorphisms between Group Representations

`Homomorphism(V, W, f): ModRed, ModRed, UserProgram -> ModRedHom`

`Homomorphism(V, W, f): ModRed, ModRed, Map -> ModRedHom`

`Homomorphism(V, W, f): ModRed, ModRed, CombFreeModHom -> ModRedHom`

Construct a homomorphism of group representations described by $f : V \to W$. Does not verify that the map indeed describes a homomorphism of group representations.

`Homomorphism(V, W, S): ModRed, ModRed, SeqEnum -> ModRedHom`

`BaseChangeCodomain: BoolElt`	`Default: false`

Construct a homomorphism of group representations $f : V \to W$, mapping the basis of V to S. If `BaseChangeCodomain` is `true`, returns a homomorphism $f : V \to W \otimes R$, where R is the base ring of V. Does not verify that the map indeed describes a homomorphism of group representations.

`Homomorphism(M, N, f): CombFreeMod, CombFreeMod, UserProgram -> CombFreeModHom`

`Homomorphism(M, N, f): CombFreeMod, CombFreeMod, Map -> CombFreeModHom`

Construct a homomorphism of R-modules described by $f : M \to N$.

`Homomorphism(M, N, S): CombFreeMod, CombFreeMod, SeqEnum -> CombFreeModHom`

`BaseChangeCodomain: BoolElt`	`Default: false`

Construct a homomorphism of R-modules $f : M \to N$, mapping the basis of M to S. If `BaseChangeCodomain` is `true`, returns a homomorphism $f : M \to N \otimes R$, where R is the base ring of V.

103.6.2 Properties of Homomorphisms of Group Representations

`Domain(f): ModRedHom -> ModRed`

`Domain(f): CombFreeModHom -> CombFreeMod`

The domain of f.

`Codomain(f): ModRedHom -> ModRed`

`Codomain(f): CombFreeModHom -> CombFreeMod`

The codomain of f.

`Kernel(f): ModRedHom -> ModRed`

The kernel of f, as a representation of G.

103.6.3 Operations on Homomorphisms of Group Representations

`Evaluate(f, v): ModRedHom, ModRedElt -> ModRedElt`

`Evaluate(f, v): CombFreeModHom, CombFreeModElt -> CombFreeModElt`

`v @ f: ModRedElt, ModRedHom -> ModRedElt`

`v @ f: CombFreeModElt, CombFreeModHom -> CombFreeModElt`

Returns $f(v)$.

`w @@ f: ModRedElt, ModRedHom -> ModRedElt`

`w @@ f: CombFreeModElt, CombFreeModHom -> CombFreeModElt`

Given a homomorphism $f : V \to W$, and an element $w \in W$, returns an element $v \in V$ such that $f(v) = w$.

103.7 Bibliography

- [DPRT24] N. Dummigan, A. Pacetti, G. Rama, and G. Tornaría. Quinary forms and paramodular forms. *Math. Comp.*, 93(348):1805–1858, 2024. URL: https://doi.org/10.1090/mcom/3815, doi:10.1090/mcom/3815.

References

[Boe67] H. Boerner. *Darstellungen von Gruppen. 2. Aufl.* Berlin-Heidelberg-New York: Springer-Verlag. XIV, 317 S. , 1967.

[BW15] Peter A. Brooksbank and James B. Wilson. The module isomorphism problem reconsidered. *J. Algebra*, 421:541–559, 2015. URL: https://doi.org/10.1016/j.jalgebra.2014.09.004, doi:10.1016/j.jalgebra.2014.09.004.

[Con90a] S. B. Conlon. Calculating characters of p-groups. *J. Symbolic Comp.*, 9:535–550, 1990.

[DK09] Susanne Danz and Burkhard Külshammer. Vertices, sources and Green correspondents of the simple modules for the large Mathieu groups. *J. Algebra*, 322(11):3919–3949, 2009.

[DKZ08] Susanne Danz, Burkhard Külshammer, and René Zimmermann. On vertices of simple modules for symmetric groups of small degrees. *J. Algebra*, 320(2):680–707, 2008.

[Dix67] J. D. Dixon. High–speed computation of group characters. *Numerische Mathematik*, 10:446–450, 1967.

[DPRT24] N. Dummigan, A. Pacetti, G. Rama, and G. Tornaría. Quinary forms and paramodular forms. *Math. Comp.*, 93(348):1805–1858, 2024. URL: https://doi.org/10.1090/mcom/3815, doi:10.1090/mcom/3815.

[Fei83] Walter Feit. The computations of some Schur indices. *Israel J. Math.*, 46(4):274–300, 1983.

[Fra82] J. S. Frame. Recursive computation of tensor power components. *Bayreuth. Math. Schr.*, pages 153–159, 1982.

[Gre59] J.A. Green. On the indecomposable representations of a finite group. *Math. Z.*, 70:430–445, 1958/59.

[JK81b] Gordon James and Adalbert Kerber. *The representation theory of the symmetric group.* Addison-Wesley Publishing Co., Reading, Mass., 1981. ISBN 0-201-13515-9. With a foreword by P. M. Cohn, With an introduction by Gilbert de B. Robinson.

[Lit40] D. E. Littlewood. *The Theory of Group Characters and Matrix Representations of Groups.* Clarendon Press (Oxford), 1940.

[Lit58] D. E. Littlewood. Products and plethysms of characters with orthogonal, symplectic and symmetric groups. *Canad. J. Math*, 10:17–32, 1958.

[Mur58] F. D. Murnaghan. The orthogonal and symplectic groups. *Comm. Dublin Inst. Adv. Studies. Ser. A, no.*, 13:146, 1958.

[New51] M. J. Newell. Modification rules for the orthogonal and symplectic groups. *Proc. Royal Irish Acad. Section A*, 54:153–163, 1951.

[Sch90b] G. J. A. Schneider. Dixon's Character Table Algorithm Revisited. *J. Symbolic Computation*, 9:601–606, 1990.

[Spe35] Wilhelm Specht. Die irreduziblen Darstellungen der symmetrischen Gruppe. *Math. Z.*, 39:696–711, 1935.

[Tur01] A. Turull. Schur indices of perfect groups. *Proc. Amer. Math. Soc.*, 130(2):367–370, 2001.

[Ung06a] W.R. Unger. Computing the character table of a finite group. *J. Symbolic Comp.*, 41(8):847–862, 2006.

Index

Symbols

+ y (*operation*), 69, 83
# L (*operation*), 34
# T (*operation*), 70
# x (*operation*), 71
- u (*operation*), 19
- y (*operation*), 69, 83
3-Modular Characters of the Higman-Sims Group (*example*), 83

A

a * v (*operation*), 103
a * x (*operation*), 69, 83
a ^ v (*operation*), 61
a in F (*operation*), 69
a notin F (*operation*), 69
AbsoluteCartanMatrix(G, K) (*function*), 33
AbsolutelyIrreducibleConstituents(M) (*function*), 56
AbsolutelyIrreducibleModule(M) (*function*), 24
AbsolutelyIrreducibleModules(G) (*function*), 32
AbsolutelyIrreducibleModules(G, K : parameters) (*function*), 31, 53
AbsolutelyIrreducibleModulesBurnside(G, K : parameters) (*function*), 55
AbsolutelyIrreducibleModulesSchur(G, K) (*function*), 31, 32
AbsolutelyIrreducibleModulesSchur(G, k, i: parameters) (*function*), 56
AbsolutelyIrreducibleModulesSchur(G, k, L, i: parameters) (*function*), 56
AbsolutelyIrreducibleModulesSchur(G, k, L: parameters) (*function*), 56
AbsolutelyIrreducibleModulesSchur(G, K: parameters) (*function*), 56
AbsolutelyIrreducibleModulesSoluble(G, p) (*function*), 31
AbsoluteModuleOverMinimalField(M) (*function*), 12, 13
AbsoluteModuleOverMinimalField(M, F) (*function*), 12
AbsoluteModulesOverMinimalField(Q, F) (*function*), 12
Access Functions for a Matrix Algebra Module (*example*), 16
Action Matrices of Algebra Elements (*example*), 62
Action(M) (*function*), 15
ActionGenerator(M, i) (*function*), 15, 18
ActionGenerators(M) (*function*), 18
ActionGroup(M) (*function*), 18
ActionMatrix(M, a) (*function*), 62
ActionMatrix(V, g) (*function*), 103
AHom(M, N) (*function*), 37
Algebra(M) (*function*), 62
AlternatingCharacter(pa) (*function*), 92
AlternatingCharacter(pa, i) (*function*), 92
AlternatingCharacterTable(d) (*function*), 93
AlternatingCharacterTable(n) (*function*), 30
AlternatingCharacterValue(pa, i, pe) (*function*), 92
AlternatingCharacterValue(pa, pe) (*function*), 92
AlternatingPower(M, n) (*function*), 101
AlternatingRepresentation(V, n) (*function*), 101
AltSpinor(G, d) (*function*), 100
Applying the Meataxe to O5(3) (*example*), 24
AssertAttribute(x, "IsCharacter", b) (*procedure*), 74
AutomorphismGroup(M) (*function*), 42

B

BaseChange(~D, P) (*procedure*), 96
BaseRing(M) (*function*), 15, 102
BaseRing(V) (*function*), 102
Basis(M) (*function*), 62, 102
Basis(R) (*function*), 66
Basis(V) (*function*), 102
Bimodule from a Direct Product (*example*), 51
Bimodule from Sym(3) and a Cyclic Group (*example*), 51
Bimodule(G,H,M) (*function*), 51
Bimodule(M,N) (*function*), 50
Blocks(T, p) (*function*), 83
Bottom(L) (*function*), 35
BrauerCharacter(M) (*function*), 82
BrauerCharacter(x, p) (*function*), 82
BrauerCharacteristic(R) (*function*), 82
BrauerCharacterTable(G, p) (*function*), 31, 82

C

CartanMatrix(G, K) (*function*), 33
Category(R) (*function*), 68
Centre(x) (*function*), 69
CentreOfEndomorphismRing(M) (*function*), 42
CFM(V) (*function*), 102

ChangePrecision(~D, prec) (*procedure*), 96
ChangeRing(M, S) (*function*), 11, 103
ChangeRing(M, S, f) (*function*), 11
ChangeRing(V, S) (*function*), 103
ChangeRing(v, S) (*function*), 104
Character Table Data for E6(2) (*example*), 85
Character Table of A5 from Subgroups (*example*), 79
Character Table of an Almost Simple Group (*example*), 68
Character(M) (*function*), 30, 31
CharacterDegrees(G) (*function*), 67
CharacterDegrees(G, z, p) (*function*), 67
CharacterDegreesPGroup(G) (*function*), 67
CharacterField(x) (*function*), 69
CharacterRing(G) (*function*), 64
CharacterRing(G, p) (*function*), 82
CharacterRing(Q) (*function*), 64
CharacterRing(R, p) (*function*), 82
Characters from a Representation of A7 (*example*), 17
CharacterTable(A) (*function*), 85
CharacterTable(G :parameters) (*function*), 66
CharacterTable(G) (*function*), 30
CharacterTable(N) (*function*), 85
CharacterTableConlon(G) (*function*), 66
CharacterTableDS(G :parameters) (*function*), 66
CharacterTableNames() (*function*), 85
CharacterWithSchurIndex(n: parameters) (*function*), 74
ClassesData(R) (*function*), 68
ClassFunctionSpace(G) (*function*), 64
ClassFunctionSpace(Q) (*function*), 64
ClassPowerCharacter(x, j) (*function*), 71
Codomain(f) (*function*), 104
CoefficientField(x) (*function*), 69
CoefficientRing(D) (*function*), 96
CoefficientRing(M) (*function*), 15, 62
CoefficientRing(V) (*function*), 97
Cohomological Dimension over Alt(8) (*example*), 47
CohomologicalDimension(CM, n) (*function*), 46
CohomologicalDimension(M, n) (*function*), 46
CohomologicalDimensions(M, n) (*function*), 47
Cohomology of a PSL(3,3) Module (*example*), 47
CohomologyGroup(CM, n) (*function*), 46
CohomologyModule(G, M) (*function*), 46
CombinatorialFreeModule(R, S) (*function*), 101
CombinatorialFreeModuleElement(M, v) (*function*), 103
Complements(M, S) (*function*), 29
Composition Factors of a Tensor Square (*example*), 18
Composition Series for O5(3) (*example*), 25
Composition(T, q) (*function*), 78
CompositionFactors(M) (*function*), 25
CompositionSeries(M) (*function*), 25
Computing Schur Indices (*example*), 72
Constituent of a Permutation Module for A7 (*example*), 7
Constituents(M) (*function*), 25
ConstituentsWithMultiplicities(M) (*function*), 25
Constructing a Character of Schur Index 6 (*example*), 73
Constructing the Dual Module (*example*), 17
Creating a Submodule Lattice (*example*), 34

D

Decomposition of a Matrix Algebra Module (*example*), 29
Decomposition(M) (*function*), 29
Decomposition(T, y) (*function*), 78
DecompositionMatrix(G, K) (*function*), 33
DefectGroup(T, b, p) (*function*), 83
DefectGroup(x, p) (*function*), 83
Degree(x) (*function*), 71
DegreeOfCharacterField(x) (*function*), 69
DeterminantRepresentation(G) (*function*), 100
Dimension(D) (*function*), 96
Dimension(e) (*function*), 35
Dimension(M) (*function*), 62, 102
Dimension(V) (*function*), 102
Direct Sums of Algebra Modules (*example*), 63
DirectSum(D1, D2) (*function*), 96
DirectSum(M) (*function*), 101
DirectSum(M, N) (*function*), 9
DirectSum(Q) (*function*), 9, 62
DirectSumDecomposition(M) (*function*), 29
DisplaySocleStructure(SL) (*procedure*), 28
Domain(f) (*function*), 104
Dual(M) (*function*), 10
DualRepresentation(V) (*function*), 101

E

e + f (*operation*), 35
e eq f (*operation*), 35
e meet f (*operation*), 35
e subset f (*operation*), 35
ElementaryPhiModule(S,d,h) (*function*), 96
ElementToSequence(u) (*function*), 19
elt< M | a_1, ..., a_n > (*constructor*), 18
elt< R | a_1, ..., a_k :parameters> (*constructor*), 64
Eltseq(u) (*function*), 19
Eltseq(v) (*function*), 104
Endomorphism Ring of a Permutation Module (*example*), 42
EndomorphismAlgebra(M) (*function*), 42
EndomorphismRing(M) (*function*), 42
Endomorphisms of a G-Homomorphism Module (*example*), 39
Evaluate(f, v) (*function*), 105
Ext(M, N) (*function*), 47

Extension(M, N, e, r) (*function*), 48
Extensions of Alt(5) Modules (*example*), 48
ExteriorAlgebra(M) (*function*), 101
ExteriorPower(M, n) (*function*), 101
ExteriorSquare(M) (*function*), 9

F

Fix(M) (*function*), 11
FixDual(M) (*function*), 11
FixDualMod(M,H) (*function*), 11
FixedField(V) (*function*), 97
FixedSubspace(H, V) (*function*), 103
FixMod(M, H) (*function*), 11
FrobeniusMatrix(D) (*function*), 96

G

g * v (*operation*), 103
g @ x (*operation*), 71
GaloisConjugate(x, j) (*function*), 71
GaloisOrbit(x) (*function*), 71
Generators(M) (*function*), 15
GHom(M, N) (*function*), 37
GHomOverCentralizingField(M, N) (*function*), 37
GModule(G) (*function*), 5
GModule(G, A) (*function*), 3, 7
GModule(G, A, B) (*function*), 7
GModule(G, I, J) (*function*), 8
GModule(G, K) (*function*), 5
GModule(G, P, d) (*function*), 8
GModule(G, Q) (*function*), 3, 8
GModule(x) (*function*), 32
GModuleAction(M) (*function*), 17
GModuleOfQuotient(M, H) (*function*), 17
Group(M) (*function*), 16
Group(R) (*function*), 68
Group(V) (*function*), 102
GroupRepresentation(G, M, action) (*function*), 99
GroupRepresentation(G, w) (*function*), 101
GroupRepresentationElement(V, m) (*function*), 103
GTensorProduct(M, N) (*function*), 9
GTensorProduct(M, N, H) (*function*), 9

H

H ! f (*operation*), 38
HasCharacterTable(A) (*function*), 85
HasComplement(M, S) (*function*), 29
HighestWeightRepresentation(G, w) (*function*), 101
HighestWeightRepresentation(G, w, p) (*function*), 101
Hom Space of Modules for a Soluble Group (*example*), 38
Hom(M, N) (*function*), 37
hom< M -> N | X > (*constructor*), 37
HomMod(M, N) (*function*), 37
Homomorphism(M, N, f) (*function*), 104
Homomorphism(M, N, S) (*function*), 104
Homomorphism(V, W, f) (*function*), 104
Homomorphism(V, W, S) (*function*), 104

I

Id(R) (*function*), 65, 82
Identity(R) (*function*), 65, 82
ImageWithBasis(X, M) (*function*), 20
IndecomposableSummands(M) (*function*), 29
Indicator(x) (*function*), 71
Induced Modules via Tensor Products (*example*), 52
Induction and Restriction for M22 (*example*), 10
Induction(M, G) (*function*), 10
Induction(Q, G) (*function*), 75, 83
Induction(R, G) (*function*), 10
Induction(x, G) (*function*), 75, 83
Inflation(M, h) (*function*), 10
InjectiveHull(M) (*function*), 44
InnerProduct(x, y) (*function*), 71
IntegerRing() ! e (*operation*), 35
Integral Representations of a Permutation (*example*), 90
Intersection(V, W) (*function*), 102
Irreducible Modules for M11 (*example*), 56
Irreducible Modules for PSL(3,4) in Characteristic 3 (*example*), 31
Irreducible Modules for PSL(3,4) over the Rationals (*example*), 32
Irreducible Modules of a Group of Order 416 (*example*), 54
Irreducible Rational Modules for M23 (*example*), 60
IrreducibleModules(G, K : parameters) (*function*), 31, 53, 56
IrreducibleModules(G, K) (*function*), 32
IrreducibleModules(G, Q : parameters) (*function*), 59
IrreducibleModulesBurnside(G, K : parameters) (*function*), 56
IrreducibleModulesSchur(G, K, i: parameters) (*function*), 57
IrreducibleModulesSchur(G, K, L, i: parameters) (*function*), 57
IrreducibleModulesSchur(G, K, L: parameters) (*function*), 57
IrreducibleModulesSchur(G, K: parameters) (*function*), 57
IsAbsolutelyIrreducible(M) (*function*), 23, 24
IsCharacter(x) (*function*), 69
IsConjugate(x, y) (*function*), 71
IsCyclic(R) (*function*), 40
IsDecomposable(M) (*function*), 23, 29
IsDirectSummand(M, S) (*function*), 29
IsEtale(D) (*function*), 96
IsFaithful(x) (*function*), 70

IsFree(M) (*function*), 23
IsGeneralizedCharacter(x) (*function*), 70
IsIrreducible(M) (*function*), 23, 24
IsIrreducible(x) (*function*), 70
IsIsomorphic(M, N) (*function*), 40
IsLeftModule(M) (*function*), 62
IsLinear(x) (*function*), 70
IsMinusOne(x) (*function*), 70
IsModuleHomomorphism(X) (*function*), 38
IsOne(x) (*function*), 70
IsOrthogonalCharacter(chi) (*function*), 78
IsPermutationCharacter(c) (*function*), 70
IsPermutationModule(M) (*function*), 23
IsProjective(M) (*function*), 23
IsReal(x) (*function*), 70
IsRealisableOverSmallerField(M) (*function*), 12
IsRealisableOverSubfield(M, F) (*function*), 12
IsRightModule(M) (*function*), 62
IsSelfDual(M) (*function*), 23
IsSemisimple(M) (*function*), 23, 29
IsSimilar(M, N) (*function*), 40
IsSymplecticCharacter(chi) (*function*), 78
IsTrivial(V) (*function*), 102
IsZero(u) (*function*), 20
IsZero(x) (*function*), 70

J

JacobsonRadical(e) (*function*), 35
JacobsonRadical(M) (*function*), 27

K

k * u (*operation*), 19
Kernel(f) (*function*), 104
Kernel(M) (*function*), 17
Kernel(x) (*function*), 69
KnownIrreducibles(G) (*function*), 65
KnownIrreducibles(R) (*function*), 65

L

L ! i (*operation*), 34
L ! S (*operation*), 35
Left and Right Modules of a Bimodule (*example*), 51
LeftBimodule(M) (*function*), 51
LeftOppositeModule(B) (*function*), 51
LiftCharacter(c, f, G) (*function*), 75, 83
LiftCharacters(T, f, G) (*function*), 75, 83
LinearCharacters(G) (*function*), 66
LowDimensionalModules(G, K, n) (*function*), 49

M

M ! 0 (*operation*), 19
M ! Q (*operation*), 19
m * v (*operation*), 103
M + N (*operation*), 22
M . i (*operation*), 15
M ^ T (*operation*), 12
M meet N (*operation*), 22
MatrixGroup(M) (*function*), 15
MaximalExtension(~M, N) (*procedure*), 48
MaximalExtension(M, N) (*function*), 48
MaximalExtension(M, N, E, r) (*function*), 48
MaximalSubmodules(e) (*function*), 35
MaximalSubmodules(M) (*function*), 27
Meataxe(M) (*function*), 23
Minimal Field of a Q8 Representation (*example*), 14
Minimal Submodules for O5(3) (*example*), 28
MinimalField(M) (*function*), 24
MinimalSubmodule(M) (*function*), 27
MinimalSubmodules(M) (*function*), 27
MinimalSubmodules(M, F) (*function*), 27
MinimalSupermodules(e) (*function*), 35
Minimize(M) (*function*), 13
Minimize(R) (*function*), 13
Module for the Wreath Product A4 wr C3 (*example*), 8
Module from a Conjugation Action (*example*), 4
Module of Dimension 6 over a Finite Field (*example*), 3
Module(A, m) (*function*), 61
Module(e) (*function*), 35
ModuleOverSmallerField(M, F) (*function*), 12
ModulesOverCommonField(M, N) (*function*), 13
ModulesOverSmallerField(Q, F) (*function*), 12
ModuleWithBasis(Q) (*function*), 62
Morphism(e) (*function*), 35
Morphism(M, N) (*function*), 20, 22

N

N eq M (*operation*), 21
N subset M (*operation*), 21
Nagens(M) (*function*), 18
Names(M) (*function*), 102
Natural Module for M11 (*example*), 5
Natural Module of a Matrix Algebra (*example*), 61
Nclasses(R) (*function*), 68
Ngens(M) (*function*), 16, 102
Ngens(V) (*function*), 102
Norm(x) (*function*), 71
NumberOfActionGenerators(M) (*function*), 16, 18
NumberOfClasses(R) (*function*), 68

O

One(R) (*function*), 65, 82
Operations in a Submodule Lattice (*example*), 35
Order(x) (*function*), 71
OrthogonalSymmetrization(chi, p) (*function*), 77

P

Parent(R) (*function*), 68

Parent(u) (*function*), 15
Parent(v) (*function*), 103
Permutation Bimodules (*example*), 52
Permutation Module for M12 (*example*), 6
PermutationBimodule(G,H,m,K) (*function*), 52
PermutationCharacter(G) (*function*), 78
PermutationCharacter(G, H) (*function*), 78
PermutationModule(G, H, K) (*function*), 6
PermutationModule(G, K) (*function*), 6
PermutationModule(G, u) (*function*), 6
PermutationModule(G, V) (*function*), 6
Phi(D, x) (*function*), 96
PhiModule(M) (*function*), 95
PhiModuleElement(x,D) (*function*), 96
PIMBlocks(CM) (*function*), 44
PIMBlocks(I, P) (*function*), 44
PowerMap(R) (*function*), 68
PrincipalCharacter(G) (*function*), 65
Printing a Character Table (*example*), 67
Projective Indecomposables for Alt(8) (*example*), 44
ProjectiveCover(M) (*function*), 44
ProjectiveIndecomposableDimensions(G, K) (*function*), 43
ProjectiveIndecomposableModule(I: parameters) (*function*), 44
ProjectiveIndecomposableModules(G, K: parameters) (*function*), 44
Pullback(V, f, G) (*function*), 101

Q

quo< M | e1, ..., en > (*constructor*), 63
quo< M | S > (*constructor*), 63
quo<M | L> (*constructor*), 22
Quotient of a Permutation Module for L(3,4) (*example*), 22

R

R ! a (*operation*), 65
R ! [a_1, ..., a_k] (*operation*), 64, 82
RadicalSignCharacter(G, d) (*function*), 100
RadicalSignCharacterSinglePrime(G, p) (*function*), 100
Random(L) (*function*), 35
Random(M) (*function*), 19
RandomBaseChange(~D) (*procedure*), 96
Rank(M) (*function*), 102
Rank(V) (*function*), 102
Rational Irreducible Modules for PSL(3,3) (*example*), 59
RationalCharacterTable(G) (*function*), 31, 59, 67
ReduceCharacters(I, C) (*function*), 79
RelativeDecomposition(M, T) (*function*), 29
RemoveIrreducibles(I, C) (*function*), 79
Representation on Homogeneous Polynomials (*example*), 8
Representation(M) (*function*), 17
Representations of a Dihedral Group of Order 20 (*example*), 57
Restriction(M, H) (*function*), 10
Restriction(Q, H) (*function*), 75, 83
Restriction(x, H) (*function*), 75, 83
Rho(G, k, j) (*function*), 100
RightAction(M) (*function*), 15
RightActionGenerator(M, i) (*function*), 18
RightBimodule(M) (*function*), 51
RightModule(B) (*function*), 51
RModule(A) (*function*), 3
RModule(Q) (*function*), 3

S

SaveCharacterTable(T, F) (*procedure*), 67
Schur Indices for the Lyons Group (*example*), 87
Schur(x, k) (*function*), 71
SchurIndex(x) (*function*), 72
SchurIndex(x, F) (*function*), 72
SchurIndex(x, Q) (*function*), 72
SchurIndexGroup(n: parameters) (*function*), 74
SchurIndices(C, s, F) (*function*), 72
SchurIndices(x) (*function*), 72
SchurIndices(x, F) (*function*), 72
SchurIndices(x, Q) (*function*), 72
Sections(G) (*function*), 18
Seminormal and Orthogonal Representations (*example*), 91
SemisimpleDecomposition(D) (*function*), 96
SetVerbose("SubmoduleLattice", i) (*procedure*), 34
Similarity versus Isomorphism of Modules (*example*), 41
Slopes of a Direct Sum of Phi-Modules (*example*), 97
Slopes(D) (*function*), 96
Socle(M) (*function*), 27
SocleFactors(M) (*function*), 27
SocleLayerFactors(M) (*function*), 28
SocleRecursive(M) (*function*), 27
SocleSeries(M) (*function*), 27
Source(M : parameters) (*function*), 49
SpinorNormRepresentation(G, d) (*function*), 100
SpinRepresentation(G, p) (*function*), 100
SSGaloisRepresentation(D) (*function*), 96, 97
SSGaloisRepresentation(E,K,w,P) (*function*), 97
StandardRepresentation(G) (*function*), 100
StructureConstant(G, i, j, k) (*function*), 72
sub< M | e1, ..., en > (*constructor*), 63
sub< M | S > (*constructor*), 63
sub<M | L> (*constructor*), 20
SubalgebraModule(B, M) (*function*), 62
SubCFModule(V, t) (*function*), 100
SubgroupFusion(G, H) (*function*), 75

Submodule of a Permutation Module for L(3,4) (*example*), 20
SubmoduleLattice(M) (*function*), 34
SubmoduleLatticeAbort(M, n) (*function*), 34
Submodules(M) (*function*), 34
Subrepresentation(V, t) (*function*), 100
SummandIsomorphism(M, N) (*function*), 40
Support(u) (*function*), 20
SymmetricCharacter(pa) (*function*), 92
SymmetricCharacterTable(d) (*function*), 92
SymmetricCharacterTable(n) (*function*), 30
SymmetricCharacterValue(pa, pe) (*function*), 92
SymmetricRepresentation(pa, pe) (*function*), 90
SymmetricRepresentation(V, n) (*function*), 101
SymmetricRepresentationOrthogonal(pa, pe) (*function*), 91
SymmetricRepresentationSeminormal(pa, pe) (*function*), 91
SymmetricSquare(M) (*function*), 9
Symmetrization(chi, p) (*function*), 77
Symplectic and Orthogonal Symmetrizations (*example*), 77
SymplecticSymmetrization(chi, p) (*function*), 77
SymSpinor(G, d, k) (*function*), 100

T

Tensor Product Rule for Symmetrizations (*example*), 77
TensorPower(M, n) (*function*), 9
TensorPower(V, d) (*function*), 101
TensorProduct(B1, B2) (*function*), 52
TensorProduct(M, N) (*function*), 9
TensorProduct(V, W) (*function*), 101
Testing for Permutation Characters (*example*), 70
Three-Dimensional Module for PSL(2,7) (*example*), 4
Top(L) (*function*), 35
TrivialModule(G, K) (*function*), 4
TrivialRepresentation(G, R) (*function*), 100
T[i] (*literal*), 70
T[i][j] (*literal*), 70

U

u * a (*operation*), 19
u * g (*operation*), 19
u * k (*operation*), 19
u + v (*operation*), 19
u / k (*operation*), 19
u - v (*operation*), 19
u in M (*operation*), 21
u[i] (*indexed*), 19
u[i] := x (*literal*), 20

V

v + w (*operation*), 103
v @ f (*operation*), 105
v - w (*operation*), 103
v ^ a (*operation*), 61
v ^ w (*operation*), 103
V eq W (*operation*), 102
v eq w (*operation*), 104
v in V (*operation*), 104
V meet W (*operation*), 102
Vertex(M : parameters) (*function*), 49
Vertices and Sources in M22.2 (*example*), 49

W

w @@ f (*operation*), 105
Weights(V) (*function*), 97
WriteGModuleOver(M, K) (*function*), 13
WriteOverSmallerField(M, F) (*function*), 12
WriteRepresentationOver(R, K) (*function*), 13

X

x * y (*operation*), 69, 83
x + y (*operation*), 69, 83
x - y (*operation*), 69, 83
x ^ g (*operation*), 71
x ^ H (*operation*), 71
x ^ j (*operation*), 69, 83
x eq y (*operation*), 69
x in y (*operation*), 69
x ne y (*operation*), 69
x notin y (*operation*), 69
x(g) (*literal*), 71
x[i] (*literal*), 71

Z

Zero(M) (*function*), 19
Zero(R) (*function*), 65, 82