April 29, 1997

Contents

Chapter 1

CHEVIE Version 3 – a short introduction

CHEVIE is a joint project of Meinolf Geck, Gerhard Hiss, Frank Lübeck, Gunter Malle, Jean Michel, and Götz Pfeiffer. The GAP–part of CHEVIE is a package, written entirely in the GAP language, which consists of

- functions implementing algorithms for finite reflection groups and their root systems, braid groups, and Iwahori-Hecke algebras, including normal forms for the elements in these objects and, for example, special functions for computing Kazhdan-Lusztig polynomials and lefts cells;
- library files containing pre-stored information about conjugacy classes of finite Coxeter groups, fake degrees and generic degrees of the irreducible characters, and character tables of finite Coxeter groups and the associated Iwahori-Hecke algebras.

The files containing these programs and data are built up in a similar way as the files in GAP. In particular, each file contains a copyright notice and the name(s) of the CHEVIE authors who wrote that file.

This package replaces the former `weyl` package. A number of functions have got new names (mathematically more correct, we hope). For example, one can now work systematically with non-crystallographic Coxeter groups, and so the former function `Weyl` has become `CoxeterGroup`. Elements in these groups are systematically permutations on the underlying root system, and reduced expressions in the generators of the group will be called `CoxeterWords`. For the convenience of the user we have provided a file which contains just assignments of the form `Weyl:=CoxeterGroup` etc., so that many functions written with the old system should still work. This file can be read into the system using the command `ReadChv("compatib")`, but we do not guarantee that this file will survive in future releases. The package itself is loaded using the command `RequirePackage("chevie")` (see ??).

The overall logical structure of this package is as follows. The basic input datum is a Cartan matrix associated with a finite reduced root system in some Euclidean space. The function `CoxeterGroup` computes from such a matrix a record which contains basic information and upon which all further applications are based. This is described in detail in chapter 2. There are extensions to this basic construction allowing, more generally, inputs like root data and automorphisms acting on the ingredients of such root data. This is described in more detail in chapter 11.

But it is important to keep in mind that the whole system is structured bottom-up. This means that, if one is only interested in the finite Coxeter groups themselves then one can just stick to the simplest form of the function `CoxeterGroup` and one does not have to worry about any of the more elaborate concepts which can be build upon this. Any higher order application just adds new components to the original record and increases its functionality.

A similar remark applies to braid groups and Iwahori–Hecke algebras: they are constructed by applying the functions `Braid` and `Hecke` to a Coxeter group record which had already been constructed before using the function `CoxeterGroup`. Again, all the functionality of the original record remains valid.

The files containing the most basic functions implementing these structures are `coxeter.g`, `braid.g` and `hecke.g`. There are more files containing functions for special purposes like `kl.g` (for Kazhdan-Lusztig polynomials).

The record computed by `CoxeterGroup` is, in the first place, a permutation group record in GAP. Thus, all GAP functions for dealing with finite groups and, in particular, permutation groups can be applied to such a record, like `Elements`, `Size`, `CharTable` to mention but a few. These are generic GAP functions in the usual sense that they first look at the operations record of `CoxeterGroup` to see which algorithm has to be taken. Another important feature of the design of the CHEVIE package is that many of these functions admit more efficient versions by taking into account the particular structure of finite Coxeter groups.

Many objects constructed in or associated with finite Coxeter groups admit some canonical labeling which carries additional information. It is precisely this labeling which is very often important to know for the application of results on Coxeter groups in Lie theory or related areas.

For example, the generic GAP function `ConjugacyClasses` applied to a Coxeter group record does not invoke the general algorithm for computing conjugacy classes of permutation groups in GAP, but first decomposes the given Coxeter group into irreducible components, and then reads canonical representatives of minimal length in the various classes of these irreducible components from library files. These canonical representatives also come with some additional information, for example Carter's admissible diagrams (see 3.14). In a similar way, the function `CharTable` does not invoke the Dixon–Schneider algorithm but proceeds in a similar way as described above. Moreover, the resulting character table comes with additional labelings of the classes and characters, like partitions of n in the case of the symmetric group $\mathfrak{S}_n$, i.e., the Coxeter group of type A_{n-1} (see 4.10).

Thus, roughly 2/3 of the total memory space required by the CHEVIE files is occupied by the files containing the basic information about the finite irreducible Coxeter groups. These files are called `weyla.g`, `weylb.g` etc. up to the biggest file `weyle8.g` whose size is about 500 KBytes. These data files are structured in a uniform manner so that any piece of information can be extracted separately from them. (For example, it is not necessary to first compute the character table in order to have labels for the characters and classes.)

The conventions that we use about normal forms of elements, labeling of classes and characters for the individual types are explained in detail in the introductions to chapters 2 and 4.

Several computations in the literature concerning the irreducible characters of finite Coxeter groups and Iwahori–Hecke algebras can now be checked or re-computed by anyone who is willing to use GAP and CHEVIE. Re-doing such computations and comparing with existing tables has sometimes lead to the discovery of bugs in the programs or to misprints in the literature. We believe that having the possibility of repeating such computations and experimenting with the results has increased the reliability of the data (and the programs).

For example, it is now a trivial matter to re-compute the tables of induce/restrict matrices (with the appropriate labeling of the characters) for exceptional finite Weyl groups (see Section 5.1). These matrices have various applications in the representation theory of finite reductive groups, see Lusztig's book [Lus85, Ch.4].

We ourselves have used these programs to prove results about the existence of elements with special properties in the conjugacy classes of finite Coxeter groups (see [GP93], [GM97]), and to compute character tables of Iwahori–Hecke algebras of exceptional type (see [Gec94], [GM97]). For a survey, see also [GHL+96].

Of course, our hope is that more applications will be added in the future! For contributions to CHEVIE from outside (or one or several among us) we have created a directory `contr` in which the corresponding files are distributed with CHEVIE. However, they do remain under the authorship and the responsibility of their authors. Files from that directory can be read into GAP using the command `ReadChv("contr/filename")`. At present, the directory `contr` contains the following files:

`murphy` by A. Mathas; it contains programs which allow calculations with the Murphy basis of the Hecke algebra of type A.

`minrep` by M. Geck and G. Pfeiffer; it contains programs (used in [GP93]) for computing representatives of minimal length in the conjugacy classes of finite Coxeter groups.

`brbase` by M. Geck and S. Kim; it contains programs for computing bi-grassmannians and the base for the Bruhat–Chevalley order on finite Coxeter groups (see [GK96]).

`braidsup` by J. Michel; it contains some supplementary programs for working with braids.

`chargood` by M. Geck and J. Michel; it contains functions (used in [GM97]) implementing algorithms to compute character tables of Iwahori–Hecke algebras, especially that of type E_8.

Finally, it should be mentioned that there is also a MAPLE-part of CHEVIE which contains generic character tables of finite groups of Lie type and tables of Green functions. The conventions about data related to the associated finite Weyl groups are compatible with those in the present package. It is planned that, in the not too far future, the MAPLE-part will be re-written in GAP.

Acknowledgments. We wish to thank the Aachen GAP team for general support over the last years.

We also gratefully acknowledge financial support by the DFG in the framework of the Forschungsschwerpunkt "Algorithmische Zahlentheorie und Algebra" since 1992.

We are indebted to Andrew Mathas for contributing a package with functions for calculating the various Kazhdan-Lusztig bases in Iwahori–Hecke algebras. (These functions have now become a part of the file `kl.g`.)

St. Andrews, Paris and Heidelberg, December 1996

Chapter 2

Root systems and finite Coxeter groups

In this chapter we describe functions for dealing with root systems and finite Coxeter groups.

A suitable reference for the general theory is, for example, the volume of Bourbaki [Bou68]; an English reference is the book of Humphreys [Hum90].

A **Coxeter group** is a group defined by a presentation of the form

$$\left\langle s_1, \ldots, s_n \mid (s_i s_j)^{m(i,j)} = 1 \quad \text{for all } i, j \right\rangle$$

for some integers $m(i,j)$, where $m(i,j) > 1$ for $i \neq j$ and $m(i,i) = 1$ for all i. The matrix $\{m(i,j)\}_{i,j}$ is called the **Coxeter matrix**; the set of Coxeter matrices such that the defined group is finite have been completely classified. A Coxeter group has a natural representation on a real vector space V of dimension the number of generators, its **reflection representation**, where the s_i act as reflections (a **reflection** in a vector space V is an element of $\mathrm{GL}(V)$ of order 2 which leaves fixed a hyperplane). It turns out that finite Coxeter groups are the same as finite real reflection groups, i.e., finite groups generated by reflections in a real vector space. The set of reflecting hyperplanes of a finite Coxeter group is in fact related to the notion of a root system, and this carries slightly more information than just the set of integers $m(i,j)$ used in the definition of Coxeter groups (by taking into account relative lengths of the roots, see below). It is this underlying geometric structure by which Coxeter groups appear in various areas of mathematics such as Lie algebras and linear algebraic groups. In GAP, the objects we will actually deal with are the Coxeter groups, together with their action on a root system.

Let us now give the precise definitions. Let V be a vector space, $V^\vee$ its dual. We will denote by $(\ ,\ )$ the natural pairing between $V^\vee$ and V. A **root system** in V is a finite set of vectors R (the **roots**), together with a map $r \mapsto r^\vee$ from R to a subset $R^\vee$ of $V^\vee$ (the **coroots**) such that:

For any $r \in R$, we have $(r^\vee, r) = 2$, so that the map $V \to V$, $x \mapsto x - (r^\vee, x)r$ defines a reflection in V (that we will call the **reflection with root** r), and this reflection stabilizes R. If R does not span V we also have to impose the condition that the map $V^\vee \to V^\vee$, $y \mapsto y - (y, r)r^\vee$ stabilizes $R^\vee$.

We will only consider **reduced** root systems, i.e., such that the only elements of R colinear with a root r are r and $-r$.

A root system R is called **crystallographic** if $(r^\vee, r)$ is an integer, for any $r \in R, r^\vee \in R^\vee$.

The dimension of the subspace V_R of V spanned by R will be called the **semi-simple rank** of R.

Remark. For the study of Coxeter groups it would be sufficient to consider root systems as certain subsets of Euclidean spaces which contain a basis of that space. Our definition is motivated by the notion of root data, which allow to describe connected reductive algebraic groups over algebraically closed fields (see for example [Spr81, Ch.9]).

The subgroup $W = W(R)$ of $\mathrm{GL}(V)$ generated by the reflections with roots in R is a finite Coxeter group given by a presentation as above. (Below, we will describe explicitly how to obtain the set of generators from the root system.) We call W a **crystallographic Coxeter** group (or a **Weyl group**) if the underlying root system is crystallographic. This condition is equivalent to the property that the reflection representation of W is defined over the rational numbers. Weyl groups can be characterized amongst finite Coxeter groups by the fact that all numbers $m(i,j)$ are in $\{2,3,4,6\}$. It turns out that all other finite-dimensional (complex) representations of a Weyl group W can also be realized over the rational numbers.

We identify V with $V^\vee$ by choosing a W-invariant bilinear form $(\ ;\ )$; then we have $r^\vee = 2r/(r;r)$. A root system R is irreducible if it is not the union of two orthogonal subsets. If R is reducible then the corresponding Coxeter group is the direct product of the Coxeter groups associated with the irreducible components of R. The irreducible root systems, and also the finite irreducible Coxeter groups, are classified by the following list of Dynkin diagrams. The labeling of the nodes is exactly the same labeling as in the function `CartanMat` described below.

```
        1   2   3           n                1   2   3           n
A_n     o---o---o-- . . . --o          B_n   o=<=o---o-- . . . --o

      1 o
         \   4              n                1   2   3           n
D_n     3 o---o---   . . . --o          C_n   o=>=o---o-- . . . --o
         /
      2 o

        1  2            1   2   3   4          1   3   4   5   6
G_2     0->-0       F_4  o---o=>=o---o      E_6  o---o---o---o---o
          6                                             |
                                                        o 2

        1   3   4   5   6   7              1   3   4   5   6   7   8
E_7     o---o---o---o---o---o          E_8  o---o---o---o---o---o---o
                |                                   |
                o 2                                 o 2

            1   2               1   2   3          1   2   3   4
I_2(m)      o---o          H_3  o---o---o     H_4  o---o---o---o
              m                   5                   4
```

These diagrams encode the presentations for Coxeter groups as follows: the vertices represent the s_i; an edge is drawn between s_i and s_j if $m(i,j) > 2$; the edge is represented as a single bond if $m(i,j) = 3$, a double bond if $m(i,j) = 4$, a triple bond if $m(i,j) = 6$ and as a single bond with the value $m(i,j)$ written above if $m(i,j) \notin \{2,3,4,6\}$. (We see that we can ignore the arrows indicating relative root lengths; thus, the diagrams of type B_n and C_n lead to identical presentations for Coxeter groups.) The last three diagrams correspond to non-crystallographic groups, excepted for the special cases $I_2(3) = A_2$, $I_2(4) = B_2$ and $I_2(6) = G_2$.

Let us now describe how the root systems are encoded in these diagrams. Let R be a root system in V. Then we can choose a linear form on V which vanishes on no element of R. According to the sign of the value of this linear form on a root $r \in R$ we call r **positive** or **negative**. Then there exists a unique subset of the set of positive roots, called the set of **fundamental roots**, such that any positive root is a linear combination with non-negative coefficients of fundamental roots. It can be shown that any two sets of fundamentals roots (corresponding to different choices of linear forms as above) can be transformed into each other by a unique element of $W(R)$, hence the relative lengths and the angles between fundamental roots are independent of any choice. Hence, if $\{r_1, \ldots, r_n\}$ is a set of fundamental roots and if we define the **Cartan matrix** as being the n times n matrix $C = ((r_i^\vee, r_j)_{ij})$ then this matrix is in fact unique up to simultaneous permutation of rows and columns. It is precisely this matrix which is encoded in a Dynkin diagram, as follows.

The indices for the rows of C label the nodes of the diagram. The edges, for $i \neq j$, are given as follows. If C_{ij} and C_{ji} are integers such that $|C_{ij}| \geq |C_{ji}|$ the vertices are connected by $|C_{ij}|$ lines, and if $|C_{ij}| > 1$ then we put an additional arrow on the lines pointing towards the node with label i. In all other cases, we simply put a single line equipped with the unique integer $p_{ij} \geq 1$ such that $C_{ij}C_{ji} = \cos^2(\pi/p_{ij})$.

It is now important to note that, conversely, the whole root system can be recovered from the set of fundamental roots. The reflections in $W(R)$ corresponding to the fundamental roots are called **fundamental** (or **simple**) reflections. They are precisely the generators for which the above Dynkin diagrams encode the defining relations of $W(R)$. It can be shown that for each $r \in R$ there exist fundamental reflections $s_{i_1}, \ldots, s_{i_m}$ such that $s_{i_1} \cdots s_{i_m}(r)$ is a fundamental root. Thus, all of R is obtained by applying repeatedly simple reflections to a set of roots (starting with the set of fundamental roots). The restriction of the simple reflections to V_R is determined by the Cartan matrix, so R is determined by the Cartan matrix and the set of fundamental roots.

In GAP the Cartan matrix corresponding to one of the above irreducible root systems (with the specified labeling) is returned by the command `CartanMat` which takes as input a string giving the type (e.g., `"A"`, `"B"`, ..., `"I"`) and a positive integer giving the rank. For type $I_2(m)$, we give as a third argument the integer m. This function returns a matrix (i.e., a list of lists in GAP) with entries in $\mathbb{Z}$ or in a cyclotomic extension of the rationals. Given two Cartan matrices, their matrix direct sum (corresponding to the orthogonal direct sum of the root systems) can be produced by the function `DirectSumMat`.

The function `CoxeterGroup` takes as input some data which determine the roots and the coroots and produces a GAP permutation group record, where the Coxeter group is represented by its faithful action on the root system R, with additional components containing basic information about R (a system of fundamental roots etc...).

The function `CoxeterGroup` has several forms; in the first form, it is assumed that the simple roots are the basis of V (the matrix of the coroots expressed in the dual basis of $V^\vee$ is then equal to the Cartan matrix); the argument is the Cartan matrix of the root system (irreducible or not, and with any ordering of the simple roots).

If one only wants to work with Cartan matrices with a labeling as specified by the above list, the function call can be simplified. Instead of `CoxeterGroup( CartanMat("D", 4 ) )` the following is also possible.

```
gap> W := CoxeterGroup( "D", 4 );          # Coxeter group of type D4
CoxeterGroup("D", 4)
gap> PrintArray( W.cartan );
[ [   2,   0,  -1,   0 ],
  [   0,   2,  -1,   0 ],
  [  -1,  -1,   2,  -1 ],
  [   0,   0,  -1,   2 ] ]
```

(The matrix printed is the **Cartan matrix**.)

Also, the Coxeter group record associated to a direct sum of irreducible root systems with the above standard labeling can be obtained by listing the types of the irreducible components:

```
gap> W := CoxeterGroup( "A", 2, "B", 2 );; PrintArray( W.cartan );
[ [   2,  -1,   0,   0 ],
  [  -1,   2,   0,   0 ],
  [   0,   0,   2,  -2 ],
  [   0,   0,  -1,   2 ] ]
```

(The same record is constructed by applying `CoxeterGroup` to the matrix `CartanMat("A", 2, "B", 2)` or to `DirectSumMat(CartanMat("A", 2), CartanMat("B", 2))`.)

In the second more general form, one gives as argument to `CoxeterGroup` two matrices, one whose lines are the roots expressed in a basis of V, and the second whose lines are the coroots expressed in the corresponding dual basis of $V^\vee$. In this form, the roots need not generate V.

```
gap> W := CoxeterGroup( [ [ -1, 1, 0], [ 0, -1, 1 ] ],
>                       [ [ -1, 1, 0], [ 0, -1, 1 ] ] );
CoxeterGroup([ [ -1, 1, 0 ], [ 0, -1, 1 ] ],
[ [ -1, 1, 0 ], [ 0, -1, 1 ] ])
gap> MatXPerm( W, W.generators[1] );
[ [ 0, 1, 0 ], [ 1, 0, 0 ], [ 0, 0, 1 ] ]
```

(here we have represented the symmetric group on 3 letters as the permutation of the basis vectors of V — the semi-simple rank is 2).

The definition of root systems implies that every $w \in W$ induces a permutation of the elements in R, and that the corresponding permutation representation of W on R is faithful. If we label the positive roots by `[1 .. N]`, and the negative roots by `[N+1 .. 2*N]`, then we can represent each fundamental reflection by the permutation of `[ 1 .. 2*N ]` which it induces on the root vectors. The representation of W in GAP is as the permutation group defined by this faithful permutation representation. All this is done by the command `CoxeterGroup` which produces a permutation group record containing the relevant information (see the precise description in Section 2.5). This record is all that the following programs and commands need. See the following chapter for more details on how to work with the elements in W and different representations for them (permutations, reduced expressions, matrices).

We close this informal introduction with some remarks.

• Since, in the first place, the Coxeter group record is a group record with the component `isPermGroup` set to `true`, all GAP functions defined for permutation groups work for Coxeter groups, but sometimes there are improvements, exploiting the particular nature of these groups.

• There is a function `InfoChevie` which is set equal to the GAP function `Ignore` when you load CHEVIE. If you redefine it by `InfoChevie:=Print;` then the CHEVIE functions will print some additional information in the course of their computations.

• The user should observe limitations on storage for working with these programs, e.g., the command `Elements` applied to a Weyl group of type E_8 will try to compute all elements as words in the fundamental reflections. Every computer will run out of memory!

2.1 CartanMat

`CartanMat(` *type* `,` *n* `)`

returns the Cartan matrix of Dynkin type *type* and rank n. Admissible types are the strings `"A"`, `"B"`, `"C"`, `"D"`, `"E"`, `"F"`, `"G"`, `"H"`, `"I"`.

```
gap> C := CartanMat( "F", 4 );;
gap> PrintArray( C );
[ [   2,  -1,   0,   0 ],
  [  -1,   2,  -1,   0 ],
  [   0,  -2,   2,  -1 ],
  [   0,   0,  -1,   2 ] ]
```

For type $I_2(m)$, which is in fact an infinity of types depending on the number m, a third argument is needed specifying the integer m so the syntax is in fact `CartanMat( "I", 2,` m `)`:

```
gap> CartanMat( "I", 2, 5 );
[ [ 2, E(5)^2+E(5)^3 ], [ E(5)^2+E(5)^3, 2 ] ]
```

`CartanMat(` *type1* `,` *n1* `, ... ,` *typek* `,` *nk* `)`

returns the direct sum of `CartanMat(` *type1* `,` *n1* `)`, ..., `CartanMat(` *typek* `,` *nk* `)`. One can use as argument a computed list of types by `ApplyFunc( CartanMat, [` *type1* `,` *n1* `, ... ,` *typek* `,` *nk* `] )`.

This function requires the package "chevie" (see ??).

2.2 CartanType

`CartanType(` C `)`

`CartanType(` D `)`

returns the type of the Cartan Matrix C. The result is a list each element of which describes an irreducible component of C, as pair `[type,indices]` where `type` is the type (`"A"`, `"B"`, `"D"`, etc...) of the component and `indices` the indices in C where it sits, so that

`C{indices}{indices} = CartanMat( type, Length( indices ) )`

A triple `[type, indices, `m`]` is actually returned for a component of type $I_2(m)$. The `indices` are arranged in the standard order in which we give Dynkin diagrams.

```
gap> C := [ [ 2, 0, -1 ], [ 0, 2, 0 ], [ -1, 0, 2 ] ];;
gap> CartanType( C );
[ [ "A", [ 1, 3 ] ], [ "A", [ 2 ] ] ]
```

The argument to `CartanType` can also be a domain (i.e., D should be a record with a field `operations.CartanType`, and that function is then called with D as argument — this is used for Coxeter groups and Coxeter cosets).

This function requires the package "chevie"(see ??).

2.3 CartanName

`CartanName(` *type* `)`

`CartanName(` D `)`

takes as argument a type *type* as returned by `CartanType`. Returns the name of the root system with that type, which is the concatenation of the names of its irreducible components, with `x` added in between.

```
gap> C := [ [ 2, 0, -1 ], [ 0, 2, 0 ], [ -1, 0, 2 ] ];;
gap> CartanName( CartanType( C ) );
"A2xA1"
gap> CartanName( CartanType( CartanMat( "I", 2, 7 ) ) );
"I2(7)"
```

The argument to `CartanName` can also be a domain (i.e., D should be a record with a field `operations.CartanName`, and that function is then called with D as argument — this is used for Coxeter groups and Coxeter cosets).

This function requires the package "chevie"(see ??).

2.4 PrintDynkinDiagram

`PrintDynkinDiagram(` *type* `)`

`PrintDynkinDiagram(` D `)`

This is a purely descriptive routine, which, by printing the Dynkin diagram of a type *type* (the result of `CartanType`) shows how the generators of the corresponding group are labeled on its Dynkin diagram.

```
gap> C := [ [ 2, 0, -1 ], [ 0, 2, 0 ], [ -1, 0, 2 ] ];;
gap> t := CartanType( C );
[ [ "A", [ 1, 3 ] ], [ "A", [ 2 ] ] ]
gap> PrintDynkinDiagram( t );
A2    1 - 3
A1    2
gap> PrintDynkinDiagram( CartanType( CartanMat( "E", 8) ) );
E8          2
            |
```

```
1 - 3 - 4 - 5 - 6 - 7 - 8
```

The argument to `PrintDynkinDiagram` can also be a domain (i.e., D should be a record with a field `operations.PrintDynkinDiagram`, and that function is then called with D as argument — this is used for Coxeter groups and Coxeter cosets).

This function requires the package "chevie"(see **??**).

2.5 CoxeterGroup

`CoxeterGroup(` *simpleRoots*`,` *simpleCoroots*`[,` *omega*`] )`

`CoxeterGroup(` C`[, "sc"][,` *omega*`] )`

`CoxeterGroup(` *type1*`,` *n1*`, ... ,` *typek*`,` *nk*`[, "sc"][,` *omega*`] )`

`CoxeterGroup(` *rec* `)`

This function returns a permutation group record containing the basic information about the Coxeter group and the root system determined by its arguments. In the first form a set of roots is given explicitly as the lines of the matrix *simpleRoots*, representing vectors in a vector space V, as well as a set of coroots as the lines of the matrix *simpleCoroots* expressed in the dual basis of $V^\vee$. The product C=*simpleCoroots*`*TransposedMat(`*simpleRoots*`)` must be a valid Cartan matrix. The dimension of V can be greater than `Length(`C`)`. The length of C is called the **semisimple rank** of the Coxeter datum, while the dimension of V is called its **rank**.

In the second form C is a Cartan matrix, and the call `CoxeterGroup(`C`)` is equivalent to

`CoxeterGroup(IdentityMat(Length(`C`)),`C`)`.

In this case, the root system is embedded in the lattice of integral vectors of V like the root system of an adjoint algebraic reductive group in the lattice of characters of a maximal torus.

If the optional `"sc"` argument is given, the situation is reversed: the simple coroots are given by the identity matrix, and the simple roots by the transposed of C (this corresponds to the embedding of the root system in the lattice of characters of a maximal torus in a **simply connected algebraic group**).

The third form is equivalent to

`CoxeterGroup(CartanMat(`*type1*`,` *n1*`, ...,` *typek*`,` *nk*`) [, "sc"][,` *omega*`])`.

The resulting record, that we will call a **Coxeter datum**, has additional entries describing various information on the root system and Coxeter group that we describe below.

The argument *omega* in one of the first three forms can be used to specify a group of automorphisms of the Coxeter datum, that is, a group of invertible linear transformations of V which preserve the set of roots and whose adjoint maps preserve the set of coroots. When the rank is equal to the semisimple rank (we then say that the Coxeter datum is semisimple), this can be given as a permutation group (on the roots). Otherwise it must be given as a matrix group.

The last form takes as an argument a record which has a field `coxeter` and returns the value of this field. This is used to return the Coxeter group of objects derived from Coxeter groups, such as Coxeter cosets, Hecke algebras and braid elements.

We document the following entries in a Coxeter datum record which are guaranteed to remain present in future versions of the package. Other undocumented entries should not be relied upon, they may change without notice.

`isCoxeterGroup`, `isDomain`, `isGroup`, `isPermGroup`, `isFinite`
: `true`

`cartan`
: the Cartan matrix C

`simpleRoots`
: the matrix of simple roots

`simpleCoroots`
: the matrix of simple coroots

`semisimpleRank`
: the length of C

`rank`
: the length of `TransposedMat(.simpleRoots)`

`N`
: the number of positive roots

`roots`
: the root vectors, given as linear combinations of fundamental roots (note that in a former version of the package only the positive roots were stored). The first `N` roots are positive, the next `N` are the corresponding negative roots. Moreover, the first `semisimpleRank` roots are the fundamental roots corresponding to the rows of C. The positive roots are ordered by increasing height.

`coroots`
: the same information for the coroots. The coroot corresponding to a given root is in the same relative position in the list of coroots as the root in the list of roots.

`rootLengths`
: the vector of length of roots. The shortest roots in an irreducible subsystem are given the length 1, the others length 2 (or 3 in type G_2).

`orbitRepresentative`
: this is a list of same length as `roots`, which for each root, gives the smallest index of a root in the same W-orbit.

`orbitRepresentativeElements`
: a list of same length as `roots`, which for the i-th root, gives an element w of W of minimal length such that `i=orbitRepresentative[i]^w`.

`matgens`
: the matrices (in row convention) of the simple reflections of the Coxeter group with respect to the basis consisting of the fundamental root vectors.

`generators`
: the generators as permutations of the root vectors. They are given in the same order as the first `semisimpleRank` roots.

`omega`
: the value of the argument *omega* if it has been specified. Otherwise unbound.

```
gap> W := CoxeterGroup( "A", 4 );;
gap> PrintArray( W.cartan );
[ [   2,  -1,   0,   0 ],
  [  -1,   2,  -1,   0 ],
  [   0,  -1,   2,  -1 ],
  [   0,   0,  -1,   2 ] ]
gap> W.matgens;
[ [ [ -1, 0, 0, 0 ], [ 1, 1, 0, 0 ], [ 0, 0, 1, 0 ], [ 0, 0, 0, 1 ] ],
  [ [ 1, 1, 0, 0 ], [ 0, -1, 0, 0 ], [ 0, 1, 1, 0 ], [ 0, 0, 0, 1 ] ],
  [ [ 1, 0, 0, 0 ], [ 0, 1, 1, 0 ], [ 0, 0, -1, 0 ], [ 0, 0, 1, 1 ] ],
  [ [ 1, 0, 0, 0 ], [ 0, 1, 0, 0 ], [ 0, 0, 1, 1 ], [ 0, 0, 0, -1 ] ]
 ]
gap> W.roots;
[ [ 1, 0, 0, 0 ], [ 0, 1, 0, 0 ], [ 0, 0, 1, 0 ], [ 0, 0, 0, 1 ],
  [ 1, 1, 0, 0 ], [ 0, 1, 1, 0 ], [ 0, 0, 1, 1 ], [ 1, 1, 1, 0 ],
  [ 0, 1, 1, 1 ], [ 1, 1, 1, 1 ], [ -1, 0, 0, 0 ], [ 0, -1, 0, 0 ],
  [ 0, 0, -1, 0 ], [ 0, 0, 0, -1 ], [ -1, -1, 0, 0 ],
  [ 0, -1, -1, 0 ], [ 0, 0, -1, -1 ], [ -1, -1, -1, 0 ],
  [ 0, -1, -1, -1 ], [ -1, -1, -1, -1 ] ]
```

This function requires the package "chevie"(see ??).

2.6 Operations and functions for Coxeter groups

All permutation group operations are defined on Coxeter groups. However, the following operations and functions have been rewritten or added, respectively, to take advantage of the particular structure of real reflection groups:

`=`
: Two Coxeter data are equal if they are equal as permutation groups and the fields `simpleRoots` and `simpleCoroots` agree (independently of the value of any other bound fields). Also the fields .omega should be equal (if the field is absent — `omega` not specified — this is considered specifying the trivial group).

`Print`
: prints a Coxeter group in a form that can be input back in GAP as a Coxeter group.

`Size`
: uses the classification of Coxeter groups to work faster (specifically, uses the function `ReflectionDegrees`).

`Elements`
: returns the set of elements. They are computed using 3.11. (Note that in an earlier version of the package the elements were sorted by length. You can still get such a list by `Concatenation( List( [1..W.N], i -> CoxeterElementsLength(W, i)))`)

`CartanType`
: returns `CartanType` of the Cartan matrix.

`PrintDynkinDiagram`
: prints the Dynkin diagram corresponding to the Cartan matrix.

`CartanName`
: returns the `CartanName` of the `CartanType`.

`ConjugacyClasses`
: Uses classification of Coxeter groups to work faster, and the resulting list is given in the same order as the result of `ChevieClassInfo` (see 3.14).

`CharTable`
: Uses the classification of Coxeter groups to work faster, and the result has better labeling than the default (see Chapter 4).

`ChevieClassInfo`
: Is part of the Coxeter groups operations record in order to have versions for Coxeter groups and Coxeter cosets. This function returns additional information on the classes which is contained in the character table, but this function returns it without first computing `CharTable(`W`)`. See the explicit description in 3.14.

`CharParams`
: Is part of the Coxeter groups operations record in order to have versions for Coxeter groups and Coxeter cosets. This function returns the list of parameters for the irreducible characters of W: partitions for type `A`, double partitions for type `B`, etc... This is used by functions which return information for individual characters, like `FakeDegree`, `SchurElement`, etc...

`CharName`
: Is part of the Coxeter groups operations record in order to have versions for Coxeter groups and Coxeter cosets. This function takes as argument a parameter for a character (see `CharParams`) and returns a string which is used to label the character is various displayed tables.

`ChevieCharInfo`
: This function returns additional information on the irreducible characters, see 4.10 for more details.

`PositionClass`, `ClassInvariants`, `FusionConjugacyClasses`
: Use the classification of Coxeter groups to work faster. `PositionClass(W,x)` returns the index of the conjugacy class of x in the list of the classes of W, see 13.3.

These functions require the package "chevie"(see ??).

Chapter 3

Elements in finite Coxeter groups

Let (W, S) be a finite Coxeter system as in the previous chapter. For a given element $w \in W$, the **length** $l(w)$ is defined to be the smallest integer k such that w can be written as a product of k fundamental reflections. Such an expression of shortest possible length will be called a **reduced word** for w. A user might be interested to think of the elements of W as such words in the generating fundamental reflections. For these programs, we represent a word simply as a list of integers corresponding to the fundamental roots, e.g., `[]` is the identity element, and `[1]`, `[2]`, etc. represent the reflection along the first, the second etc. fundamental root vector. For computational purposes, it might be better to use the permutation of an element w on the root vectors. The functions `CoxeterWord` and `PermCoxeterWord` will do the conversion of one form into the other.

```
gap> W := CoxeterGroup( "D", 4 );;
gap> p := PermCoxeterWord( W, [ 1, 3, 2, 1, 3 ] );
( 1,14,13, 2)( 3,17, 8,18)( 4,12)( 5,20, 6,15)( 7,10,11, 9)(16,24)
(19,22,23,21)
gap> CoxeterWord( W, p );
[ 1, 3, 1, 2, 3 ]
```

We notice that the word we started with and the one that we ended up with, are not the same. But of course, they represent the same element of W. The reason for this difference is that the function `CoxeterWord` always computes a reduced word which is the lexicographically smallest among all possible expressions of an element of W as a word in the fundamental reflections! The function `ReducedCoxeterWord` does the same but with an arbitrary word as input (and not with a permutation). In particular, the element used in the above example has length 5. Sometimes, it is not necessary to compute a reduced word for an element w and one is only interested in the length $l(w)$; this can be computed very effectively from the permutation, since it is also given by the number of positive roots mapped to negative roots by w, i.e., by the number of $i \in \{1, \ldots, N\}$ such that $i^w > N$. This is what does the function `CoxeterLength` in the following example where we also show how to compute the unique element of maximal length in W.

```
gap> LongestCoxeterWord( W );   # the (unique) longest element in W
```

```
[ 1, 2, 3, 1, 2, 3, 4, 3, 1, 2, 3, 4 ]
gap> w0 := LongestCoxeterElement( W ); # = PermCoxeterWord( W, last )
( 1,13)( 2,14)( 3,15)( 4,16)( 5,17)( 6,18)( 7,19)( 8,20)( 9,21)(10,22)
(11,23)(12,24)
gap> CoxeterLength( W, w0 );
12
```

There is another way of computing a reduced word which is more canonical for some purposes. For any set I of generators, let w_I be the unique element of maximal length which can be made in the generators I. (Note that this element is an involution.) Now take any $w \in W$ and compute the set $L(w)$ of all generators s such that $l(sw) < l(w)$. In GAP this is done by the function `LeftDescentSet`. Brieskorn [Bri71] has noticed that $w_{L(w)}$ divides w, in the sense that $l(w_{L(w)}) + l(w_{L(w)} w) = l(w)$. We can now divide w by $w_{L(w)}$ and continue this process with the quotient. In this way, we obtain a reduced expression $w = w_{L_1} \cdots w_{L_r}$ where $L_i = L(w_{L_i} \cdots w_{L_r})$ for all i, which we call the **Brieskorn normal form** of w (and where we use the lexicographically smallest expression for each w_{L_i}). The CHEVIE package will use this form if you set `CHEVIE.BrieskornNormalForm:= true;`. When you load CHEVIE this variable is initialized with `false` (see also 3.2).

We give an example of some other commands:

```
gap> List( Reflections( W ), i -> CoxeterWord( W, i ) );
[ [ 1 ], [ 2 ], [ 3 ], [ 4 ], [ 1, 3, 1 ], [ 2, 3, 2 ], [ 3, 4, 3 ],
  [ 1, 2, 3, 1, 2 ], [ 1, 3, 4, 3, 1 ], [ 2, 3, 4, 3, 2 ],
  [ 1, 2, 3, 4, 3, 1, 2 ], [ 3, 1, 2, 3, 4, 3, 1, 2, 3 ], [ 1 ],
  [ 2 ], [ 3 ], [ 4 ], [ 1, 3, 1 ], [ 2, 3, 2 ], [ 3, 4, 3 ],
  [ 1, 2, 3, 1, 2 ], [ 1, 3, 4, 3, 1 ], [ 2, 3, 4, 3, 2 ],
  [ 1, 2, 3, 4, 3, 1, 2 ], [ 3, 1, 2, 3, 4, 3, 1, 2, 3 ] ]
gap> l := List( [1 .. W.N+1], x -> CoxeterElementsLength( W, x-1 ) );;
gap> List( l, Length );
[ 1, 4, 9, 16, 23, 28, 30, 28, 23, 16, 9, 4, 1 ]
```

The last line tells us that there is 1 element of length 0, there are 4 elements of length 4, etc.

For many basic functions (like `CoxeterElementsLength`, `Reflections`, `Bruhat`, etc.) we have chosen the convention that if the input is an element of a Coxeter group, then this element should by given as a permutation (and similarly for the output). As a rule of thumb one should keep in mind that, if in some application one has to do a lot of computations with Coxeter group elements then using the low level GAP functions for permutations is usually much faster than manipulating lists of reduced expressions. For example, suppose we are given an element $w \in W$ and a generator s_i and we want to check if $l(s_i w) < l(w)$. Then it is more efficient to represent w by a permutation and to check the condition `i^w>W.N` than to work with a reduced expression and check the condition

```
Length( ReducedCoxeterWord( W, Concatenation( [i], w ) ) )<Length( w );
```

3.1 PermCoxeterWord

`PermCoxeterWord(` W `,` w `)`

returns the permutation on the root vectors determined by an element which is given as a list w of integers representing the standard generators of the Coxeter group W.

```
gap> W := CoxeterGroup( "G", 2 );;
gap> PermCoxeterWord( W, [1,2,1] );
( 1,12)( 2, 4)( 3, 9)( 6, 7)( 8,10)
```

See also 3.2.

This function requires the package "chevie" (see ??).

3.2 CoxeterWord

`CoxeterWord(` *W* , *w* `)`

returns a reduced word in the standard generators of the Coxeter group W determined by the permutation w on the root vectors.

```
gap> W := CoxeterGroup( "A", 3 );;
gap> w := ( 1,11)( 3,10)( 4, 9)( 5, 7)( 6,12);;
gap> w in W;
true;
gap> CHEVIE.BrieskornNormalForm := true;;
gap> CoxeterWord( W, w );
[ 1, 3, 2, 1, 3 ]
gap> CHEVIE.BrieskornNormalForm := false;;
gap> CoxeterWord( W, w );
[ 1, 2, 3, 2, 1 ]
```

For the definition of the Brieskorn normal form, see the introduction to this chapter. If the global variable `CHEVIE.BrieskornNormalForm` is set to `false` (which is automatically the case when you load CHEVIE), the result of `CoxeterWord` is the lexicographically smallest reduced word for w.

See also 3.1 and 3.4.

This function requires the package "chevie" (see ??).

3.3 CoxeterLength

`CoxeterLength(` *W* , *w* `)`

returns the length of the permutation w, which corresponds to an element in the Coxeter group W, as a reduced expression in the standard generators.

```
gap> W := CoxeterGroup( "F", 4 );;
gap> p := PermCoxeterWord( W, [ 1, 2, 3, 4, 2 ] );
( 1,44,38,25,20,14)( 2, 5,40,47,48,35)( 3, 7,13,21,19,15)
( 4, 6,12,28,30,36)( 8,34,41,32,10,17)( 9,18)(11,26,29,16,23,24)
(27,31,37,45,43,39)(33,42)
gap> CoxeterLength( W, p );
5
gap> CoxeterWord( W, p );
[ 1, 2, 3, 2, 4 ]
```

This function requires the package "chevie" (see ??).

3.4 ReducedCoxeterWord

`ReducedCoxeterWord(` W , w `)`

returns a reduced expression for an element of the Coxeter group W, which is given as a list w of integers where each entry `i` in this list represents the `i`-th standard generator of W.

```
gap> W := CoxeterGroup( "E", 6 );;
gap> ReducedCoxeterWord( W, [ 1, 1, 1, 1, 1, 2, 2, 2, 3 ] );
[ 1, 2, 3 ]
```

This function requires the package "chevie"(see ??).

3.5 LeftDescentSet

`LeftDescentSet(` W, w `)`

The set of generators s such that $l(sw) < l(w)$, given as a list of integers.

```
gap> W := CoxeterGroup( "A", 2 );;
gap> w := PermCoxeterWord( W, [ 1, 2 ] );;
gap> LeftDescentSet( W, w );
[ 1 ]
```

See also 3.6.

This function requires the package "chevie"(see ??).

3.6 RightDescentSet

`RightDescentSet(` W, w `)`

The set of generators s such that $l(ws) < l(w)$, given as a list of integers.

```
gap> W := CoxeterGroup( "A", 2 );;
gap> w := PermCoxeterWord( W, [ 1, 2 ] );;
gap> RightDescentSet( W, w );
[ 2 ]
```

See also 3.5.

This function requires the package "chevie"(see ??).

3.7 Reflections

`Reflections(` W `)`

returns the set of reflections in the Coxeter group W (as permutations). The `i`-th entry in this list is the reflection along the `i`-th root in W`.roots`.

```
gap> W := CoxeterGroup( "B", 2 );; W.roots;
[ [ 1, 0 ], [ 0, 1 ], [ 1, 1 ], [ 2, 1 ], [ -1, 0 ], [ 0, -1 ],
  [ -1, -1 ], [ -2, -1 ] ]
gap> Reflections( W );
[ (1,5)(2,4)(6,8), (1,3)(2,6)(5,7), (2,8)(3,7)(4,6), (1,7)(3,5)(4,8),
  (1,5)(2,4)(6,8), (1,3)(2,6)(5,7), (2,8)(3,7)(4,6), (1,7)(3,5)(4,8) ]
```

This function requires the package "chevie"(see ??).

3.8 LongestCoxeterElement

`LongestCoxeterElement(` *W* `)`

returns the unique element of maximal length of the Coxeter group W as a permutation.

```
gap> LongestCoxeterElement( CoxeterGroup( "A", 4 ) );
( 1,14)( 2,13)( 3,12)( 4,11)( 5,17)( 6,16)( 7,15)( 8,19)( 9,18)(10,20)
```

This function requires the package "chevie"(see ??).

3.9 LongestCoxeterWord

`LongestCoxeterWord(` *W* `)`

returns a reduced expression in the standard generators for the unique element of maximal length of the Coxeter group W.

```
gap> LongestCoxeterWord( CoxeterGroup( "A", 4 ) );
[ 1, 2, 1, 3, 2, 1, 4, 3, 2, 1 ]
```

This function requires the package "chevie"(see ??).

3.10 HighestShortRoot

`HighestShortRoot(` *W* `)`

Let W be an irreducible Coxeter group. `HighestShortRoot` computes the unique short root of maximal height of W. Note that if all roots have the same length then this is the unique root of maximal height, which can also be obtained by `W.roots[W.N]`. An error message is returned for W not irreducible.

```
gap> W := CoxeterGroup( "G", 2 );;  W.roots;
[ [ 1, 0 ], [ 0, 1 ], [ 1, 1 ], [ 1, 2 ], [ 1, 3 ], [ 2, 3 ],
  [ -1, 0 ], [ 0, -1 ], [ -1, -1 ], [ -1, -2 ], [ -1, -3 ],
  [ -2, -3 ] ]
gap> HighestShortRoot( W );
4
gap> W1 := CoxeterGroup( "A", 1, "B", 3 );;
gap> HighestShortRoot( W1 );
Error,  group is not irreducible
 in
HighestShortRoot( W1 ) called from
main loop
brk>
```

This function requires the package "chevie"(see ??).

3.11 CoxeterElementsLength

`CoxeterElementsLength(` *W*`,` *l* `)`

returns as a list of permutations the list of all elements of W of length l.

```
gap> W := CoxeterGroup( "G", 2 );;
```

```
gap> e := CoxeterElementsLength( W, 6 );
[ ( 1, 7)( 2, 8)( 3, 9)( 4,10)( 5,11)( 6,12) ]
gap> e[1] = LongestCoxeterElement( W );
true
```

The result of the computation of elements of a given length is stored in the component `elts` of the record of W. Thus `W.elts[lw+1]` contains the list of all elements of length lw in W. There are a number of programs (like `criticalPairs`, see Chapter 10) which refer to the ordering of the elements in these lists in `W.elts`.

This function requires the package "chevie"(see ??).

3.12 CoxeterWords

`CoxeterWords(` W `)`

returns the list of all elements in the Coxeter group W. The ordering is the same as that given by concatenating the lists of elements of length $0, 1, \ldots$ obtained by the function `CoxeterElementsLength`.

```
gap> CoxeterWords( CoxeterGroup( "G", 2 ) );
[ [  ], [ 2 ], [ 1 ], [ 2, 1 ], [ 1, 2 ], [ 2, 1, 2 ], [ 1, 2, 1 ],
  [ 2, 1, 2, 1 ], [ 1, 2, 1, 2 ], [ 2, 1, 2, 1, 2 ],
  [ 1, 2, 1, 2, 1 ], [ 1, 2, 1, 2, 1, 2 ] ]
```

This function requires the package "chevie"(see ??).

3.13 CoxeterConjugacyClasses

`CoxeterConjugacyClasses(` W `)`

returns a list of representatives of the conjugacy classes of the Coxeter group W. Each element in this list is given as a word in the standard generators, where the generator s_i is represented by the number i in a list. Each representative has the property that it is of minimal length in its conjugacy class.

```
gap> CoxeterConjugacyClasses( CoxeterGroup( "F", 4 ) );
[ [  ],
  [ 1, 2, 1, 3, 2, 1, 3, 2, 3, 4, 3, 2, 1, 3, 2, 3, 4, 3, 2, 1, 3, 2,
      3, 4 ], [ 2, 3, 2, 3 ], [ 2, 1 ],
  [ 2, 3, 2, 3, 4, 3, 2, 1, 3, 4 ],
  [ 3, 2, 4, 3, 2, 1, 3, 2, 4, 3, 2, 1 ], [ 4, 3 ],
  [ 1, 2, 1, 4, 3, 2, 1, 3, 2, 3 ],
  [ 3, 2, 1, 4, 3, 2, 1, 3, 2, 3, 4, 3, 2, 1, 3, 2 ],
  [ 3, 2, 4, 3, 2, 1, 3, 2 ], [ 4, 3, 2, 1 ], [ 1 ],
  [ 2, 3, 2, 3, 4, 3, 2, 3, 4 ], [ 1, 4, 3 ], [ 4, 3, 2 ],
  [ 2, 3, 2, 1, 3 ], [ 3 ], [ 1, 2, 1, 3, 2, 1, 3, 2, 3 ],
  [ 2, 1, 4 ], [ 3, 2, 1 ], [ 2, 4, 3, 2, 3 ], [ 1, 3 ], [ 3, 2 ],
  [ 2, 3, 2, 3, 4, 3, 2, 1, 3, 2, 4, 3, 2, 1 ], [ 2, 4, 3, 2, 1, 3 ] ]
```

See also 3.14.

This function requires the package "chevie"(see ??).

3.14 ChevieClassInfo

`ChevieClassInfo(` *W* `)`

returns information about the conjugacy classes of the finite Coxeter group *W*. The result is a record with three components: `classtext` contains `CoxeterConjugacyClass(`*W*`)`, `classnames` contains corresponding names for the classes, and `classparams` gives corresponding parameters for the classes (see also Chapter 4).

```
gap> W := CoxeterGroup( "D", 4 );;
gap> ChevieClassInfo( W );
rec(
  classtext :=
   [ [  ], [ 1, 2 ], [ 1, 2, 3, 1, 2, 3, 4, 3, 1, 2, 3, 4 ], [ 1 ],
      [ 1, 2, 3 ], [ 1, 2, 4 ], [ 1, 4 ], [ 2, 4 ],
      [ 1, 3, 1, 2, 3, 4 ], [ 1, 3 ], [ 1, 2, 3, 4 ], [ 1, 3, 4 ],
      [ 2, 3, 4 ] ],
  classparams :=
   [ [ [ [ 1, 1, 1, 1 ], [  ] ] ], [ [ [ 1, 1 ], [ 1, 1 ] ] ],
      [ [ [  ], [ 1, 1, 1, 1 ] ] ], [ [ [ 2, 1, 1 ], [  ] ] ],
      [ [ [ 1 ], [ 2, 1 ] ] ], [ [ [ 2 ], [ 1, 1 ] ] ],
      [ [ [ 2, 2 ], '+' ] ], [ [ [ 2, 2 ], '-' ] ],
      [ [ [  ], [ 2, 2 ] ] ], [ [ [ 3, 1 ], [  ] ] ],
      [ [ [  ], [ 3, 1 ] ] ], [ [ [ 4 ], '+' ] ], [ [ [ 4 ], '-' ] ] ]
   ,
  classnames := [ "1111.", "11.11", ".1111", "211.", "1.21", "2.11",
      "22.+", "22.-", ".22", "31.", ".31", "4.+", "4.-" ] )
```

For a general description of the conjugacy classes in the Weyl groups, see [Car72]. The relevance of taking representatives of minimal length is explained in [GP93].

See also 4.10.

This function requires the package "chevie"(see ??).

3.15 Bruhat

`Bruhat(` *W*`,` *y*`,` *w*`[,` *ly*`,` *lw*`] )`

returns `true`, if the element y is less than or equal to the element w of the Coxeter group W for the Bruhat order, and `false` otherwise (y is less than w if a reduced expression for y can be extracted from one for w). Both y and w must be given as permutations on the root vectors of W. The optional arguments ly, lw can contain the length of the elements y and w. (In a computation with many calls to `Bruhat` this may speed up the computation considerably.) See [Hum90, (5.9) and (5.10)] for further properties of the Bruhat order.

```
gap> W := CoxeterGroup( "H", 3 );;
gap> w := PermCoxeterWord( W, [ 1, 2, 1, 3 ] );;
gap> b := Filtered( Elements( W ), i -> Bruhat( W, i, w,
>                                 CoxeterLength( W, i ), 4 ) );;
gap> List( b, x -> CoxeterWord( W, x ) );
[ [  ], [ 3 ], [ 2 ], [ 2, 1 ], [ 2, 3 ], [ 2, 1, 3 ], [ 1 ],
```

```
[ 1, 3 ], [ 1, 2 ], [ 1, 2, 1 ], [ 1, 2, 3 ], [ 1, 2, 1, 3 ] ]
```

This function requires the package "chevie"(see ??).

3.16 MatXPerm

`MatXPerm(` W, w `)`

Let w be a permutation of the roots of the Coxeter datum W acting on the vector space V. `MatXPerm` returns the matrix of a linear transformation of V which acts trivially on the orthogonal of the coroots and has same effect as w on the simple roots. Only the image of the simple roots by w is used.

```
gap> W := CoxeterGroup(
> [ [ 2, 0,-1, 0, 0, 0, 1 ], [ 0, 2, 0,-1, 0, 0, 0 ],
>   [-1, 0, 2,-1, 0, 0,-1 ], [ 0,-1,-1, 2,-1, 0, 0 ],
>   [ 0, 0, 0,-1, 2,-1, 0 ], [ 0, 0, 0, 0,-1, 2, 0 ] ],
> [ [ 1, 0, 0, 0, 0, 0, 0 ], [ 0, 1, 0, 0, 0, 0, 0 ],
>   [ 0, 0, 1, 0, 0, 0, 0 ], [ 0, 0, 0, 1, 0, 0, 0 ],
>   [ 0, 0, 0, 0, 1, 0, 0 ], [ 0, 0, 0, 0, 0, 1, 0 ] ] );;
gap> w0 := LongestCoxeterElement( W );;
gap> mx := MatXPerm( W, w0 );
[ [ 0, 0, 0, 0, 0, -1, 1 ], [ 0, -1, 0, 0, 0, 0, 2 ],
  [ 0, 0, 0, 0, -1, 0, 3 ], [ 0, 0, 0, -1, 0, 0, 4 ],
  [ 0, 0, -1, 0, 0, 0, 3 ], [ -1, 0, 0, 0, 0, 0, 1 ],
  [ 0, 0, 0, 0, 0, 0, 1 ] ]
```

This function requires the package "chevie"(see ??).

3.17 MatYPerm

`MatYPerm(` W, w `)`

Let w be a permutation of the roots of the Coxeter datum W acting on the vector space $V^\vee$. `MatYPerm` returns the matrix of a linear transformation of $V^\vee$ which acts trivially on the orthogonal of the roots and has same effect as w on the simple coroots. Only the image of the simple coroots by w is used.

We continue the example from `MatXPerm` and obtain:

```
gap> my := MatYPerm( W, w0 );
[ [ 0, 0, 0, 0, 0, -1, 0 ], [ 0, -1, 0, 0, 0, 0, 0 ],
  [ 0, 0, 0, 0, -1, 0, 0 ], [ 0, 0, 0, -1, 0, 0, 0 ],
  [ 0, 0, -1, 0, 0, 0, 0 ], [ -1, 0, 0, 0, 0, 0, 0 ],
  [ 1, 2, 3, 4, 3, 1, 1 ] ]
```

This function requires the package "chevie"(see ??).

3.18 PermMatX

`PermMatX(` W, M `)`

Let M be a linear transformation of the vector space V on which the Coxeter datum W acts which preserves the set of roots. `PermMatX` returns the corresponding permutation of the roots; it signals an error if M does not normalize the set of roots.

We continue the example from `MatXPerm` and obtain:

```
gap> PermMatX( W, mx ) = w0;
true
```

This function requires the package "chevie"(see ??).

3.19 PermMatY

`PermMatX(` W`,` M `)`

Let M be a linear transformation of the vector space $V^\vee$ on which the Coxeter datum W acts which preserves the set of coroots. `PermMatY` returns the corresponding permutation of the coroots; it signals an error if M does not normalize the set of coroots.

We continue the example from `MatYPerm` and obtain:

```
gap> PermMatY( W, my ) = w0;
true
```

This function requires the package "chevie"(see ??).

Chapter 4

Character tables for Coxeter groups

The ordinary complex character table of any finite Coxeter group is computed in CHEVIE using the function `CharTable`. This command first checks whether or not W is irreducible. For each irreducible Coxeter group a character table record is computed either using recursive formulas (for the classical types: these functions are part of GAP and described in detail in [Pfe94a]) or read into the system from a library file (for the exceptional types: the tables can be found in the Cambridge ATLAS [CCN+85], for example). Thus, character tables can be obtained quickly even for very large groups (e.g., E_8) since they are obtained via the classification and special algorithms or stored tables for irreducible groups. The above record is a usual character table record as defined in GAP, but with some additional components.

It is important to note that the conjugacy classes and the irreducible characters of a finite Coxeter group each have a canonical labeling by certain combinatorial objects, and that such labelings are contained in a consistent way in the tables of CHEVIE. For the classes, these are partitions, pairs of partitions, or Carter's admissible diagrams [Car72]. For the characters, these are again partitions, pairs of partitions, or pairs of two integers (n, e) where n is the degree of the character and e is the smallest symmetric power of the natural reflection representation containing the given character as a constituent. This information is obtained by using the functions `ChevieClassInfo` and `ChevieCharInfo`. It is printed automatically when you display the character table in GAP.

The prototype for this is the symmetric group $\mathfrak{S}_{n+1}$ (type A_n) where the classes and characters are parametrized by partitions of $n + 1$.

```
gap> W := CoxeterGroup( "A", 3 );;
gap> Display( CharTable( W ));
W( A3 )

     2    3    2    3    .  2
     3    1    .    .    1  .

       1111  211   22   31  4
```

```
        2P 1111 1111 1111   31 22
        3P 1111  211   22 1111  4

   1111      1   -1    1    1 -1
   211       3   -1   -1    .  1
   22        2    .    2   -1  .
   31        3    1   -1    . -1
   4         1    1    1    1  1
```

Recall that our Coxeter groups acts a reflection group on the real vector space V (we do not assume that V is spanned by the roots, i.e., that `W.rank = W.semisimpleRank`). This reflection representation and its character contain some further useful information about W.

Let SV be the symmetric algebra of V. The invariants of W in SV are called the **polynomial invariants** of W. They are generated as a polynomial ring by $\dim V$ homogeneous algebraically independent polynomials $f_1, \ldots, f_{\dim V}$. The polynomials f_i are not uniquely determined but their degrees are. The f_i are called the **basic invariants** of W, and their degrees the **reflection degrees** of W. Let I be the ideal generated by the homogeneous invariants of positive degree in SV. Then SV/I is isomorphic to the regular representation of W as a W-module. It is thus a graded (by the degree of elements of SV) version of the regular representation of W. The polynomial which gives the graded multiplicity of a character χ of W in the graded module SV/I is called the **fake degree** of χ. (See 4.2, 4.3 and 4.5.)

Using these constructions and the generic degrees of the corresponding one-parameter generic Iwahori–Hecke algebra, one can associate four integers a, A, b, B with each irreducible character of W (see the functions 4.8, 4.9, 4.6, 4.7, and [Car85, Ch.11] for more details). These will also be used in the operations of truncated inductions in chapter 5.

Iwahori-Hecke algebras associated with finite Coxeter groups also have character tables, see Chapter 8.

We now describe, for each type of irreducible finite Coxeter groups, our conventions about labeling the classes and characters. Assume that the Dynkin diagram of W is labeled as in chapter 2.

Type A_n ($n \geq 0$). In this case we have $W \cong \mathfrak{S}_{n+1}$. The classes and characters are labeled by partitions of $n+1$. The partition corresponding to a class describes the cycle type for the elements in that class. The partition corresponding to a character describes the type of the Young subgroup such that the trivial character induced from this subgroup contains that character with multiplicity 1 and every other character occuring in this induced character has a higher a-value. Thus, the sign character corresponds to the partition (1^{n+1}) and the trivial character to the partition $(n+1)$. The character of the reflection representation of W is labeled by $(n, 1)$.

Type B_n ($n \geq 2$). In this case $W = W(B_n)$ is isomorphic to the wreath product of the cyclic group of order 2 with the symmetric group $\mathfrak{S}_n$. Hence the classes and characters are parametrized by pairs of partitions such that the total sum of their parts equals n. The pair corresponding to a class describes the signed cycle type for the elements in that class, as in [Car72]. We use the convention that if (λ, μ) is such a pair then λ corresponds to the positive and μ to the negative cycles. Thus, $(1^n, -)$ and $(-, 1^n)$ label the trivial class and the class containing the longest element, respectively. The pair corresponding to an irreducible character is determined via Clifford theory, as follows.

We have a semidirect product decomposition $W(B_n) = N.\mathfrak{S}_n$ where N is the standard n-dimensional $\mathbb{F}_2^n$-vector space. For $a, b \geq 0$ such that $n = a + b$ let $\eta_{a,b}$ be the irreducible character of N which takes value 1 on the first a standard basis vectors and value -1 on the next b standard basis vectors of N. Then the inertia subgroup of $\eta_{a,b}$ has the form $T_{a,b} := N.(\mathfrak{S}_a \times \mathfrak{S}_b)$ and we can extend $\eta_{a,b}$ trivially to an irreducible character $\tilde{\eta}_{a,b}$ of $T_{a,b}$. Let α and β be partitions of a and b, respectively. We take the tensor product of the corresponding irreducible characters of $\mathfrak{S}_a$ and $\mathfrak{S}_b$ and regard this as an irreducible character of $T_{a,b}$. Multiplying this character with $\tilde{\eta}_{a,b}$ and inducing to $W(B_n)$ yields an irreducible character $\chi = \chi_{(\alpha,\beta)}$ of $W(B_n)$. This defines the correspondence between irreducible characters and pairs of partitions as above.

For example, the pair $((n), -)$ labels the trivial character and $(-, (1^n))$ labels the sign character. The character of the natural reflection representation is labeled by $((n-1), (1))$.

Type D_n ($n \geq 4$). In this case $W = W(D_n)$ can be embedded as a subgroup of index 2 into the Coxeter group $W(B_n)$. The intersection of a class of $W(B_n)$ with $W(D_n)$ is either empty or a single class in $W(D_n)$ or splits up into two classes in $W(D_n)$. This also leads to a parametrization of the classes of $W(D_n)$ by pairs of partitions (λ, μ) as before but where the number of parts of μ is even and where there are two classes of this type if μ is empty and all parts of λ are even. In the latter case we denote the two classes in $W(D_n)$ by $(\lambda, +)$ and $(\lambda, -)$, where we use the convention that the class labeled by $(\lambda, +)$ contains a representative which can be written as a word in $\{s_1, s_3, \ldots, s_n\}$ and $(\lambda, -)$ contains a representative which can be written as a word in $\{s_2, s_3, \ldots, s_n\}$.

By Clifford theory the restriction of an irreducible character of $W(B_n)$ to $W(D_n)$ is either irreducible or splits up into two irreducible components. Let (α, β) be a pair of partitions with total sum of parts equal to n. If $\alpha \neq \beta$ then the restrictions of the irreducible characters of $W(B_n)$ labeled by (α, β) and (β, α) are irreducible and equal. If $\alpha = \beta$ then the restriction of the character labeled by (α, α) splits into two irreducible components which we denote by $(\alpha, +)$ and $(\alpha, -)$. Note that this can only happen if n is even. In order to fix the notation we use a result of [Ste89] which describes the value of the difference of these two characters on a class of the form $(\lambda, +)$ in terms of the character values of the symmetric group $\mathfrak{S}_{n/2}$. Recall that it is implicit in the notation $(\lambda, +)$ that all parts of λ are even. Let λ' be the partition of $n/2$ obtained by dividing each part by 2. Then the value of

$$\chi_{(\alpha,-)} - \chi_{(\alpha,+)}$$

on an element in the class $(\lambda, +)$ is given by $2^{k(\lambda)}$ times the value of the irreducible character of $\mathfrak{S}_{n/2}$ labeled by α on the class of cycle type λ'. (Here, $k(\lambda)$ denotes the number of non-zero parts of λ.)

The labels for the trivial, the sign and the natural reflection character are the same as for $W(B_n)$, since these characters are restrictions of the corresponding characters of $W(B_n)$.

Types G_2 and F_4. The matrices of character values and the orderings and labelings of the irreducible characters are exactly the same as in [Car85, p.412/413]. Note, however, that in CHEVIE we have reversed the labeling of the Dynkin diagrams to be in accordance with the conventions in [Lus85, (4.8) and (4.10)].

The classes are labeled by Carter's admissible diagrams [Car72]. A character is labeled by a pair (n, b) where n denotes the degree and b the corresponding b-invariant. If there are several characters with the same pair (n, b) we attach a prime to them, as in [Car85].

For type F_4 the result of **ChevieCharInfo** contains an additional component **kondo** which contains the labels originally given by Kondo (and which are also used in [Lus85, (4.10)]). The reflection character is labeled by $(4, 1)$ or 4_2 (Kondo).

Types E_6, E_7, E_8. The character tables are re-constructed from the compound tables in the Cambridge ATLAS [CCN+85], p.26, p.46 and p.85, respectively (or, they can be recomputed with GAP by the general Dixon-Schneider algorithm). The classes are labeled by Carter's admissible diagrams [Car72]. A character is labeled by the pair (n, b) where n denotes the degree and b is the corresponding b-invariant. For these types, this gives a unique labeling of the characters. The result of **ChevieCharInfo** contains an additional component **frame** which contains the labels originally given by Frame (and which are used in [Lus85], (4.11), (4.12), and (4.13)). For type E_6, E_7, E_8, respectively, the reflection character is the one with label $(6, 1)$, $(7, 1)$, $(8, 1)$ or 6_p, $7'_a$, 8_z (Frame).

Non-crystallographic types $I_2(m)$, H_3, H_4. In these cases we do not have canonical labelings for the classes.

Each character for type H_3 is uniquely determined by the pair (n, b) where n is the degree and b the corresponding b-invariant. For type H_4 there are just two characters (those of degree 30) for which the corresponding pairs are the same. These two characters are nevertheless distinguished by their fake degrees: the first of these (in the CHEVIE-table) has fake degree $q^{10} + q^{12}+$ higher terms, while the second has fake degree $q^{12} + q^{14}+$ higher terms. The characters in the CHEVIE-table for type H_4 are ordered in the same way as in [AL82].

Finally, the characters of degree 2 for type $I_2(m)$ are ordered as follows. Let ε be a primitive m-th root of unity. Then matrix representations affording the characters of degree 2 are given by:

$$\varphi_j : s_1 s_2 \mapsto \begin{pmatrix} \varepsilon^j & 0 \\ 0 & \varepsilon^{-j} \end{pmatrix}, \quad s_1 \mapsto \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix},$$

where $1 \leq j \leq (m-1)/2$ for m odd, and $1 \leq j \leq (m-2)/2$ for m even. The natural reflection representation determined by the root system is $\rho = \varphi_1$.

In GAP we take ε as `E(` m `)`. Then the characters in the CHEVIE-table are ordered as $\varphi_1, \varphi_2, \ldots$.

4.1 CharTable for Coxeter groups

`CharTable(` W `)`

returns the character table of the Coxeter group W. This is a character table record as defined in GAP. It is assigned to the record component `charTable` of W. The function `CharTable` checks if this component is already bound. The (variable) record components `classtext`, `classnames` and `irredinfo` contain the following information.

`classtext`

a list of reduced expressions for minimal length representatives in the conjugacy classes of W

`classnames`
: a list of Carter's admissible diagrams (for exceptional, irreducible types) or partitions resp. pairs of partitions (for classical, irreducible types). If the Coxeter group is not irreducible tuples of such labels are given.

`irredinfo`
: a list of records with a component `charname` which contains a string with a label for an irreducible character. For an irreducible Coxeter group this label can either be a partition or a pair of partitions, or it is a pair (n, e) where n is the degree of the character and e is the smallest symmetric power of the character of the natural reflection representation which contains the given character as a component. The optional second component, `charparam`, contains this label as a list of numbers (if it is possible to give it in this way). The labels for the group is a list of labels corresponding to the irreducible components (as given by `CartanType`).

```
gap> W := CoxeterGroup( "G", 2);;
gap> ct := CharTable( W );
CharTable( "W( G2 )" )
gap> ct.classtext;
[ [  ], [ 2 ], [ 1 ], [ 1, 2 ], [ 1, 2, 1, 2 ], [ 1, 2, 1, 2, 1, 2 ] ]
gap> ct.classnames;
[ " ", "~A_1", "A_1", "G_2", "A_2", "A_1 + ~A_1" ]
gap> ct.irredinfo;
[ rec(
      charparam := [ [ 1, 0 ] ],
      charname := "phi_{1,0}" ), rec(
      charparam := [ [ 1, 6 ] ],
      charname := "phi_{1,6}" ), rec(
      charparam := [ [ 1, 3, "'" ] ],
      charname := "phi_{1,3}'" ), rec(
      charparam := [ [ 1, 3, "''" ] ],
      charname := "phi_{1,3}''" ), rec(
      charparam := [ [ 2, 1 ] ],
      charname := "phi_{2,1}" ), rec(
      charparam := [ [ 2, 2 ] ],
      charname := "phi_{2,2}" ) ]
```

These functions require the package "chevie"(see ??).

4.2 ReflectionCharValue

`ReflectionCharValue(` W, w `)`

Let V be the vector space on which the Coxeter group W acts, and let w be the linear automorphism of V induced by a permutation of the roots (acting trivially on the orthogonal of the roots if `W.rankW.semisimpleRank`). `ReflectionCharValue` returns the trace of w on V.

```
gap> W := CoxeterGroup( "A", 3 );
CoxeterGroup("A", 3)
gap> List( Elements( W ), x -> ReflectionCharValue( W, x ) );
```

```
[ 3, 1, -1, 0, -1, -1, 0, 1, 0, -1, 1, 0, 1, -1, 0, 1, -1, -1, 0, -1,
  1, 0, -1, 0 ]
```

This function requires the package "chevie"(see ??).

4.3 ReflectionDegrees

`ReflectionDegrees(` *W* `)`

returns a list holding the degrees of W as a reflection group on the vector space V on which it acts. These are the degrees $d_1, \ldots, d_{\dim V}$ of the basic invariants of W in SV, written in increasing order. They reflect various properties of W; in particular, their product is the size of W.

```
gap> W := CoxeterGroup( "H", 4 );
CoxeterGroup("H", 4)
gap> ReflectionDegrees( W );
[ 2, 12, 20, 30 ]
gap> Size( W );
14400
```

This function requires the package "chevie"(see ??).

4.4 FakeDegrees

`FakeDegrees(` *W*`,` *q* `)`

returns a list holding the fake degrees of W as a reflection group on the vector space V on which it acts, evaluated at q. These are the graded multiplicities of the irreducible characters of W in the quotient SV/I where SV is the symmetric algebra of V and I is the ideal generated by the homogeneous invariants of positive degree in SV. The ordering of the result corresponds to the ordering of the characters in `CharTable(W)`.

```
gap> q := X( Rationals );; q.name := "q";;
gap> FakeDegrees( CoxeterGroup( "A", 2 ), q );
[ q^3, q^2 + q, q^0 ]
```

This function requires the package "chevie"(see ??).

4.5 FakeDegree

`FakeDegree(` *W*`,` *phi*`,` *q* `)`

returns the fake degree of the character labelled by *phi* of the Coxeter group W as a reflection group on the vector space V on which it acts, evaluated at q. The label *phi* should be the parameter given as a result of `CharParams` (see section 2.6).

```
gap> q := X( Rationals );; q.name := "q";;
gap> FakeDegree( CoxeterGroup( "A", 2 ), [ [ 2, 1 ] ], q );
q^2 + q
```

This function requires the package "chevie"(see ??).

4.6 LowestPowerFakeDegrees

`LowestPowerFakeDegrees(` W `)`

return a list holding the b-function for all irreducible characters of W, that is, for each character χ, the valuation of the fake degree of χ. The ordering of the result corresponds to the ordering of the characters in `CharTable(W)`.

```
gap> LowestPowerFakeDegrees( CoxeterGroup( "D", 4 ) );
[ 6, 6, 7, 12, 4, 3, 6, 2, 2, 4, 1, 2, 0 ]
```

This function requires the package "chevie"(see ??).

4.7 HighestPowerFakeDegrees

`HighestPowerFakeDegrees(` W `)`

returns a list holding the B-function for all irreducible characters of W, that is, for each character χ, the degree of the fake degree of χ. The ordering of the result corresponds to the ordering of the characters in `CharTable(W)`.

```
gap> HighestPowerFakeDegrees( CoxeterGroup( "D", 4 ) );
[ 10, 10, 11, 12, 8, 9, 10, 6, 6, 8, 5, 6, 0 ]
```

This function requires the package "chevie"(see ??).

4.8 LowestPowerGenericDegrees

`LowestPowerGenericDegrees(` W `)`

returns a list holding the a-function for all irreducible characters of the Coxeter group W, that is, for each character χ, the valuation of the generic degree of χ (in the one-parameter Iwahori-Hecke algebra `Hecke(W,X(Rationals))` corresponding to W). The ordering of the result corresponds to the ordering of the characters in `CharTable(W)`.

```
gap> LowestPowerGenericDegrees( CoxeterGroup( "D", 4 ) );
[ 6, 6, 7, 12, 3, 3, 6, 2, 2, 3, 1, 2, 0 ]
```

This function requires the package "chevie"(see ??).

4.9 HighestPowerGenericDegrees

`HighestPowerGenericDegrees(` W `)`

returns a list holding the A-function for all irreducible characters of the Coxeter group W, that is, for each character χ, the degree of the generic degree of χ (in the one-parameter Iwahori-Hecke algebra `Hecke(W,X(Rationals))` corresponding to W). The ordering of the result corresponds to the ordering of the characters in `CharTable(W)`.

```
gap> HighestPowerGenericDegrees( CoxeterGroup( "D", 4 ) );
[ 10, 10, 11, 12, 9, 9, 10, 6, 6, 9, 5, 6, 0 ]
```

This function requires the package "chevie"(see ??).

4.10 ChevieCharInfo

`ChevieCharInfo(` W `)`

returns information about the irreducible characters of the finite Coxeter group W. The result is a record with the components: `charparams` contains the result of `CharParams(`W`)`, `charnames` contains the corresponding names returned by `CharName`, and `a`, `A`, `b`, `B`, contain the results of `LowestPowerGenericDegrees(`W`)`, `HighestPowerGenericDegrees(`W`)`, `LowestPowerFakeDegrees(`W`)`, `HighestPowerFakeDegrees(`W`)`, respectively. Moreover, the components `positionId` and `positionSgn` contain the position of the trivial and the sign character, respectively, in the character table of W. If W is irreducible, there is an additional component `positionRefl` which contains the position of the character of the reflection representation.

```
gap>  ChevieCharInfo( CoxeterGroup( "G", 2 ) );
rec(
  charparams := [ [ [ 1, 0 ] ], [ [ 1, 6 ] ], [ [ 1, 3, "'" ] ],
      [ [ 1, 3, "''" ] ], [ [ 2, 1 ] ], [ [ 2, 2 ] ] ],
  charnames :=
   [ "phi_{1,0}", "phi_{1,6}", "phi_{1,3}'", "phi_{1,3}''",
      "phi_{2,1}", "phi_{2,2}" ],
  a := [ 0, 6, 1, 1, 1, 1 ],
  A := [ 0, 6, 5, 5, 5, 5 ],
  b := [ 0, 6, 3, 3, 1, 2 ],
  B := [ 0, 6, 3, 3, 5, 4 ],
  positionId := 1,
  positionSgn := 2,
  positionRefl := 5 )
```

If W is irreducible of type F_4 or of type E_n $(n = 6, 7, 8)$ then there is an additional component `kondo` or `frame`, respectively, which gives the labeling of the characters as determined by Kondo and Frame.

```
gap> W := CoxeterGroup( "E", 6 );;
gap> ChevieCharInfo( W ).frame;
[ "1_p", "1_p'", "10_s", "6_p", "6_p'", "20_s", "15_p", "15_p'",
  "15_q", "15_q'", "20_p", "20_p'", "24_p", "24_p'", "30_p", "30_p'",
  "60_s", "80_s", "90_s", "60_p", "60_p'", "64_p", "64_p'", "81_p",
  "81_p'" ]
```

See also 3.14.

This function requires the package "chevie"(see ??).

4.11 PositionId and PositionSgn

`PositionId(` W `)`

`PositionSgn(` W `)`

return the position of the trivial and the sign character, respectively, in the character table of the group W.

```
gap> W := CoxeterGroup( "D", 4 );;
```

```
gap> PositionId( W );
13
gap> PositionSgn( W );
4
```

For `PositionId`, one could also give an arbitrary finite group or its character table as argument.

See also 3.14.

This function requires the package "chevie" (see ??).

Chapter 5

Reflection subgroups

Let W be a finite Coxeter group, corresponding to a root system R in an Euclidean space V. Let S be the set of generators corresponding to some system of fundamental roots in R. Then $\{wsw^{-1} \mid w \in W, s \in S\}$ is the set of all reflections in W. A **reflection subgroup** H of W is a subgroup generated by the reflections it contains. If H is generated by reflections with respect to a subset of the fundamental roots it is called a **standard parabolic subgroup** of W. A **parabolic subgroup** is a subgroup H of W which is conjugate to some standard parabolic subgroup. A parabolic subgroup H is characterized among all reflection subgroups of W by the fact that the set of all roots in R for which the corresponding reflection lies in H is closed under (real) linear combinations.

It is a theorem discovered by Deodhar [Deo89] and Dyer [Dye90] independently at the same time that a reflection subgroup H has a canonical set of fundamental roots even if it is not parabolic: Let Q be the set of all roots for which the corresponding reflection lies in H; then a fundamental system of roots for H is given by the positive roots $t \in Q$ such that the set of roots whose sign is changed by the reflection with root t meets Q in the single element t. This is used by the routine `ReflectionSubgroup` to determine the root system Q of a reflection subgroup H. This function takes the original record for W and a list of indices for the roots as input.

```
gap> W := CoxeterGroup( "G", 2 );
CoxeterGroup("G", 2)
gap> W.roots[4];
[ 1, 2 ]
gap> H := ReflectionSubgroup( W, [ 2, 4 ] );
ReflectionSubgroup(CoxeterGroup("G", 2), [ 2, 3 ])
gap> PrintDynkinDiagram( H );
~A2    2 - 3                              # not a parabolic subgroup
```

We also see that the result of the above algorithm is that `W.roots[2]` and `W.roots[3]` form a system of simple roots in H.

The line containing the Dynkin diagram of H introduces a convention: we use the notation `"~A"` to denote a root subsystem of type `"A"` generated by short roots.

The record for the subgroup contains additional components the most important of which is `rootInclusion` which gives the positions of the roots in Q in the original root system R:

```
gap> H.rootInclusion;
[ 2, 3, 4, 8, 9, 10 ]
```

The inverse map is stored in `H.rootRestriction`. If H is a standard parabolic subgroup of W then the length function on H (with respect to its set of generators) is the restriction of the length function on W. This need not no longer be true for arbitrary reflection subgroups of W:

```
gap> CoxeterLength( W, H.generators[2] );
3
gap> CoxeterLength( H, H.generators[2] );
1
```

In GAP, the Coxeter group W is represented as a permutation group on R. Consequently, a reflection subgroup $H \subseteq W$ is a permutation subgroup, i.e., its elements are represented as permutations of the roots of the *parent* group. This has to be kept in mind when working with reduced expressions and functions like `CoxeterWord`, and `PermCoxeterWord`.

Reduced words in simple reflections of H:

```
gap> el := CoxeterWords( H );
[ [  ], [ 2 ], [ 3 ], [ 2, 3 ], [ 3, 2 ], [ 2, 3, 2 ] ]
```

Reduced words in the generators of H:

```
gap> el1 := List( el, x -> H.rootRestriction{ x } );
[ [  ], [ 1 ], [ 2 ], [ 1, 2 ], [ 2, 1 ], [ 1, 2, 1 ] ]
```

Permutations on the roots of W:

```
gap> el2 := List( el, x -> PermCoxeterWord( H, x ) );
[ (), ( 1, 5)( 2, 8)( 3, 4)( 7,11)( 9,10),
  ( 1,12)( 2, 4)( 3, 9)( 6, 7)( 8,10),
  ( 1, 5,12)( 2,10, 3)( 4, 9, 8)( 6, 7,11),
  ( 1,12, 5)( 2, 3,10)( 4, 8, 9)( 6,11, 7),
  ( 2, 9)( 3, 8)( 4,10)( 5,12)( 6,11) ]
```

Reduced words in the generators of W:

```
gap> List( el2, x -> CoxeterWord( W, x ) );
[ [  ], [ 2 ], [ 1, 2, 1 ], [ 2, 1, 2, 1 ], [ 1, 2, 1, 2 ],
  [ 2, 1, 2, 1, 2 ] ]
```

Another basic result about reflection subgroups is that each coset of H in W contains a unique element of minimal length. Since a coset is a subset of W, the length of elements is taken with respect to the roots of W.) See 5.4.

In many applications it is useful to know the decomposition of the irreducible characters of W when we restrict them from W to a reflection subgroup H (defined as in Section 5). In order to apply the usual GAP functions for inducing and restricting characters and computing scalar products, we need to know the fusion map for the conjugacy classes of H into those of W. This is done, as usual, with the GAP function `FusionConjugacyClasses`, which calls a special implementation for Coxeter groups. The decomposition of induced characters into irreducibles then is a simple matter of combining some functions which already exist in GAP. There is an additional function, `InductionTable`, which performs this job.

```
gap> W := CoxeterGroup( "G", 2 );;
```

```
gap> W.roots[4];
[ 1, 2 ]
gap> H := ReflectionSubgroup( W, [ 2, 4 ] );;
gap> Display( InductionTable( H, W ) );
Induction from ~A2 into G2
            | 111 21 3
______________________
phi_{1,0}   |   .  . 1
phi_{1,6}   |   1  . .
phi_{1,3}'  |   .  . 1
phi_{1,3}'' |   1  . .
phi_{2,1}   |   .  1 .
phi_{2,2}   |   .  1 .
```

We have similar functions for the j-induction and the J-induction of characters. These operations are obtained by truncating the induced characters by using the a-invariants and b-invariants associated with the irreducible characters of W (see 5.6 and 5.7).

5.1 ReflectionSubgroup

`ReflectionSubgroup(` W`,` r `)`

Returns the reflection subgroup of the real reflection group W generated by the reflections with roots specified by r. r is a list of indices specifying a subset of the roots of W.

A reflection subgroup H of W is a permutation subgroup, and otherwise has the same fields as a Coxeter group, with some new ones added which express the relationship with the parent W:

`rootInclusion`
: the indices of the roots in the roots of W

`parentN`
: the number of positive roots of W

`rootRestriction`
: a list of length `2*H.parentN` with entries in positions `H.rootInclusion` bound to `[1..2*H.N]`.

A Coxeter group which is not a subgroup actually also contains these fields, set to the trivial values:`rootInclusion = [ 1 .. 2*W.N ]`, `parentN = W.N` and `rootRestriction = [ 1 .. 2*W.N ]`.

With these fields, you can, e.g., test if the i-th fundamental root of H root is left positive by the element w of H (given as a permutation of the roots of the parent) with

`H.rootInclusion[ i ] ^w <= H.parentN`

Programs dealing with Coxeter groups should be written this way to work for Coxeter subgroups (instead of just writing `i^w <= H.N`).

`ReflectionSubgroup` returns a subgroup of the parent group of the argument (like the GAP function `Subgroup`).

```
gap> W := CoxeterGroup( "F", 4 );;
gap> H := ReflectionSubgroup( W, [ 1, 2, 11, 20 ] );
```

```
ReflectionSubgroup(CoxeterGroup("F", 4), [ 1, 2, 9, 16 ])
gap> CartanName( H );
"D4"                                 # not a parabolic subgroup
gap> H.rootRestriction;
[ 1, 2,,, 5,,,, 3,, 6,,, 8,, 4,, 7,, 9,, 10, 11, 12, 13, 14,,, 17,,,,
  15,, 18,,, 20,, 16,, 19,, 21,, 22, 23, 24 ]
gap> ReflectionSubgroup( H, [ 1, 2, 6 ] );
ReflectionSubgroup(CoxeterGroup("F", 4), [ 1, 2, 3 ])
```

This function requires the package "chevie"(see ??).

5.2 Functions for reflection subgroups

All functions for Coxeter groups are actually defined for reflection subgroups. The generators for the subgroups are labeled according to the corresponding number of the root they represent in the parent group. This affects the labeling given by all functions dealing with words and generators, e.g., `PrintDynkinDiagram` or `PermCoxeterWord`.

```
gap> W := CoxeterGroup( "F", 4 );
CoxeterGroup("F", 4)
gap> H := ReflectionSubgroup( W, [ 10, 11, 12 ] );
ReflectionSubgroup(CoxeterGroup("F", 4), [ 10, 11, 12 ])
gap> PrintDynkinDiagram( H );
C2    11 > 10
~A1    12
gap> LongestCoxeterWord( H );
[ 10, 11, 10, 11, 12 ]
```

Also, as one may notice in the example above, there is one particularity of the functions `CartanType`, `CartanName` and `PrintDynkinDiagram` for Coxeter subgroups: an irreducible subsystem which consists of short roots in a system which has longer roots (i.e., type `"B"`, `"C"`, `"G"` or `"F"`) is labeled as type `"~A"`.

These functions require the package "chevie"(see ??).

5.3 ReducedInCoxeterCoset

`ReducedInCoxeterCoset(` W `,` w `)`

w is an automorphism of the Coxeter group W or of a parent group of W, given as a permutation of the roots. `ReducedInCoxeterCoset` returns the unique element in the right coset $W.w$ which sends all roots of W to positive roots.

```
gap> W := CoxeterGroup("F", 4 );;
gap> H := ReflectionSubgroup( W, [ 1, 2, 9, 16 ] );;
gap> PrintDynkinDiagram( H );
D4   9
      \
       1 - 16
      /
     2
gap> w := PermCoxeterWord( W, [ 3, 2, 3, 4, 3, 2 ] );;
```

```
gap> f := ReducedInCoxeterCoset( H, w );;
gap> CoxeterWord( W, f );
[ 4, 3 ]
gap> H.rootInclusion{[ 1 ..4 ]};
[ 1, 2, 9, 16 ]
```

The triality automorphism of D_4:

```
gap> OnTuples( H.rootInclusion{[ 1 .. 4 ]}, f );
[ 1, 9, 16, 2 ]
```

This function requires the package "chevie" (see ??).

5.4 ReducedRightCosetRepresentatives

`ReducedRightCosetRepresentatives(` W, H `)`

returns a list of reduced elements in the Coxeter group W which are distinguished representatives for the right cosets of the reflection subgroup H in W. The distinguished representative in the coset $H.w$ is the unique element in the coset which sends all roots of H to positive roots (the element returned by `ReducedInCoxeterCoset`). It is also the element of minimal length in the coset. The representatives are returned in order of increasing length.

```
gap> W := CoxeterGroup( "B", 3 );;
gap> H := ReflectionSubgroup(W, [ 2, 3 ]);;
gap> List( ReducedRightCosetRepresentatives( W, H ),
>                                x-> CoxeterWord( W, x ) );
[ [  ], [ 1 ], [ 1, 2 ], [ 1, 2, 1 ], [ 1, 2, 3 ], [ 1, 2, 1, 3 ],
  [ 1, 2, 1, 3, 2 ], [ 1, 2, 1, 3, 2, 1 ] ]
```

This function requires the package "chevie" (see ??).

5.5 PermCosetsSubgroup

`PermCosetsSubgroup(` W, H `)`

returns the list of permutations induced by the standard generators of the Coxeter group W on the cosets of the Coxeter subgroup H. The cosets are in the order determined by the result of the function `ReducedRightCosetRepresentatives(` W, H `)`.

```
gap> W := CoxeterGroup( "F", 4 );;
gap> PermCosetsSubgroup( W, ReflectionSubgroup( W, [ 1, 2, 3 ] ) );
[ ( 4, 5)( 6, 7)( 8,10)(16,18)(17,20)(19,21),
  ( 3, 4)( 7, 9)(10,12)(14,16)(15,17)(21,22),
  ( 2, 3)( 4, 6)( 5, 7)( 9,11)(12,14)(13,15)(17,19)(20,21)(22,23),
  ( 1, 2)( 6, 8)( 7,10)( 9,12)(11,13)(14,15)(16,17)(18,20)(23,24) ]
```

This function requires the package "chevie" (see ??).

5.6 jInductionTable for Macdonald-Lusztig-Spaltenstein induction

`jInductionTable(` H, W `)`

computes the decomposition into irreducible characters of W of the j-induced of the irreducible characters of H. The j-induced of χ is the sum of the irreducible components of the induced of χ which have same b-function (see 4.6) as χ. In the table the rows correspond to the characters of the parent group, the columns to those of the subgroup. What is returned is actually a record with several fields:`scalar` contains the induction table proper, and there is a `Display` method. The other fields contain labeling information taken from the character tables of H and W when it exists.

```
gap> W := CoxeterGroup( "D", 4);;
gap> H := ReflectionSubgroup( W, [ 1, 3 ] );;
gap> Display( jInductionTable( H, W ) );
j-Induction from A2 into D4
      | 111 21 3
----------------
11.+  |   .  . .
11.-  |   .  . .
1.111 |   .  . .
.1111 |   .  . .
11.2  |   .  . .
1.21  |   1  . .
.211  |   .  . .
2.+   |   .  . .
2.-   |   .  . .
.22   |   .  . .
1.3   |   .  1 .
.31   |   .  . .
.4    |   .  . 1
```

This function requires the package "chevie"(see ??).

5.7 JInductionTable

`JInductionTable(` H, W `)`

`JInductionTable` computes the decomposition into irreducible characters of W of the J-induced of the irreducible characters of H. The J-induced of χ is the sum of the irreducible components of the induced of χ which have same a-function (see 4.8) as χ. In the table the rows correspond to the characters of the parent group, the columns to those of the subgroup. What is returned is actually a record with several fields:`scalar` contains the induction table proper, and there is a `Display` method. The other fields contain labeling information taken from the character tables of H and W when it exists.

```
gap> W := CoxeterGroup( "D", 4 );;
gap> H := ReflectionSubgroup( W, [ 1, 3 ] );;
gap> Display( JInductionTable( H, W ) );
J-Induction from A2 into D4
      | 111 21 3
----------------
11.+  |   .  . .
11.-  |   .  . .
1.111 |   .  . .
```

```
.1111 |    .  .  .
11.2  |    1  .  .
1.21  |    1  .  .
.211  |    .  .  .
2.+   |    .  .  .
2.-   |    .  .  .
.22   |    .  .  .
1.3   |    .  1  .
.31   |    .  .  .
.4    |    .  .  1
```

This function requires the package "chevie" (see ??).

Chapter 6

Artin-Tits braid groups

Let W be a Coxeter group, with presentation

$$\langle s_1, \ldots, s_n \mid s_i^2 = 1, \ \underbrace{s_i s_j s_i \cdots}_{m(i,j) \text{ factors}} = \underbrace{s_j s_i s_j \cdots}_{m(i,j) \text{ factors}} \quad \text{for all } i,\ j \text{ with } i \neq j\rangle$$

for some Coxeter matrix $M = \{m(i,j)\}_{i,j}$ such that the Coxeter group is finite. The braid group $B = B(M)$ associated to W is the group defined by the presentation

$$\langle \mathbf{s}_1, \ldots, \mathbf{s}_n \mid \ \underbrace{\mathbf{s}_i \mathbf{s}_j \mathbf{s}_i \cdots}_{m(i,j) \text{ factors}} = \underbrace{\mathbf{s}_j \mathbf{s}_i \mathbf{s}_j \cdots}_{m(i,j) \text{ factors}} \quad \text{for all } i,\ j \text{ with } i \neq j\rangle$$

Usually, we will use the same symbols for generators and elements of W and of B, except that we will use bold letters for elements of B. This is justified by the fact that there is a canonical section to the natural quotient map $B \to W$ given as follows. Let $w \in W$ and choose a reduced expression $w = s_1 \cdots s_m$. Then the corresponding product $\mathbf{w} = \mathbf{s}_1 \cdots \mathbf{s}_m \in B$ is independent of the chosen minimal expression. Let B^+ be the submonoid of B consisting of all words in the generators $\mathbf{s}_i$ of B (words with no inverses); in this way, we can identify W with the subset B^+_{red} of B^+ which consists of the elements of B^+ of the same length as their image in W.

If W is a real reflection group in the vector space V with root system R, then B has also a topological definition as the fundamental group Π^1 of the space $((V - \bigcup_{r \in R} H_r) \otimes \mathbb{C})/W$, where H_r is the hyperplane orthogonal to the root r; however, we will not use this here.

We will represent in GAP words of the braid group associated to the Coxeter group W using a normal form based on theorems of Deligne [Del72] (which extend previous work of Brieskorn, Saito [BS72] and Garside [Gar69]):

(i) Let $\mathbf{w}_0$ be the element in B^+ which lifts the longest element of W, as explained above. Then, given any element $\mathbf{b} \in B$, there is some power $\mathbf{w}_0^i$ such that $\mathbf{w}_0^i \mathbf{b} \in B^+$.

(ii) Let $\mathbf{b} \in B^+$. Then there is a unique longest element $\alpha(\mathbf{b})$ of B^+_{red} which divides $\mathbf{b}$ on the left (which means that $\mathbf{b} = \alpha(\mathbf{b})\mathbf{y}$ for some $\mathbf{y} \in B^+$).

We use (i) and (ii) to represent an element $\mathbf{b} \in B$: we first compute the smallest power of $\mathbf{w}_0$ such that $\mathbf{w}_0^i\mathbf{b} \in B^+$, and we represent $\mathbf{b}$ by the couple $(i, \mathbf{w}_0^i\mathbf{b})$. We are thus reduced to the case where $\mathbf{b} \in B^+$. We then represent $\mathbf{b}$ by the sequence of elements of the Coxeter group $\alpha(\mathbf{b}), \alpha(\alpha(\mathbf{b})^{-1}\mathbf{b}), \ldots$. Given a Coxeter group W, the function `Braid(W)` constructs an element of the associated braid group from a list of generators. Here is an example:

```
gap> W := CoxeterGroup( "A", 4 );;
gap> w := Braid( W )( 1, 2, 3, 4 );
1234
gap> w ^ 3;
121321432.343
gap> w ^ 4;
w0^1.232432
gap> w ^ -1;
w0^-1.232432
```

How the element is printed exactly is controlled by the variable `PrintBraid`. This variable is a component of the global file `CHEVIE`, and the user can change its value whenever he or she wants during a GAP session. When you load the CHEVIE package, `CHEVIE.PrintBraid` is initialized to the string `"Deligne"`, as in the above examples. If it is set to `"GAP"`, then the element is printed in a form which after assigning `B:=Braid(W);` can be input back into GAP:

```
gap> CHEVIE.PrintBraid;
"Deligne"
gap> CHEVIE.PrintBraid := "GAP";
"GAP"
gap> w;
B([ 1, 2, 3, 4 ])
gap> w ^ 3;
B([ 1, 2, 1, 3, 2, 1, 4, 3, 2, 3, 4, 3 ])
gap> w ^ -1;
B([ 2, 3, 2, 4, 3, 2 ],-1)
```

The third possible value `"Charney"` for `CHEVIE.PrintBraid` is based on an idea of Ruth Charney [Cha92] to find the smallest element $\mathbf{y} \in B^+$ such that $\mathbf{y}.\mathbf{b} \in B^+$, and print $\mathbf{b}$ by printing $(\mathbf{y})^{-1}$ followed by the Deligne normal form of $\mathbf{y}.\mathbf{b}$:

```
gap> CHEVIE.PrintBraid := "Charney";
"Charney"
gap> w ^ -1;
(1234)^-1.
```

6.1 Construction of braid elements

`Braid(` *W* `)(` *s1* `,` .. `,` *sn* `)`

`Braid(` *W* `)(` *list* `[,` *pw0* `])`

`Braid(` *W* `)(` *p* `[,` *pw0* `])`

Let W be a Coxeter group and let w be an element of W, represented as a permutation p of the roots, or as a sequence $s_1, .., s_n$ of integers representing a (non necessarily reduced) word in the generators of W. The calls above return the element of the braid monoid of W defined by w. If *pw0* (a positive or negative integer) is given, the resulting element is multiplied in the braid group by $\mathbf{w}_0^{pw0}$. The result of `Braid(W)` is a braid-making function, which can be assigned to make conveniently braid elements as in the example below.

A braid element $\mathbf{b}$ corresponding to the Coxeter group W is represented as a record with fields `pw0`, `elm` and `operations`. `elm` represents an element $\mathbf{x} \in B^+$, and `elm` and `pw0` are such that $\mathbf{b} = \mathbf{w}_0^{\mathrm{pw0}}\mathbf{x}$ and $\mathbf{w}_0^{-1}\mathbf{x} \notin B^+$. `elm` is a sequence of elements of W (permutations of the roots), which are the image in W of $\alpha(\mathbf{b}), \alpha(\alpha(\mathbf{b})^{-1}\mathbf{b}), \ldots$. `operations` contains the operations described in the section 6.2.

```
gap> CHEVIE.PrintBraid := "Deligne";;
gap> W := CoxeterGroup( "A", 3 );;
gap> B := Braid( W );
function ( arg ) ... end
gap> B( W.generators[1] );
1
gap> B( 2, 1, 2, 1, 1 );
121.1.1
gap> B( [ 2, 1, 2, 1, 1 ], -1 );
w0^-1.121.1.1
```

This function requires the package "chevie"(see ??).

6.2 Operations for braid elements

A braid element $\mathbf{b}$ corresponding to the Coxeter group W is represented as a record with fields `pw0`, `elm` and `operations` which contains the operations described below. All examples below are with `CHEVIE.PrintBraid="Deligne"`.

*b1 * b2*

The multiplication of two braid elements is defined, if they are braid elements of the same group, returning a braid element.

```
gap> W := CoxeterGroup( "A", 2 );;
gap> a := Braid( W )( [1] );
1
gap> b := Braid( W )( [2] );
2
gap> a * b;
12
```

b1 ^ i

A braid element can be raised to an integral, positive or negative, power, returning a braid element.

```
gap> ( a * b ) ^ 4;
w0^2.12
gap> ( a * b ) ^ -1;
w0^-1.2
```

b1 ^ *b2*

This returns $b_1^{-1}b_2b_1$.

```
gap> a ^ b;
w0^-1.21.12
```

b1 / *b2*

This returns $b_1b_2^{-1}$.

```
gap> a / b;
w0^-1.2.21
```

`CoxeterGroup(` *b* `)`

This function returns the Coxeter group for which b is a braid element.

```
gap> CoxeterGroup( a );
CoxeterGroup("A", 2)
```

`String(` *b* `)`

`Print(` *b* `)`

`String` returns a display form of the element b, and `Print` prints the result of `String`. The way elements are printed depends on the global variable `CHEVIE.PrintBraid`. If set to `"GAP"`, the elements are printed in a form which can be read in back by the function `Braid(W)`. If set to `"Deligne"` (resp. `"Charney"`) the Deligne (resp. Charney) normal form (as explained in the introduction) is printed:

```
gap> CHEVIE.PrintBraid := "GAP";;
gap> ( a * b ) ^ -1;
B([ 2 ],-1)
gap> CHEVIE.PrintBraid := "Charney";;
gap> ( a * b ) ^ -1;
(12)^-1.
gap> CHEVIE.PrintBraid := "Deligne";;
gap> ( a * b ) ^ -1;
w0^-1.2
```

`Frobenius(` *WF* `)(` *b* `)`

The Frobenius of a Coxeter coset associated to `CoxeterGroup(`*b*`)` can be applied to b. See the Chapter on Coxeter cosets.

This function requires the package "chevie"(see ??).

6.3 PermBraid

`PermBraid(` *b* `)`

Returns the image of the braid element b in the Coxeter group `CoxeterGroup(`*b*`)` of which b is a braid element.

```
gap>  W := CoxeterGroup( "A", 3 );;
gap>  b := Braid( W )( [ 2, 1, 2, 1, 1 ] );
121.1.1
gap> p := PermBraid( b );
```

```
( 1, 8)( 2, 7)( 3, 6)( 4,10)( 9,12)
gap> CoxeterWord( W, p );
[ 1, 2, 1 ]
```

This function requires the package "chevie" (see ??).

6.4 WordBraid

`WordBraid(` *b* `)`

b is a braid element which must be in the braid monoid. `WordBraid` returns as a list of indices the list of generators of the braid monoid $B(W)$ of which b is an element (the group W can be obtained by `CoxeterGroup(b)`).

```
gap>  W := CoxeterGroup( "A", 3 );;
gap>  b := Braid( W )( [ 2, 1, 2, 1, 1 ] );
121.1.1
gap> WordBraid( b );
[ 1, 2, 1, 1, 1 ]
```

This function requires the package "chevie" (see ??).

6.5 GoodCoxeterWord

`GoodCoxeterWord(` W, w `)`

Let W be a Coxeter group with associated braid monoid B^+. `GoodCoxeterWord` checks if the element w of W (given as sequence of generators of W) represents a "good element" in the sense of Geck-Michel [GM97] of the braid monoid, i.e., if $\mathbf{w}^d$ (where d is the order of the element w in W, and $\mathbf{w}$ is the element of B^+_{red} with image w) is a product of (the braid elements corresponding to) longest elements in a decreasing chain of parabolic subgroups of W. If this is true, then a list of couples, the corresponding subsets of the generators with their multiplicities in the chain, is returned. Otherwise, `false` is returned.

Good elements have nice properties with respect to their eigenvalues in irreducible representations of the Hecke-Iwahori algebra associated to W. The representatives in the component `classtext` of `ChevieClassInfo(`W`)` are all good elements of minimal length in their class.

```
gap> W := CoxeterGroup( "F", 4 );;
gap> w:=[ 2, 3, 2, 3, 4, 3, 2, 1, 3, 4 ];;
gap> GoodCoxeterWord( W, w );
[ [ [ 1 .. 4 ], 2 ], [ [ 3, 4 ], 4 ] ]
gap> OrderPerm( PermCoxeterWord( W, w ) );
6
gap> Braid( W )( w ) ^ 6;
w0^2.343.343.343.343
gap>  GoodCoxeterWord( W, [ 3, 2, 3, 4, 3, 2, 1, 3, 4, 2 ] );
false
```

This function requires the package "chevie" (see ??).

Chapter 7

Complex reflection groups, cyclotomic algebras

Preliminary support for complex reflection groups and cyclotomic Hecke algebras has been added to the CHEVIE package. A complex reflection group is a group W acting on a vector space V, and generated by pseudo-reflections in V. The **field of definition** of W is defined to be the field of definition of V. It turns out that, as for rational reflection groups (Weyl groups), all representations of a complex reflection group W are defined over the field of definition of W (cf. [Ben76] and D. Bessis thesis). Similarly to Coxeter groups, complex reflection groups are represented by the permutation representation on a set of `roots` in V invariant by W and such that all reflections in W are reflections with respect to some root (see 7.1). However there is no general theory on how to construct a nice set of roots for an arbitrary reflection group; the roots given in GAP where obtained case-by-case in an ad hoc way.

Irreducible complex reflection groups have been classified by Shephard and Todd. They contain one infinite family depending on 3 parameters, and 34 "exceptional"groups (which have been given by Shephard and Todd a number which actually varies from 4 to 37, and covers also the exceptional Coxeter groups, e.g., `CoxeterGroup("E",8)` is the group of Shephard-Todd number 37).

The cyclotomic Hecke algebra (see 7.3) corresponding to a complex reflection group is defined in a similar way as the Iwahori–Hecke algebra; for details see [BM93]. G. Malle has computed character tables for some of these algebras, including all 2-dimensional groups, see [BM93] and [Mal96]; CHEVIE contains those of type $G(e,1,1)$, G_4, G_5, G_6, G_8, G_9, G_{12} and G_{25} in the Shephard-Todd classification.

7.1 ComplexReflectionGroup

`ComplexReflectionGroup(` *STnumber* `)`

`ComplexReflectionGroup(` p`,` q`,` r `)`

The first form of `ComplexReflectionGroup` returns the complex reflection group which has Shephard-Todd number *STnumber*, see [ST54]. The second form returns the imprimitive complex reflection group $G(p,q,r)$.

```
gap> G := ComplexReflectionGroup( 4 );
ComplexReflectionGroup(4)
gap> ReflectionDegrees( G );
[ 4, 6 ]
gap> Size( G );
24
gap> q := X( Cyclotomics );; q.name := "q";;
gap> FakeDegrees( G, q );
[ q^0, q^8, q^4, q^7 + q^5, q^3 + q, q^5 + q^3, q^6 + q^4 + q^2 ]
```

Complex reflection groups are represented as permutation group records with the following additional fields

`roots`
: a set of complex roots in V, given as a matrix, on which W has a faithful permutation representation. `roots[1..semisimpleRank]` should be linearly independent. Roots are not always of same length, and sometimes the number of roots may be greater than the order of W!

`semisimpleRank`
: the dimension of the subspace of V generated by the roots (for an irreducible group, equal to the dimension of V).

7.2 Operations for complex reflection groups

All permutation group operations are defined on complex reflection groups. The following operations and functions have been rewritten to take advantage of the particular structure of reflection groups:

`Print`
: prints a complex reflection group in a form that can be input back in GAP.

`ReflectionDegrees`
: the list of reflection degrees.

`Size`
: Uses the product of the `ReflectionDegrees` to work faster.

`FakeDegrees(W,q)`
: The list of fake degrees as polynomials in q.

`ReflectionCharValue(W,w)`
: The value of the reflection character on the element w of W, given as a permutation of the roots.

These functions require the package "chevie"(see ??).

7.3 Hecke for complex reflection groups

`Hecke(` G, *para* `)`

returns the cyclotomic Hecke algebra corresponding to the complex reflection group G. The parameters of this algebra are specified in the variable *para*, which may be either a single value or a list of parameters.

```
gap> G := ComplexReflectionGroup( 4, 2, 3 );
ComplexReflectionGroup(4,2,3)
gap> v := X( Cyclotomics );; v.name := "v";;
gap> CH := Hecke( G, v );
Hecke(ComplexReflectionGroup(4,2,3),v)
```

This function requires the package "chevie"(see ??).

7.4 Operations for cyclotomic Hecke algebras

`Group`
: returns the complex reflection group from which the cyclotomic Hecke algebra was generated.

`Print`
: prints the cyclotomic Hecke algebra in a form which can be read back into GAP.

`SchurElements`
: returns Schur elements (analogously defined as for Iwahori–Hecke algebras) for some types of exceptional cyclotomic Hecke algebras.

`CharTable`
: returns the character table for some types of cyclotomic Hecke algebras, namely those of type $G(e,1,1)$, G_4, G_5, G_6, G_8, G_9, G_{12} and G_{25} in the Shephard-Todd classification. This is a record with exactly the same components as for the corresponding complex reflection group but where the component `irreducibles` contains the values of the irreducible characters of the algebra on certain basis elements T_w where w runs over the elements in the component `classtext`. Thus, the value are now polynomials in the parameters of the algebra.

```
gap> G := ComplexReflectionGroup( 4 );
ComplexReflectionGroup(4)
gap> v := X( Cyclotomics );; v.name := "v";;
gap> CH := Hecke( G, v );
Hecke(ComplexReflectionGroup(4),v)
gap> Display( CharTable( CH ) );
H(G4)

       2  3       3           1            1  2          1           1
       3  1       1           1            1  .          1           1

         1a      2a          3a           3b 4a         6a          6b
      2P 1a      1a          3b           3a 2a         3a          3b
      3P 1a      2a          1a           1a 4a         2a          2a

phi_{1,0}     1     v^6           v          v^2 v^3        v^2        v^10
phi_{1,4}     1       1           A           /A   1         /A           A
phi_{1,8}     1       1          /A            A   1          A          /A
phi_{2,1}     2 (-2)v^3   v+(E(3)) v^2+(E(3)^2)   .   (E(3))v (E(3)^2)v^5
phi_{2,3}     2 (-2)v^3 v+(E(3)^2)   v^2+(E(3))   . (E(3)^2)v   (E(3))v^5
phi_{2,5}     2      -2          -1           -1   .          1           1
```

```
phi_{3,2}     3  (3)v^2        v-1        v^2-1  -v           .            .

A = E(3)
  = (-1+ER(-3))/2 = b3
```

This function requires the package "chevie"(see ??).

Chapter 8

Iwahori-Hecke algebras

In this chapter we describe functions for dealing with Iwahori-Hecke algebras associated to finite Coxeter groups.

Let W be a finite Coxeter group, with generators $S = \{s_1, \ldots, s_n\}$. As before, let $m(i,j)$ denote the order of the product $s_i s_j$. Let R be a commutative ring with 1 and $q_1, \ldots, q_n$ be elements in R such that $q_i = q_j$ whenever $m(i,j)$ is odd. Thus, we have $q_i = q_j$ whenever s_i and s_j are conjugate in W. The corresponding Iwahori-Hecke algebra with parameters $\{q_i\}$ is a deformation of the group algebra of W over R where the multiplication of basis elements involves the parameters q_i and where the deformation is trivial if all q_i are equal to 1.

More precisely, the Iwahori-Hecke algebra $H = H(W, R, \{q_i\})$ is the associative R-algebra with $1 = T_1$ generated by elements $T_{s_1}, \ldots, T_{s_n}$ subject to the following relations.

$$\begin{array}{rcll} (T_{s_i} - q_i)(T_{s_i} + 1) & = & 0 & \text{for all } i \\ T_{s_i} T_{s_j} T_{s_i} \cdots & = & T_{s_j} T_{s_i} T_{s_j} \cdots & \text{for } i \neq j \text{ and with } m(i,j) \text{ factors on each side.} \end{array}$$

Since the generators T_{s_i} satisfy the braid relations, the algebra H is in fact a quotient of the group algebra of the braid group associated with W. It follows that, if $w = s_{i_1} \cdots s_{i_m} = s_{j_1} \cdots s_{j_m}$ are two reduced expressions of $w \in W$ as products of fundamental reflections then the corresponding products of the generators T_{s_i} respectively T_{s_j} will give the same element of H, which we may therefore denote by T_w. Then the elements $\{T_w \mid w \in W\}$ actually form a free R-basis of H. The multiplication of two arbitrary basis elements T_v, T_w (for $v, w \in W$) is then performed as follows. Choose a reduced expression for v, say $v = s_{i_1} \cdots s_{i_k}$. Then T_v is the product of the corresponding generators T_{s_i} hence we are reduced to the case where $v = s_i$ for some i. In this case, we have

$$T_{s_i} T_w = \begin{cases} T_{s_i w} & \text{if } l(s_i w) = l(w) + 1 \\ q_i T_{s_i w} + (q_i - 1) T_w & \text{if } l(s_i w) = l(w) - 1. \end{cases}$$

There is a universal choice for R and $\{q_i\}$: Let $u_1, \ldots, u_n$ be indeterminates over $\mathbb{Q}$ such that $u_i = u_j$ whenever $m(i,j)$ is odd, and let $A_0 = \mathbb{Z}[u_1, \ldots, u_n]$ be the corresponding polynomial ring. Then $H_0 := H(W, A_0, \{u_i\})$ is called the **generic Iwahori-Hecke algebra** associated with W. If R and $\{q_i\}$ are given as above then $H(W, R, \{q_i\})$ can be obtained by specialization from H_0: There is a unique ring homomorphism $f : A_0 \to R$ such that $f(u_i) = q_i$ for all i. Then we can view R as an A_0-module via f and we can identify $H(W, R, \{q_i\}) = R \otimes_{A_0} H_0$.

If all u_i are equal we call the corresponding algebra the one-parameter Iwahori-Hecke algebra associated with W. Certain invariants associated with the irreducible characters of this algebra play a special role in the representation theory of the underlying finite Coxeter groups, namely the a- and A-invariants which were already used in chapter 4 (see 4.8, 5.7).

For basic properties of Iwahori-Hecke algebras and their relevance to the representation theory of finite groups of Lie type, we refer to [CR87], Sections 67 and 68.

In the following example, we compute the multiplication table for the 0-Iwahori–Hecke algebra associated with the Coxeter group of type A_2.

```
gap> W := CoxeterGroup( "A", 2 );
CoxeterGroup("A", 2)
```

Algebra with all parameters equal to 0:

```
gap> H := Hecke( W, 0 );
Hecke(CoxeterGroup("A", 2),[ 0, 0 ],[  ])
```

Create the T-basis:

```
gap> T := Basis( H, "T" );
function ( arg ) ... end
gap> el := CoxeterWords( W );
[ [  ], [ 2 ], [ 1 ], [ 2, 1 ], [ 1, 2 ], [ 1, 2, 1 ] ]
```

Multiply any two T-basis elements:

```
gap> mat := []; for i in [1..6] do mat[i]:=[]; for j in [1..6] do
> Add( mat[i], T( el[i]) * T( el[j] ) ); od; od;
gap> PrintArray(mat);
[ [     T(),      T(2),      T(1),    T(2,1),    T(1,2),  T(1,2,1) ],
  [    T(2),     -T(2),    T(2,1),   -T(2,1),  T(1,2,1), -T(1,2,1) ],
  [    T(1),    T(1,2),     -T(1),  T(1,2,1),   -T(1,2), -T(1,2,1) ],
  [  T(2,1),  T(1,2,1),   -T(2,1), -T(1,2,1), -T(1,2,1),  T(1,2,1) ],
  [  T(1,2),   -T(1,2),  T(1,2,1), -T(1,2,1), -T(1,2,1),  T(1,2,1) ],
  [ T(1,2,1), -T(1,2,1), -T(1,2,1),  T(1,2,1),  T(1,2,1), -T(1,2,1) ] ]
```

Thus, we can not only work with generic algebras where the parameters are indeterminates. In the following chapter we will see that this also works on the level of characters and representations.

8.1 Hecke

`Hecke(` *W* `,` `[`*parameter(s)*`,` `[`*sqrtparameter(s)*`]]` `)`

`Hecke(` *rec* `)`

Constructs the Iwahori-Hecke algebra H of the given Coxeter group. *parameters* can be either a vector of length **`W.semisimpleRank`**, giving the parameter for each fundamental reflection, or a single value which is taken as the common parameter. Similarly, if given, *sqrtparameter* specifies the list of the roots of the parameters, or is a single value which specifies the root of the single value given as a parameter. If *parameters* are not given, they are assumed to be equal to 1. The Iwahori-Hecke algebra then degenerates to the group algebra of the Coxeter group.

The reason *sqrtparameter* may have to be given is that certain operations on the algebra require taking such square roots (like the character values of algebras of type E_7, E_8, or two-parameter G_2). Moreover, even if we had wanted to make a standard choice of a square root this would have been impossible because GAP is unable to take the square root of an arbitrary value (such as an indeterminate).

The second form takes as an argument a record which has a field `hecke` and returns the value of this field. This is used to return the Hecke algebra of objects derived from Hecke algebras, such as Hecke elements in various bases.

```
gap> W := CoxeterGroup( "B", 3 );
CoxeterGroup("B", 3)
gap> u := X( Rationals );; u.name := "u";;
```

One parameter algebra without and with specifying square roots:

```
gap> H := Hecke( W, u );
Hecke(CoxeterGroup("B", 3),[ u, u, u ],[  ])
gap> H := Hecke( W, u^2, u );
Hecke(CoxeterGroup("B", 3),[ u^2, u^2, u^2 ],[ u, u, u ])
gap> H := Hecke( W, [ u^6, u^4, u^4 ], [ u^3, -u^2, -u^2 ] );
Hecke(CoxeterGroup("B", 3),[ u^6, u^4, u^4 ],[ u^3, -u^2, -u^2 ])
```

The parameters do not have to be indeterminates:

```
gap> H := Hecke( W, 9, 3 );
Hecke(CoxeterGroup("B", 3),[ 9, 9, 9 ],[ 3, 3, 3 ])
gap> H := Hecke( W, [ u^6, u^4, u^8] );
Error, Hecke algebra parameters should be equal for conjugate
   reflections in Hecke( W, [ u^6, u^4, u^8 ] )) called from main loop
brk>
```

This function requires the package "chevie"(see ??).

8.2 HeckeSubAlgebra

`HeckeSubAlgebra(` H, r `)`

Given an Hecke Algebra H and a set of roots of `Group(`H`)` given as their index in the roots, return the Hecke sub-algebra generated by the T_s corresponding to these roots. The roots must be simple roots if any parameter is not 1.

As for Subgroup, a subalgebra of a subalgebra is given as a subalgebra of the parent algebra.

```
gap> u := X( Rationals );; u.name := "u";;
gap> H := Hecke( CoxeterGroup( "B", 2 ), u );
Hecke(CoxeterGroup("B", 2),[ u, u ],[  ])
gap> HeckeSubAlgebra( H, [ 1, 4 ] );
Hecke(ReflectionSubgroup(CoxeterGroup("B", 2), [ 1, 2 ]),[ u, u ],
[  ])
gap> HeckeSubAlgebra( H, [ 1, 7 ] );
```

```
Error, Generators of a sub-Hecke algebra should be simple reflections
in HeckeSubAlgebra( H, [ 1, 7 ] ) called from main loop
brk>
```

This function requires the package "chevie"(see ??).

8.3 Operations and functions for Hecke algebras

`Group`
: returns the Coxeter group from which the Hecke algebra was generated.

`Print`
: prints the Hecke algebra in a form which can be read back into GAP.

`SchurElements`
: see the explicit description in Sections 9.8 and 9.7.

`CharTable`
: returns the character table of the Hecke algebra. This is a record with exactly the same components as for the corresponding finite Coxeter group but where the component `irreducibles` contains the values of the irreducible characters of the algebra on basis elements T_w where w runs over the elements in the component `classtext`. Thus, the value are now polynomials in the parameters of the algebra. For more details see the chapter 9.

`Basis`
: the `T` basis is described in the section below. Other bases are described in chapter 10.

8.4 Construction of Hecke elements of the T basis

`Basis(` *H*`, "T")`

Let H be a Iwahori-Hecke algebra. The function `Basis(H,"T")` returns a function which can be used to make elements of the usual `T` basis of the algebra. It is convenient to assign this function with a shorter name when computing with elements of the Hecke algebra. In what follows we assume that we have done the assignment:

```
gap> T := Basis( H, "T" );
function ( arg ) ... end
```

`T(` *perm* `)`

Let *perm* be a permutation which is an element w of the Coxeter group `Group(H)`. This call returns the basis element T_w of H.

`T(` *perms*`,` *coeffs*`)`

In this form, *perms* is a vector of permutations and *coeffs* a vector of coefficients which should be of the same length k. The element $\mathrm{coeffs}[1]T_{\mathrm{perms}[1]} + \ldots + \mathrm{coeffs}[k]T_{\mathrm{perms}[k]}$ of H is returned.

`T(` *list* `)`

`T(` *s1*`, ..,` *sn* `)`

In the above two forms, $s_1, \ldots, s_n$ is a sequence of integers representing generators of the Coxeter group attached to H, or *list* is a GAP list of such integers. The element $T_{s_1}T_{s_2}\ldots T_{s_n}$ is returned.

```
gap> W := CoxeterGroup( "B", 3 );;
gap> u := X( Rationals );; u.name := "u";;
gap> H := Hecke( W, u );;
gap> T := Basis( H, "T" );
function ( arg ) ... end
gap> T( 1, 2 ) = T( [ 1, 2 ] );
true
gap> T( 1, 2 ) = T( PermCoxeterWord( W, [ 1, 2 ] ) );
true
gap> l := [ [], [ 1, 2, 3 ], [ 1 ], [ 2 ], [ 3 ] ];;
gap> pl := List( l, i -> PermCoxeterWord( W, i ) );;
gap> h := T( pl, [ u^100, 1/u^20, 1, -5, 0 ] );
u^100T()+T(1)-5T(2)+0T(3)+u^-20T(1,2,3)
gap> h.elm;
[ (), ( 1,16,13,10, 7, 4)( 2, 8,12,11,17, 3)( 5, 9, 6,14,18,15),
  ( 1,10)( 2, 6)( 5, 8)(11,15)(14,17),
  ( 1, 4)( 2,11)( 3, 5)( 8, 9)(10,13)(12,14)(17,18),
  ( 2, 5)( 3,12)( 4, 7)( 6, 8)(11,14)(13,16)(15,17) ]
gap> h.coeff;
[ u^100, u^(-20), 1, -5, 0 ]
```

The last two lines show that a Hecke element is represented internally by a list of elements of W and the corresponding list of coefficients of the basis elements in H.

Note that the function `T` just creates a Hecke element. In order to have a fast function which does this, the resulting element is not normalized in any way, nor is it even checked if some coefficients are zero or not (see the last line in the above example). A normal form is computed by applying the function `Normalize` to such an element (see below).

The way elements of the Iwahori-Hecke algebra are printed depends on the global variable `PrintHecke` which is a component of the global variable `CHEVIE` (see the analogous remarks in chapter 6). If set to `"GAP"`, they are printed in a way which can be input back in GAP. When you load CHEVIE, the variable is initially set to the string `""`.

```
gap> CHEVIE.PrintHecke := "GAP";
"GAP"
gap> T( pl, [ u^100, 1/u^20, 1, -5, 0 ] );
u^100*T()+T(1)-5*T(2)+0*T(3)+u^-20*T(1,2,3)
gap> CHEVIE.PrintHecke := "";
""
```

8.5 Operations for Hecke elements of the T basis

All examples below are with `CHEVIE.PrintHecke=""`.

`Hecke(` *a* `)`

returns the Hecke algebra of which *a* is an element.

a `*` *b*

The multiplication of two elements given in the `T` basis of the same Iwahori-Hecke algebra is defined, returning a Hecke element expressed in the `T` basis.

```
gap> q := X( Rationals );; q.name := "q";;
gap> H := Hecke( CoxeterGroup( "A", 2 ), q );
Hecke(CoxeterGroup("A", 2),[ q, q ],[  ])
gap> T := Basis( H, "T" );
function ( arg ) ... end
gap> ( T() + T( 1 ) ) * ( T() + T( 2 ) );
T()+T(1)+T(2)+T(1,2)
gap> T( 1 ) * T( 1 );
qT()+(q-1)T(1)
gap> T( 1, 1 ); # the same
qT()+(q-1)T(1)
```

a ^ *i*

A element of the `T` basis with a coefficient whose inverse is still a Laurent polynomial in q can be raised to an integral, positive or negative, power, returning another element of the algebra. An arbitrary element of the algebra can only be raised to a positive power.

```
gap> ( q * T( 1, 2 ) ) ^ -1;
(q^-1-2q^-2+q^-3)T()+(-q^-2+q^-3)T(1)+(-q^-2+q^-3)T(2)+q^-3T(2,1)
gap> ( T( 1 ) + T( 2 ) ) ^ -1;
Error, negative exponent implemented only for single T_w in
<rec1> ^ <rec2> called from
main loop
brk>
gap> ( T( 1 ) + T( 2 ) ) ^ 2;
2qT()+(q-1)T(1)+(q-1)T(2)+T(1,2)+T(2,1)
```

a / *b*

This is equivalent to $a * b^{-1}$.

a + *b*

a - *b*

Elements of the algebra expressed in the `T` basis can be added or subtracted, giving other elements of the algebra.

```
gap> T( 1 ) + T();
T()+T(1)
gap> T( 1 ) - T( 1 );
0
```

`Normalize(` *a* `)`

normalizes the element a; in particular, terms with zero coefficient are removed.

```
gap> h := T( [ PermCoxeterWord( CoxeterGroup( H ), [ 1 ] ),() ],
>                                                  [ 0, q^100 ] );
q^100T()+0T(1)
gap> Normalize( h );
gap> h;
q^100T()
```

`Print(` *a* `)`

prints the element a, using the form initialized in CHEVIE.PrintHecke.

`String(` *a* `)`

provides a string containing the same result that is printed with `Print`.

`Coefficient(` *a*`,` *w* `)`

Returns the coefficient of the Hecke element a on the basis element T_w. Here w can be given either as a Coxeter word or as a permutation.

`AlphaInvolution(` *a* `)`

This implements the involution on the algebra defined by $T_w \mapsto T_{w^{-1}}$.

```
gap> AlphaInvolution( T( 1, 2 ) );
T(2,1)
```

`BetaInvolution(` *a* `)`

This is only defined if all the parameters of the Iwahori-Hecke algebra are equal, and they are either equal to 1 or all sqrtParameters are bound. If v is the square root of the first parameter, it implements the involution on the algebra defined by $v \mapsto v^{-1}$ and $T_w \mapsto v^{-l(w_0)}T_{w_0 w}$.

`AltInvolution(` *a* `)`

This is only defined if all the parameters of the Iwahori-Hecke algebra are equal, and they are either equal to 1 or all sqrtParameters are bound. If v is the square root of the first parameter, it implements the involution on the algebra defined by $v \mapsto -v^{-1}$ and $T_w \mapsto (-v^{-2})^{l(w)}T_w$. Essentially it corresponds to tensoring with the sign representation.

`Frobenius(`*WF*`)(` *a* `)`

The Frobenius of a Coxeter Coset associated to `CoxeterGroup(Hecke(`*a*`))` can be applied to a. For more details see chapter 11.

These functions require the package "chevie"(see ??).

8.6 CreateHeckeBasis

`CreateHeckeBasis(`*basis*`,` *ops*`)`

creates a new basis for Hecke algebras in which to do computations. (The design of this function has benefited from conversation with Andrew Mathas, the author of the package `Specht`).

The first argument *basis* must be a unique (different from that used for other bases) character string. The second argument *ops* is a record which should contain at least two fields, *ops*`.T` and *ops*`.(`*basis*`)` which should contain :

ops`.T` a function which takes an element in the basis *basis* and converts it to the `T` basis.

ops`.(`*basis*`)` a function which takes an element in the `T` basis and converts it to the *basis* basis.

After the call to `CreateHeckeBasis`, a new field `(`*basis*`)` is added to `HeckeAlgebraOps` which contains a function to create elements of the *basis* basis. These elements will have the standard operations for Hecke elements: `+`, `-`, `*`, `^`, `=`, `Print`, `Coefficient`, plus all extra operations that the user may have specified in *ops*. It is thus possible to create a new basis which has extra operations. In addition, for any already created basis y of the algebra, the function `(`y`)` will have the added capability to convert elements from the *basis* basis to the y basis. If the user has provided a field *ops*`.(`y`)` , the function found there will be used. Otherwise, the function *ops*`.T` will be used to convert our *basis* element to the `T` basis, followed by calling the function `(`y`)` which was given in *ops* at the time the y basis was created, to convert to the y basis. The following forms of the Basis function will be accepted (as for the `T` basis):

`Basis(` H`,` *basis* `)(` *perm* `)`

`Basis(` H`,` *basis* `)(` *perms*`,` *coeffs*`)`

`Basis(` H`,` *basis* `)(` *list* `)`

`Basis(` H`,` *basis* `)(` *s1*`, ..` `,` *sn* `)`

One should note, however that for the last two forms only reduced expressions will be accepted in general.

Below is an example where the basis $t_w = q^{l(x)/2}T_w$ is created and used. Here we have set $v = q^{1/2}$. As an example of an extra operation in *ops*, we have given a method for `BetaInvolution`. If methods for one of `BetaInvolution`, `AltInvolution` are given they will be automatically called by the generic functions with the same name. Here we have set `BetaInvolution` to use the same method as used by `T` basis elements.

In order to understand the following code, one has to recall that an arbitrary Hecke element is a record; the basis elements are labeled by the Coxeter group elements in the component `elm` and the corresponding coefficients in the component `coeff`. For efficiency reasons, it is necessary to describe the conversion functions on such arbitrary Hecke elements and not just for one basis element T_w or t_w.

```
gap> CreateHeckeBasis( "t", rec(
> T := h->Basis( Hecke(h), "T" )( h.elm, List( [1 .. Length( h.elm )],
>  i->Hecke(h).sqrtParameter[1]^CoxeterLength(
>             CoxeterGroup( Hecke(h) ), h.elm[i] ) *  h.coeff[i] ) ),
>
> t := h->Basis( Hecke(h), "t" )( h.elm, List( [1 .. Length( h.elm )],
>  i->Hecke(h).sqrtParameter[1]^-CoxeterLength(
>              CoxeterGroup( Hecke(h) ), h.elm[i] ) * h.coeff[i] ) ),
>
> BetaInvolution := h->Basis( Hecke( h ),"t")(
>                    HeckeAlgebraOps.T.BetaInvolution(
>                            Basis( Hecke( h ), "T" )( h ) ) ) ) );
gap> v := X( Rationals );; v.name := "v";;
gap> H := Hecke( CoxeterGroup( "A", 3 ), v ^ 2, v );;
gap> h := Basis( H, "t" )( 3, 1, 2 );
t(1,3,2)
gap> h1 := Basis( H, "T")( h );
v^3T(1,3,2)
gap> h2 := Basis( H, "t" )( h1 );
t(1,3,2)
gap> BetaInvolution( h2 );
```

```
v^-12t(2,1,3)
```

This function requires the package "chevie"(see **??**).

Chapter 9

Representations of Iwahori-Hecke algebras

Let W be a finite Coxeter group with generators $\{s_1, \ldots, s_n\}$, and $H = H(W, R, \{q_i\})$ a corresponding Iwahori-Hecke algebra over the ring R as defined in chapter 8. We shall now describe functions for dealing with representations and characters of H.

The fact that the algebra H is given by a presentation makes it particularly easy to work with representations. Assume we are given any set of matrices $M_1, \ldots, M_n \in R^{d\times d}$. The fact that H is given by generators and defining relations immediately implies that there is a (unique) representation $\rho : H \to R^{d\times d}$ such that $\rho(T_{s_i}) = M_i$ for all i, if and only if the matrices M_i satisfy the same relations as those for the generators T_{s_i} of H.

A general approach for the construction of representations is in terms of W-graphs, see [KL79, p.165]. Any such W-graph carries a representation of H. Note that, for these purposes, it is necessary to assume that the parameters of H are squares of some elements of the ground ring. The simplest example, the standard W-graph defined in [KL79, Ex. 6.2] yields a "deformation of the natural reflection representation of W. This can be produced in CHEVIE using the function `HeckeReflectionRepresentation`.

Another possibility to construct W-graphs is by using the Kazhdan-Lusztig theory of left cells (see [KL79]); see the following chapter for more details.

In a similar way as the ordinary character table of the finite Coxeter group W is defined, one also has a character table for the Iwahori-Hecke algebra H in the case when the ground ring A is a field such H is split and semisimple. The generic choice for such a ground ring is the rational function field $K = \overline{\mathbb{Q}}(v_1, \ldots, v_n)$ where the parameters of the corresponding algebra H_K are given by $q_i = v_i^2$ for all i.

By Tits' Deformation Theorem (see [CR87, Sec. 68], for example), the algebra H_K is (abstractly) isomorphic to the group algebra of W over K. Moreover, we have a bijection between the irreducible characters of H_K and W, given as follows. Let χ be an irreducible character of H_K. Then we have $\chi(T_w) \in A$ where $A = \overline{\mathbb{Z}}[v_1, \ldots, v_n]$ and $\overline{\mathbb{Z}}$ denotes the ring of algebraic integers in $\overline{\mathbb{Q}}$. We can find a ring homomorphism $f\colon A \to \overline{\mathbb{Q}}$ such that $f(a) = a$ for all $a \in \overline{\mathbb{Z}}$ and $f(v_i) = 1$ for $i = 1, \ldots, n$. Then it turns out that the function

$\chi_f: w \mapsto f(\chi(T_w))$ is an irreducible character of W, and the assignment $\chi \mapsto \chi_f$ defines a bijection between the irreducible characters of H_K and W.

Now this bijection does depend on the choice of f. But one should keep in mind that this only plays a role in the case where W is a non-crystallographic Coxeter group. In all other cases, as is well-known, the character table of W is rational; moreover, the values of the irreducible characters of H_K at basis elements T_w lie in the ring $\mathbb{Z}[v_1, \ldots, v_n]$.

The character table of H_K is defined to be the square matrix $(\chi(T_w))$ where χ runs over the irreducible characters of H_K and w runs over a set of representatives of *minimal length* in the conjugacy classes of W. The character tables of Iwahori-Hecke algebras (in this sense) are known for all types: the table for type A was first computed by Starkey (see the description of his work in [Car86]); then different descriptions with different proofs were given in [Ram91] and [Pfe94b]. The tables for the non crystallographic types $I_2(m)$, H_3, H_4 can be constructed from the explicit matrix representations given in [CR87, Sec. 67C], [Lus81] and [AL82], respectively. For the classical types B and D see [HR94] and [Pfe96]. The tables for the remaining exceptional types were computed in [Gec94], [Gec95] and [GM97].

If H is an Iwahori-Hecke algebra over an arbitrary ground ring R as above, then the GAP function `CharTable` applied to the corresponding record returns a character table record which is build up in exactly the same way as for the finite Coxeter group W itself but where the record component `irreducibles` contains the character values which are obtained from those of the generic multi-parameter algebra H_K by specializing the indeterminates v_i to the variables in `sqrtParameters`.

9.1 HeckeReflectionRepresentation

`HeckeReflectionRepresentation(` W `)`

returns a list of matrices which give the reflection representation of the Iwahori-Hecke algebra corresponding to the Coxeter group W. The function `Hecke` must have been applied to the record W.

```
gap> v:= X( Rationals );;  v.name := "v";;
gap> H := Hecke(CoxeterGroup( "B", 2) , v^2, v);
Hecke(CoxeterGroup("B", 2),[ v^2, v^2 ],[ v, v ])
gap> ref:= HeckeReflectionRepresentation( H );
[ [ [ -v^0, 0*v^0 ], [ -v^2, v^2 ] ],
  [ [ v^2, -2*v^0 ], [ 0*v^0, -v^0 ] ] ]
gap> H := Hecke( CoxeterGroup( "H", 3 ));;
gap> HeckeReflectionRepresentation( H );
[ [ [ -1, 0, 0 ], [ -1, 1, 0 ], [ 0, 0, 1 ] ],
  [ [ 1, E(5)+2*E(5)^2+2*E(5)^3+E(5)^4, 0 ], [ 0, -1, 0 ],
      [ 0, -1, 1 ] ], [ [ 1, 0, 0 ], [ 0, 1, -1 ], [ 0, 0, -1 ] ] ]
```

This function requires the package "chevie"(see ??).

9.2 CheckHeckeDefiningRelations

`CheckHeckeDefiningRelations(` H , t `)`

returns true or false, according to whether a given set t of matrices corresponding to the standard generators of the Coxeter group *Group(H)* defines a representation of the Iwahori-Hecke algebra H or not.

```
gap> H := Hecke(CoxeterGroup( "F", 4 ));;
gap> r := HeckeReflectionRepresentation( H );;
gap> CheckHeckeDefiningRelations( H, r );
true
```

This function requires the package "chevie"(see ??).

9.3 CharTable for Hecke algebras

`CharTable(` H `)`

`CharTable` returns the character table record of the Iwahori-Hecke algebra H. This is basically the same as the character table of a Coxeter group described earlier with the exception that the component `irreducibles` contains the matrix of the values of the irreducible characters of the generic Iwahori-Hecke algebra specialized at the parameters in the component `parameter` of H. Thus, if all these parameters are equal to $1 \in \mathbb{Q}$ then the component `irreducibles` just contains the ordinary character table of the underlying Coxeter group.

The function `CharTable` first recognizes the type of H, then calls special functions for each type involved in H and finally forms the direct product of all these tables.

```
gap> W := CoxeterGroup( "G", 2 );;
gap> u := X( Rationals );;  u.name := "u";;
gap> v := X( LaurentPolynomialRing( Rationals ) );; v.name := "v";;
gap> u := u * v^0;;
gap> H := Hecke( W, [ u^2, v^2 ], [ u, v ] );
Hecke(CoxeterGroup("G", 2),[ u^2*v^0, v^2 ],[ u*v^0, v ])
gap> Display( CharTable( H ) );
H(G2)

             2  2         2      2          1         1         2
             3  1         .      .          1         1         1

                       ~A_1    A_1        G_2       A_2 A_1 + ~A_1
            2P                            A_2       A_2
            3P         ~A_1    A_1 A_1 + ~A_1           A_1 + ~A_1

phi_{1,0}       1         v^2   (u^2)   (u^2)v^2  (u^4)v^4   (u^6)v^6
phi_{1,6}       1          -1      -1          1         1          1
phi_{1,3}'      1         v^2      -1       -v^2       v^4       -v^6
phi_{1,3}''     1          -1   (u^2)     (-u^2)     (u^4)     (-u^6)
phi_{2,1}     (2)    v^2+(-1) (u^2-1)       (u)v (-u^2)v^2 (-2u^3)v^3
phi_{2,2}     (2)    v^2+(-1) (u^2-1)      (-u)v (-u^2)v^2  (2u^3)v^3
```

As mentioned before, the record components `classparam`, `classnames` and `irredinfo` contain canonical labels and parameters for the classes and the characters (see the introduction to chapter 4 and also 4.10). For direct products, sequences of such canonical labels of the individual types are given.

We can also have character tables for algebras where the parameters are not necessarily indeterminates:

```
gap> H1 := Hecke( W, [ E(6)^2, E(6)^4 ],[ E(6), E(6)^2 ] );
Hecke(CoxeterGroup("G", 2),[ E(3), E(3)^2 ],[ -E(3)^2, E(3) ])
gap> ct := CharTable( H1 );
CharTable( "H(G2)" )
gap> Display( ct );
H(G2)
              2 2    2   2          1   1          2
              3 1    .   .          1   1          1

                   ~A_1 A_1       G_2 A_2 A_1 + ~A_1
             2P                   A_2 A_2
             3P    ~A_1 A_1 A_1 + ~A_1    A_1 + ~A_1

phi_{1,0}      1    A  /A           1   1          1
phi_{1,6}      1   -1  -1           1   1          1
phi_{1,3}'     1    A  -1          -A  /A         -1
phi_{1,3}''    1   -1  /A         -/A   A         -1
phi_{2,1}      2    B  /B          -1  -1          2
phi_{2,2}      2    B  /B           1  -1         -2

A = E(3)^2
  = (-1-ER(-3))/2 = -1-b3
B = E(3)+2*E(3)^2
  = (-3-ER(-3))/2 = -2-b3
gap> RankMat( ct.irreducibles );
5
```

The last result tells us that the specialized character table is no more invertible.

Character tables of Iwahori–Hecke algebras were introduced in [GP93]; see also the introduction to this chapter for further information about the origin of the various tables.

This function requires the package "chevie"(see ??).

9.4 HeckeCharValues

`HeckeCharValues(` T `[,`*irreds*`])`

T is an element of an Iwahori-Hecke algebra (expressed in any basis) and *irreds* is a set of irreducible characters of the algebra (given as vectors). `HeckeCharValues` returns the values of *irreds* on the element T (the method used is to convert to the `T` basis, and then use `HeckeClassPolynomials`). If *irreds* is not given, all character values are returned.

```
gap> q := X( Rationals );; q.name := "q";;
gap> H := Hecke( CoxeterGroup( "B", 2 ), q ^ 2, q );;
gap> HeckeCharValues( Basis( H, "C'" )( 1, 2, 1 ) );
[ 0*q^0, q + q^(-1), 0*q^0, q^3 + 2*q + 2*q^(-1) + q^(-3),
  -q - q^(-1) ]
```

See also 9.5.

This function requires the package "chevie" (see ??).

9.5 HeckeClassPolynomials

`HeckeClassPolynomials(` *h* `[,` *reps*`] )`

returns the class polynomials of the Hecke element h of the Hecke algebra H with respect to representatives *reps* of minimal length in the conjugacy classes of the Coxeter group *Group(H)*.

If absent, *reps* is taken as `CoxeterConjugacyClasses(Group(H))`. These polynomials have the following property. Given the class polynomials p corresponding to h and the matrix X of the values of the irreducible characters of the Iwahori-Hecke algebra on T_w (for w in *reps*), then the product `X*p` is the list of values of the irreducible characters on the element h of the Iwahori-Hecke algebra.

```
gap> u := X( Rationals );; u.name := "u";;
gap> W := CoxeterGroup( "A", 3 );
CoxeterGroup("A", 3)
gap> H := Hecke( W, u );;
gap> h := Basis( H, "T" )( LongestCoxeterElement( W ) );
T(1,2,1,3,2,1)
gap> cp := HeckeClassPolynomials( h );
[ 0*u^0, 0*u^0, u^2, u^3 - 2*u^2 + u, u^3 - u^2 + u - 1 ]
gap> CharTable( H ).irreducibles * cp;
[ u^0, -u^2, 2*u^3, -u^4, u^6 ]
```

So, the entries in this list are the values of the irreducible characters on the basis element corresponding to the longest element in the Coxeter group.

The class polynomials were introduced in [GP93].

This function requires the package "chevie" (see ??).

9.6 PoincarePolynomial

`PoincarePolynomial(` *H* `)`

The Poincaré polynomial of the Hecke algebra H, which is equal to `SchurElements(`*H*`)[`*ind*`]` where *ind* is the position of the 1-dimensional index representation in the character table of H, that is, the representation which maps T_{s_i} to the corresponding parameter q_i.

```
gap> q := X( Rationals );; q.name := "q";;
gap> W := CoxeterGroup( "G", 2 );; H := Hecke( W, q );
Hecke(CoxeterGroup("G", 2),[ q, q ],[  ])
gap> PoincarePolynomial( H );
q^6 + 2*q^5 + 2*q^4 + 2*q^3 + 2*q^2 + 2*q + 1
```

This function requires the package "chevie" (see ??).

9.7 SchurElements

`SchurElements(` *H* `)`

returns the list of constants arising from the Schur relations for the irreducible characters of the Iwahori-Hecke algebra H.

The Schur element corresponding to an irreducible character χ is also equal to P/D_χ where P is the Poincare polynomial and D_χ is the generic degree of χ. Note, however, that this only works if $D_\chi \neq 0$. (We can have $D_\chi = 0$ if the parameters of H are suitably chosen roots of unity, for example.) The ordering of the degrees corresponds to the ordering of the characters as returned by the function `CharTable`. Note that the Schur element corresponding to the *ind*-character is the Poincare polynomial P.

```
gap> u := X( Rationals );; u.name := "u";;
gap> v := X( LaurentPolynomialRing( Rationals ) );; v.name := "v";;
gap> schur := SchurElements( Hecke( CoxeterGroup( "G", 2 ),
>                                   [ u ^ 2, v ^ 2 ], [ u, v ] ) );
[ (u^6 + u^4)*v^6 + (u^6 + 2*u^4 + u^2)*v^4 + (u^4 + 2*u^2 + 1)*v^
    2 + (u^2 + 1), (1 + u^(-2)) + (1 + 2*u^(-2) + u^(-4))*v^(
    -2) + (u^(-2) + 2*u^(-4) + u^(-6))*v^(-4) + (u^(-4) + u^(-6))*v^(
    -6), (u^(-4) + u^(-6))*v^6 + (u^(-2) + 2*u^(-4) + u^(-6))*v^4 + (
    1 + 2*u^(-2) + u^(-4))*v^2 + (1 + u^(-2)),
  (u^2 + 1) + (u^4 + 2*u^2 + 1)*v^(-2) + (u^6 + 2*u^4 + u^2)*v^(
    -4) + (u^6 + u^4)*v^(-6), (2*u^0)*v^2 + (2*u - 2*u^(-1))*v + (2*u^
    2 - 2 + 2*u^(-2)) + (-2*u + 2*u^(-1))*v^(-1) + (2*u^0)*v^(-2),
  (2*u^0)*v^2 + (-2*u + 2*u^(-1))*v + (2*u^2 - 2 + 2*u^(-2)) + (2*u -
    2*u^(-1))*v^(-1) + (2*u^0)*v^(-2) ]
```

The Poincaré polynomial is just the Schur element corresponding to the trivial (or index) representation:

```
gap> schur[1];
(u^6 + u^4)*v^6 + (u^6 + 2*u^4 + u^2)*v^4 + (u^4 + 2*u^2 + 1)*v^
2 + (u^2 + 1)
```

(But note that the trivial character is not always the first character!) For further information about generic degrees and connections with the representation theory of finite groups of Lie type, see [BC72] and [Car85].

This function requires the package "chevie" (see ??).

9.8 SchurElement

`SchurElement(` *H*`,` *phi* `)`

returns the constants arising from the Schur relations for the irreducible character *phi* of the Iwahori-Hecke algebra H. *phi* should be specified by its parameter (see `CharParams` in section 2.6) .

```
gap> u := X( Rationals );; u.name := "u";;
gap> v := X( LaurentPolynomialRing( Rationals ) );; v.name := "v";;
gap> H := Hecke( CoxeterGroup( "G", 2 ), [ u ^ 2, v ^ 2 ], [ u, v ] );
Hecke(CoxeterGroup("G", 2),[ u^2, v^2 ],[ u, v ])
gap> SchurElement( H, [ [ 1, 3, "'" ] ] );
(u^(-4) + u^(-6))*v^6 + (u^(-2) + 2*u^(-4) + u^(-6))*v^4 + (1 + 2*u^(
-2) + u^(-4))*v^2 + (1 + u^(-2))
```

This function requires the package "chevie" (see ??).

9.9 GenericDegrees

We do not have a function for the generic degrees of an Iwahori-Hecke algebra since they are not always defined (for example, if the parameters of the algebra are roots of unity). If we have a generic multi-parameter Iwahori-Hecke algebra H then the generic degrees are certainly defined, and they can be computed with the command:

`List( SchurElements(` H `), x -> PoincarePolynomial(` H `) / x );`

(See 9.6 and 9.8.)

9.10 HeckeCentralMonomials

`HeckeCentralMonomials(` HW `)`

Returns the scalars by which the central element $T_{w_0}^2$ acts on irreducible representations of HW.

```
gap> v := X( Cyclotomics );; v.name := "v";;
gap> H := Hecke( CoxeterGroup( "H", 3 ), v ^ 2, v );;
gap> HeckeCentralMonomials( H );
[ v^0, v^60, v^24, v^36, v^20, v^20, v^40, v^40, v^30, v^30 ]
```

This function requires the package "chevie"(see ??).

9.11 HeckeCharValuesGood

`HeckeCharValuesGood(` HW`,` w `)`

Let HW be a Hecke algebra for the Coxeter group `CoxeterGroup(`HW`)`, let w be a good element of `CoxeterGroup(`HW`)` in the sense of [GM97], and let d be the order of w.

`HeckeCharValuesGood` computes the values of the irreducible characters of the Iwahori-Hecke algebra HW on T_w^d. The point is that the character table of the Hecke algebra is not used, and that all the eigenvalues of T_w^d are monomials in `HW.parameters`, so this can be used to find the absolute value of the eigenvalues of T_w, a step towards computing the character table of the Hecke algebra.

We continue the example in `HeckeCentralMonomial`:

```
gap> HeckeCharValuesGood( H, [ 1, 2, 3 ] );
[ v^0, v^60, 5*v^24, 5*v^36, 3*v^20, 3*v^20, 3*v^40, 3*v^40, 4*v^30,
  4*v^30 ]
```

This function requires the package "chevie"(see ??).

Chapter 10

Kazhdan-Lusztig polynomials and bases

There is a number of programs in CHEVIE for computing Kazhdan-Lusztig polynomials, left cells, and the various Kazhdan-Lusztig bases of Iwahori-Hecke algebras (see [KL79]).

From a computational point of view, Kazhdan-Lusztig polynomials are quite a challenge. It seems that the best approach still is by using the recursion formula in the original article [KL79]. One can first run a number of standard checks on a given pair of elements to see if the computation of the corresponding polynomial can be reduced to a similar computation for elements of smaller length, for example. One such check involves the notion of critical pairs (cf. [Alv87]):We say that a pair of elements $w_1, w_2 \in W$ such that $w_1 \leq w_2$ is critical if $\mathcal{L}(w_2) \subseteq \mathcal{L}(w_1)$ and $\mathcal{R}(w_2) \subseteq \mathcal{R}(w_1)$, where $\mathcal{L}$ and $\mathcal{R}$ denote the left and right descent set, respectively.

Now if $y, w \in W$ are arbitrary elements with $y \leq w$ then there always exists a critical pair (w_1, w_2) such that the Kazhdan-Lusztig polynomials $P_{y,w}$ and P_{w_1,w_2} are equal. Given two elements y and w, such a critical pair is found by the function `CriticalPair`.

The CHEVIE programs for computing Kazhdan-Lusztig polynomials are organized in such a way that whenever the polynomial corresponding to a critical pair is computed then this pair and the polynomial are stored in the component `criticalPairs` of the record of the underlying Coxeter group. (This is different to earlier versions of CHEVIE.)

A good example to see how long the programs will take for computations in big Coxeter groups is the following:

```
LeftCells( CoxeterGroup( "F", 4 ) );
```

which takes 20 minutes cpu time on a SPARCstation 5 computer. The computation of all Kazhdan-Lusztig polynomials for type F_4 takes a bit more than 1 hour.

However, it still seems to be out of range to re-do Alvis' computation of the Kazhdan–Lusztig polynomials of the Coxeter group of type H_4 in a computer algebra system like GAP. For such applications, it is probably more efficient to use a special purpose program like the one provided by F. DuCloux [DuC91].

The code for the Kazhdan-Lusztig bases `C`, `D` and their primed versions has been written by Andrew Mathas. Some other bases (e.g., the Murphy basis) can be found in his package `Specht`.

10.1 KazhdanLusztigPolynomial

`KazhdanLusztigPolynomial(` *W*, *y*, *w* `[,` *ly*, *lw* `] )`

returns the Kazhdan-Lusztig polynomial in the indeterminate `X(Rationals)` corresponding to the elements y and w (given as permutations) of the Coxeter group W. The optional variables ly and lw contain the length of y and w, respectively. The above form for the input has been chosen for efficiency reasons. If one prefers to give as input just two reduced expressions, one can define a new function as follows (for example):

```
gap> klpol := function( W, x, y)
>      return KazhdanLusztigPolynomial( W, PermCoxeterWord( W, x ),
>                   PermCoxeterWord( W, y ), Length( x ), Length( y ) );
>    end;
function ( W, x, y ) ... end
```

We use this function in the following example where we compute the polynomials $P_{1,w}$ for all elements w in the Coxeter group of type A_3.

```
gap> q := X( Rationals );; q.name := "q";;
gap> W := CoxeterGroup( "B", 3 );;
gap> el := CoxeterWords( W );
[ [  ], [ 3 ], [ 2 ], [ 1 ], [ 3, 2 ], [ 2, 1 ], [ 2, 3 ], [ 1, 3 ],
  [ 1, 2 ], [ 2, 1, 2 ], [ 3, 2, 1 ], [ 2, 3, 2 ], [ 2, 1, 3 ],
  [ 1, 2, 1 ], [ 1, 3, 2 ], [ 1, 2, 3 ], [ 3, 2, 1, 2 ],
  [ 2, 1, 2, 3 ], [ 2, 3, 2, 1 ], [ 2, 1, 3, 2 ], [ 1, 2, 1, 2 ],
  [ 1, 3, 2, 1 ], [ 1, 2, 1, 3 ], [ 1, 2, 3, 2 ], [ 3, 2, 1, 2, 3 ],
  [ 2, 1, 2, 3, 2 ], [ 2, 3, 2, 1, 2 ], [ 2, 1, 3, 2, 1 ],
  [ 1, 3, 2, 1, 2 ], [ 1, 2, 1, 2, 3 ], [ 1, 2, 1, 3, 2 ],
  [ 1, 2, 3, 2, 1 ], [ 2, 3, 2, 1, 2, 3 ], [ 2, 1, 2, 3, 2, 1 ],
  [ 2, 1, 3, 2, 1, 2 ], [ 1, 3, 2, 1, 2, 3 ], [ 1, 2, 1, 2, 3, 2 ],
  [ 1, 2, 1, 3, 2, 1 ], [ 1, 2, 3, 2, 1, 2 ], [ 2, 1, 2, 3, 2, 1, 2 ],
  [ 2, 1, 3, 2, 1, 2, 3 ], [ 1, 2, 3, 2, 1, 2, 3 ],
  [ 1, 2, 1, 2, 3, 2, 1 ], [ 1, 2, 1, 3, 2, 1, 2 ],
  [ 2, 1, 2, 3, 2, 1, 2, 3 ], [ 1, 2, 1, 2, 3, 2, 1, 2 ],
  [ 1, 2, 1, 3, 2, 1, 2, 3 ], [ 1, 2, 1, 2, 3, 2, 1, 2, 3 ] ]
gap> List( el, w -> klpol( W, [], w ) );
[ q^0, q^0, q^0, q^0, q^0, q^0, q^0, q^0, q^0, q^0, q^0, q^0, q^0,
  q^0, q^0, q^0, q^0, q^0, q^0, q + 1, q^0, q^0, q^0, q^0, q + 1,
  q^0, q^0, q + 1, q^0, q^0, q + 1, q + 1, q^0, q + 1, q^0, q + 1,
  q^0, q^2 + 1, q + 1, q^2 + q + 1, q + 1, q + 1, q^0, q^0, q^2 + 1,
  q^0, q + 1, q^0 ]
```

The set of Kazhdan–Lusztig polynomials that were found during this computation is contained in the record component `klpol` (as lists of coefficients):

```
gap> W.klpol;
[ [ 1, 1 ], [ 1 ], [ 1, 0, 1 ], [ 1, 1, 1 ] ]
```

Thus, we have found four different Kazhdan-Lusztig polynomials, namely $1+q$, 1, $1+q^2$ and $1+q+q^2$.

This function requires the package "chevie" (see ??).

10.2 CriticalPair

`CriticalPair(` W, y, w, ly `)`

Given an element y of length ly in the Coxeter group W and an element w the function `CriticalPair` returns a triple (z, w, lz) where (z, w) is a critical pair (i.e., we have $\mathcal{L}(w) \subseteq \mathcal{L}(z)$ and $\mathcal{R}(w) \subseteq \mathcal{R}(z)$ and lz is the length of z. This critical pair is chosen so that the corresponding Kazhdan–Lusztig polynomials $P_{z,w}$ and $P_{y,w}$ are equal.

```
gap> W := CoxeterGroup( "F", 4 );
CoxeterGroup("F", 4)
gap> w := LongestCoxeterElement( W ) * W.generators[1];;
gap> CoxeterLength( W, w );
23
gap> y := PermCoxeterWord( W, [ 1, 2, 3, 4 ] );;
gap> cr := CriticalPair( W, y, w, 4 );;
gap> CoxeterWord( W, cr[1] );
[ 2, 3, 2, 1, 3, 4, 3, 2, 1, 3, 2, 3, 4, 3, 2, 3 ]
gap> cr[3];
16
gap> KazhdanLusztigPolynomial( W, y, w, 4, 23 );
q^3 + 1
gap> KazhdanLusztigPolynomial( W, cr[1], cr[2], 16, 23 );
q^3 + 1
```

This function requires the package "chevie" (see ??).

10.3 KazhdanLusztigCoefficient

`KazhdanLusztigCoefficient(` W, y, w, `[` ly, lw,`]` k `)`

returns the k-th coefficient of the Kazhdan-Lusztig polynomial corresponding to the elements y and w, which must be given as permutations on the root vectors, of the Coxeter group W. Again, the optional variables ly and lw contain the length of y and w, respectively.

```
gap> W := CoxeterGroup( "B", 4 );;
gap> y := [ 1, 2, 3, 4, 3, 2, 1 ];;
gap> py := PermCoxeterWord( W, y );
( 1,28)( 2,15)( 4,27)( 6,16)( 7,24)( 8,23)(11,20)(12,17)(14,30)(18,31)
(22,32)
gap> x := [ 1 ];;
gap> px := PermCoxeterWord( W, x );
( 1,17)( 2, 8)( 6,11)(10,14)(18,24)(22,27)(26,30)
gap> Bruhat( W, px, py );
true
gap> List([0..3],i->KazhdanLusztigCoefficient( W, px, py, 1, 7, i ) );
[ 1, 2, 1, 0 ]
```

So the Kazhdan-Lusztig polynomial corresponding to x and y is $1 + 2u + u^2$.

This function requires the package "chevie"(see ??).

10.4 KazhdanLusztigMue

`KazhdanLusztigMue(` W, y, w `[,` ly, lw `] )`

given elements y and w in the Coxeter group W, this function returns the coefficient of degree $(l(w) - l(y) - 1)/2$ of the Kazhdan-Lusztig polynomial corresponding to y and w. The optional variables ly and lw contain the length of y and w, respectively.

Of course, the result of this function could also be obtained by

`KazhdanLusztigCoefficient(` W, y, w, ly, lw, `(`lw `-` ly `-1)/2)`

but there are some speed-ups compared to this general function.

This function requires the package "chevie"(see ??).

10.5 LeftCells

`LeftCells(` W `)`

returns a list of pairs. The first component of each pair consists of the reduced words in the Coxeter group W which lie in one left cell C, the second component consists of the corresponding matrix of highest coefficients $\mu_{y,w}$, where y, w are in C.

```
gap> W := CoxeterGroup( "G", 2 );;
gap> LeftCells(W);
[ [ [ [  ] ], [ [ 0 ] ] ],
  [ [ [ 1 ], [ 2, 1 ], [ 1, 2, 1 ], [ 2, 1, 2, 1 ], [ 1, 2, 1, 2, 1 ]
         ],
      [ [ 0, 1, 0, 0, 0 ], [ 1, 0, 1, 0, 0 ], [ 0, 1, 0, 1, 0 ],
          [ 0, 0, 1, 0, 1 ], [ 0, 0, 0, 1, 0 ] ] ],
  [ [ [ 1, 2, 1, 2, 1, 2 ] ], [ [ 0 ] ] ],
  [ [ [ 2 ], [ 1, 2 ], [ 2, 1, 2 ], [ 1, 2, 1, 2 ], [ 2, 1, 2, 1, 2 ]
         ],
      [ [ 0, 1, 0, 0, 0 ], [ 1, 0, 1, 0, 0 ], [ 0, 1, 0, 1, 0 ],
          [ 0, 0, 1, 0, 1 ], [ 0, 0, 0, 1, 0 ] ] ] ]
```

This function requires the package "chevie"(see ??).

10.6 LeftCellRepresentation

`LeftCellRepresentation(` W `,` *cell* `)`

returns a list of matrices giving the left cell representation of the Iwahori-Hecke algebra W. The argument *cell* is a pair with first component a list of reduced words which form a left cell, and second component the corresponding matrix of highest coefficients of the corresponding Kazhdan-Lusztig polynomials. Typically, *cell* is an entry from the result of the function `LeftCells`.

```
gap> v := X( Cyclotomics ) ;; v.name := "v";;
gap> W := Hecke(CoxeterGroup( "H", 3), v^2, v );
```

```
Hecke(CoxeterGroup("H", 3),[ v^2, v^2, v^2 ],[ v, v, v ])
gap> c := LeftCells( CoxeterGroup( W ) );;
gap> List( c, i -> Length( i[ 1 ] ) );
[ 1, 6, 5, 8, 5, 6, 1, 5, 8, 5, 5, 6, 6, 5, 8, 5, 5, 8, 5, 6, 6, 5 ]
gap> LeftCellRepresentation(W,c[3]);
[ [ [ -v^0, v, 0*v^0, 0*v^0, 0*v^0 ],
      [ 0*v^0, v^2, 0*v^0, 0*v^0, 0*v^0 ],
      [ 0*v^0, v, -v^0, v, 0*v^0 ],
      [ 0*v^0, 0*v^0, 0*v^0, v^2, 0*v^0 ],
      [ 0*v^0, 0*v^0, 0*v^0, 0*v^0, v^2 ] ],
  [ [ v^2, 0*v^0, 0*v^0, 0*v^0, 0*v^0 ], [ v, -v^0, v, 0*v^0, 0*v^0 ],
      [ 0*v^0, 0*v^0, v^2, 0*v^0, 0*v^0 ],
      [ 0*v^0, 0*v^0, v, -v^0, v ],
      [ 0*v^0, 0*v^0, 0*v^0, 0*v^0, v^2 ] ],
  [ [ -v^0, v, 0*v^0, 0*v^0, 0*v^0 ],
      [ 0*v^0, v^2, 0*v^0, 0*v^0, 0*v^0 ],
      [ 0*v^0, 0*v^0, v^2, 0*v^0, 0*v^0 ],
      [ 0*v^0, 0*v^0, 0*v^0, v^2, 0*v^0 ],
      [ 0*v^0, 0*v^0, 0*v^0, v, -v^0 ] ] ]
```

This function requires the package "chevie"(see ??).

10.7 Hecke elements of the C basis

`Basis(` H`, "C")`

returns a function which gives the C-basis of the (one parameter generic) Iwahori-Hecke algebra H. This is defined as follows (see [Lus85, (5.1)], for example). Let W be the underlying finite Coxeter group. For $x, y \in W$ let $P_{x,y}$ be the corresponding Kazhdan–Lusztig polynomial. If $\{T_w \mid w \in W\}$ denotes the usual T-basis and $u = v^2$ the parameter for H, then

$$C_x := \sum_{y \leq x} (-1)^{l(x)-l(y)} P_{x,y}(v^{-2}) v^{l(x)-2l(y)} T_y \quad \text{for every } x \in W.$$

For example, we have $C_s = v^{-1}T_s - vT_1$ for $s \in S$. Thus, the transformation matrix between the T-basis and the C-basis is lower unitriangular, with powers of v along the diagonal. The multiplication rules for this new basis are given as follows.

$$C_s \cdot C_x = \begin{cases} -(v+v^{-1})C_x & \text{, if } sx < x \\ C_{sx} + \sum_y \mu(y,x) C_y & \text{, if } sx > x \end{cases}$$

where the sum is over all y such that $y < x$, $l(y) \not\equiv l(x) \bmod 2$ and $sy < y$. The coefficient $\mu(y,x)$ is the coefficient of degree $(l(x) - l(y) - 1)/2$ in the Kazhdan–Lusztig polynomial $P_{x,y}$.

```
gap> W := CoxeterGroup( "B", 3 );;
gap> v := X( Rationals );; v.name := "v";;
gap> H := Hecke( W, v^2, v );
Hecke(CoxeterGroup("B", 3),[ v^2, v^2, v^2 ],[ v, v, v ])
gap> T := Basis( H, "T" );
```

```
function ( arg ) ... end
gap> C := Basis( H, "C" );
function ( arg ) ... end
gap> T( C( 1 ) );
-vT()+v^-1T(1)
gap> C( T( 1 ) );
v^2C()+vC(1)
```

We can also compute character values on elements in the C-basis as follows:

```
gap> ref := HeckeReflectionRepresentation( H );;
gap> c := CharRepresentationWords( ref, CoxeterConjugacyClasses( W ) );
[ 3, 2*v^2 - 1, v^8 - 2*v^4, -3*v^12, 2*v^2 - 1, v^4, v^4 - 2*v^2,
  -v^6, v^4 - v^2, 0*v^0 ]
gap> List( ChevieClassInfo( W ).classtext, i ->
>                              HeckeCharValues( C( i ), c ) );
[ 3*v^0, -v - v^(-1), 0*v^0, 0*v^0, -v - v^(-1), 2*v^0, 0*v^0, 0*v^0,
  v^0, 0*v^0 ]
```

This function requires the package "chevie" (see ??).

10.8 Hecke elements of the primed C basis

`Basis(` H`, "C'")`

returns a function which gives the C'-basis of the (one parameter generic) Iwahori-Hecke algebra H (see [Lus85, (5.1)]). This is defined by

$$C'_x := \sum_{y \leq x} P_{x,y}(v^2) v^{-l(x)} T_y \qquad \text{for every } x \in W.$$

We have $C'_x = \mathrm{Alt}(C_x)$ for all $x \in W$ (see `AltInvolution` in section 8.5).

```
gap> v := X( Rationals );; v.name := "v";;
gap> H := Hecke( CoxeterGroup( "B", 2 ), v ^ 2, v );;
gap> h := Basis( H, "C'" )( 1 );
C'(1)
gap> h2 := h * h;
(v+v^-1)C'(1)
gap> Basis( H, "T" )( h2 );
(1+v^-2)T()+(1+v^-2)T(1)
```

This function requires the package "chevie" (see ??).

10.9 Hecke elements of the D basis

`Basis(` H`, "D")`

returns a function which gives the D-basis of the (one parameter generic) Iwahori-Hecke algebra H (see [Lus85, (5.1)]). This can be defined by

$$D_x := v^{-N} C'_{xw_0} T_{w_0} \text{ for every } x \in W,$$

where N denotes the number of positive roots in the root system of W and w_0 is the longest element of W. The D-basis is dual to the C-basis with respect to the non-degenerate form $H \times H \to \mathbb{Z}[v, v^{-1}]$, $(h_1, h_2) \mapsto \tau(h_1 \cdot h_2)$ where $\tau \colon H \to \mathbb{Z}[v, v^{-1}]$ is the linear form such that $\tau(T_1) = 1$ and $\tau(T_x) = 0$ for $x \neq 1$. We have $D_x = \beta(C'_{w_0 x})$ for all $x \in W$ (see `BetaInvolution` in section 8.5).

```
gap> W := CoxeterGroup( "B", 2 );;
gap> v := X( Rationals );; v.name := "v";;
gap> H := Hecke( W, v^2, v );
Hecke(CoxeterGroup("B", 2),[ v^2, v^2 ],[ v, v ])
gap> T := Basis( H, "T" );
function ( arg ) ... end
gap> D := Basis( H, "D" );
function ( arg ) ... end
gap> D( T( 1 ) );
vD(1)-v^2D(1,2)-v^2D(2,1)+v^3D(1,2,1)+v^3D(2,1,2)-v^4D(1,2,1,2)
gap> BetaInvolution( D( 1 ) );
C'(2,1,2)
```

This function requires the package "chevie" (see ??).

10.10 Hecke elements of the primed D basis

`Basis(` H`, "D'")`

returns a function which gives the D'-basis of the (one parameter generic) Iwahori-Hecke algebra H (see [Lus85, (5.1)]). This can be defined by

$$D'_x := v^{-N} C_{x w_0} T_{w_0} \text{ for every } x \in W,$$

where N denotes the number of positive roots in the root system of W and w_0 is the longest element of W. The D'-basis is basis dual to the C'-basis with respect to the non-degenerate form $H \times H \to \mathbb{Z}[v, v^{-1}]$, $(h_1, h_2) \mapsto \tau(h_1 \cdot h_2)$ where $\tau \colon H \to \mathbb{Z}[v, v^{-1}]$ is the linear form such that $\tau(T_1) = 1$ and $\tau(T_x) = 0$ for $x \neq 1$. We have $D'_x = \mathrm{Alt}(D_x)$ for all $x \in W$ (see `AltInvolution` in section 8.5).

```
gap> W := CoxeterGroup( "B", 2 );;
gap> v := X( Rationals );; v.name := "v";;
gap> H := Hecke( W, v^2, v );
Hecke(CoxeterGroup("B", 2),[ v^2, v^2 ],[ v, v ])
gap> T := Basis( H, "T" );
function ( arg ) ... end
gap> Dp := Basis( H, "D'" );
function ( arg ) ... end
gap> AltInvolution( Dp( 1 ) );
D(1)
gap> Dp( 1 )^3;
(v+2v^-1-5v^-5-9v^-7-8v^-9-4v^-11-v^-13)D'()+(v^2+2+v^-2)D'(1)
```

This function requires the package "chevie" (see ??).

Chapter 11

Coxeter cosets

Let R be a root system in the real vector space V as in Chapter 2. Let F_0 be an automorphism of V. We say that F_0 is an **automorphism of** R if it is of finite order, if F_0 preserves the set of roots $R \subset V$ and if the adjoint automorphism F_0^* of $V^\vee$ preserves the set of coroots $R^\vee \subset V^\vee$.

An automorphism F_0 of R normalizes the reflection group $W := W(R)$. More precisely it induces an automorphism $F : W \to W$, defined by $w \mapsto F_0 w F_0^{-1}$. A **Coxeter coset** is the coset WF_0 in the group of automorphisms of V, generated by W and F_0.

Let $\Delta \subset R$ be a set of simple roots of R. Then the set ΔF_0 is again a set of simple roots. So there is a unique element $w_1 \in W$ such that $\phi = w_1 F_0$ stabilizes Δ.

A subset C of a Coxeter coset WF_0 is called a **conjugacy class** if one of the following three equivalent conditions is fulfilled:

- C is the orbit of an element in WF_0 under the conjugation action of W.
- C is a conjugacy class of $\langle W, F_0\rangle$ contained in WF_0.
- The set $\{w \in W \mid wF_0 \in C\}$ is an F-conjugacy class of W (two elements $v, w \in W$ are called F-conjugate, if and only if there exists $x \in W$ with $v = xwF(x^{-1})$).

Let us consider the map from the set of irreducible characters of $\langle W, F_0\rangle$ to the set of characters of W given by restriction. We get the following facts from Clifford theory: The characters having a nonzero value on a class in the coset WF_0 are exactly those characters which restrict to irreducible ones on W. The irreducible characters of W in the image of this map are exactly those which are fixed under the canonical action of F. Let $\chi \in \hat{W}$ be such a character and $\chi_1, \ldots, \chi_r$ the characters of $\langle W, F_0\rangle$ which restrict to χ. Then r is the smallest positive integer such that $F_0^r \in W$ and the restrictions of the χ_i, $i = 1, \ldots, r$ to the conjugacy classes of WF_0 are equal up to scalar multiplication with r-th roots of unity.

In CHEVIE we choose (following Lusztig) for each F-stable character of W a single (not canonical) extension to a character of $\langle W, F_0\rangle$, which we will call a **preferred extension**. The table of the restrictions of the preferred extensions to the coset WF_0 is called the **character table** of the coset WF_0. (See also the subsequent section on `CharTable` for Coxeter cosets.)

We define a scalar product on the class functions of a Coxeter coset WF_0 by

$$\langle \chi, \psi \rangle := \frac{1}{\mid W \mid} \sum_{w \in W} \chi(wF_0)\bar{\psi}(wF_0).$$

Then the character table of WF_0 contains an orthonormal set of class functions on WF_0.

A **subcoset** $W_1 w_0 F_0$ of WF_0 is given by a reflection subgroup W_1 of W and an element w_0 of W such that $w_0 F_0$ is an automorphism of the root system R_1 of W_1.

We then have a natural notion of **restriction** of class functions on WF_0 to class functions on $W_1 w_0 F_0$ as well as of **induction** in the other direction. These maps are adjoint with respect to the scalar product defined above (see [BMM93a], p.15).

The question of finding the conjugacy classes and character table of a Coxeter coset can be reduced to the case of irreducible root systems R: First we assume that $F_0 = \phi$, i.e., F_0 fixes a chosen set Δ of simple roots, and consider the canonical monomorphism $\langle W, F_0 \rangle \to W \rtimes \langle F \rangle$, where F is the automorphism of W defined above. It is clear that this map induces a bijection from WF_0 to $WF \subset W \rtimes \langle F \rangle$ which preserves conjugacy classes in the cosets. The preferred extensions are defined such that they factorize over this map. Let $W_1, W_2 \leq W$ be F-stable reflection subgroups of W such that $W = W_1 \times W_2$ and let F_i the restriction of F on W_i, $i = 1, 2$. Here the preferred extension is defined via restriction to the image of $W \rtimes \langle F \rangle \hookrightarrow (W_1 \rtimes \langle F_1 \rangle) \times (W_2 \rtimes \langle F_2 \rangle)$. Thus we can reduce the determination of conjugacy classes and the character table of WF_0 to the case were W is decomposed in irreducible components $W = W_1 \times \cdots \times W_k$ which are cyclically permuted by F. In this case there are natural bijections from the F-conjugacy classes of W to the F^k-conjugacy classes of W_1 as well as from the F-stable characters of W to the F^k-stable characters of W_1. The definition of preferred extensions on WF can be reduced to the definition of preferred extensions for $W_1 F^k$. So, we are reduced to the case that W is the Coxeter group of an irreducible root system and F permutes the simple roots, hence induces a graph automorphism on the corresponding Dynkin diagram. If $F = 1$ then conjugacy classes and characters coincide with those of the Coxeter group W. The nontrivial cases to consider are (the order of F is written as left exponent to the type): ${}^2A_n$, ${}^2D_n$, ${}^3D_4$, ${}^2E_6$ and ${}^2I_2(2k+1)$. (Note that the exceptional automorphisms of order 2 which permute the Coxeter generators of the Coxeter groups of type B_2, G_2, F_4 or $I_2(2k)$ do not come from automorphisms of the underlying root systems.)

In case ${}^3D_4$ the group $W \rtimes \langle F \rangle$ can be embedded into the Coxeter group of type F_4, which induces a labeling for the conjugacy classes of the coset. The preferred extension is chosen as the (single) extension with rational values. In case ${}^2D_n$ the group $W \rtimes \langle F \rangle$ is isomorphic to a Coxeter group of type B_n. This induces a canonical labeling for the conjugacy classes of the coset and allows to define the preferred extension in a combinatorial way using the labels (pairs of partitions) for the characters of the Coxeter group of type B_n. In the remaining cases the group $W \rtimes \langle F \rangle$ is in fact isomorphic to a direct product $W \times \langle w_0 F \rangle$ where w_0 is the longest element of W. So, there is a canonical labeling of the conjugacy classes and characters of the coset by those of W. The preferred extensions are defined by describing the signs of the character values on $w_0 F$.

In GAP we construct the Coxeter coset by starting from a Coxeter datum specified by the matrices of `simpleRoots` and `simpleCoroots`, and giving in addition the matrix `F0Mat` of the map $F_0 : V \to V$ (see the commands `CoxeterCoset` and `CoxeterSubCoset`). As it is true for the Coxeter groups the elements of WF_0 are uniquely determined by the permutation they induce on the set of roots R. We consider these permutations as **Elements** of the Coxeter coset.

Coxeter cosets are implemented in GAP by a record which points to a Coxeter datum record and has additional fields holding **F0Mat**, **F0Perm** and the corresponding reduced element **phi**. Functions on the coset work with elements of the group coset WF_0 (for example, **ChevieClassInfo**); however, most definitions for elements of untwisted Coxeter groups apply without change to elements in WF_0: e.g., if we define the length of an element $w\phi \in WF_0$ as the number of positive roots it sends to negative ones, it is the same as the length of w, i.e., ϕ is of length 0, since ϕ has been chosen to preserve the set of positive roots. Similarly, the **CoxeterWord** describing $w\phi$ is the same as the one for w, etc...

We associate to a Coxeter coset WF_0 a **twisted Dynkin diagram**, consisting of the Dynkin diagram of W and the graph automorphism induced by ϕ on this diagram (this specifies the group $W \rtimes \langle F \rangle$, mentioned above, up to isomorphism). See the commands **CartanType**, **CartanName** and **PrintDynkinDiagram** for Coxeter cosets.

The motivation for introducing this notion comes from the theory of Chevalley groups, or more generally reductive algebraic groups over finite fields. Let $\mathbf{G}$ be a reductive algebraic group over the algebraic closure $\overline{\mathbb{F}}_q$ of a finite field $\mathbb{F}_q$, which is defined over $\mathbb{F}_q$, with corresponding Frobenius endomorphism F, so the finite group of rational points $\mathbf{G}(\mathbb{F}_q)$ identifies to the subgroup $\mathbf{G}^F$ of fixed points under F. Let $\mathbf{T}$ be an F-stable maximal torus of $\mathbf{G}$. The Weyl group of $\mathbf{G}$ with respect to $\mathbf{T}$ is the quotient $W = N_{\mathbf{G}}(\mathbf{T})/\mathbf{T}$. Let $X(\mathbf{T})$ be the character group of $\mathbf{T}$, that is the group of rational homomorphisms $\mathbf{T} \to \overline{\mathbb{F}}_q$. The group W acts naturally on $X(\mathbf{T})$, and thus also on the vector space $V = \mathbb{R} \otimes X(\mathbf{T})$. It is a fundamental fact in the theory of reductive groups that there is a canonical root system Φ in V defined by $(\mathbf{G}, \mathbf{T})$ such that W is the Weyl group of that root system.

The Frobenius endomorphism F acts also naturally on $X(T)$ and defines thus an endomorphism of V, which is of the form qF_0, where F_0 is an automorphism of finite order of V. We get thus a Coxeter datum (V, W, F_0).

To completely specify $\mathbf{G}^F$ up to isomorphism, we need a little more information. Let $Y(\mathbf{T})$ be the group of cocharacters of $\mathbf{T}$, that is of rational homomorphisms $\overline{\mathbb{F}}_q \to \mathbf{T}$. The $\mathbb{Z}$-modules $X(\mathbf{T})$ and $Y(\mathbf{T})$ are naturally dual to each other, and there is a canonical root system $\Phi^\vee$ in $Y(\mathbf{T})$ dual to Φ. Let $V^\vee = \mathbb{R} \otimes Y(\mathbf{T})$. The classification theorems on reductive groups show that the isomorphism type of $\mathbf{G}^F$ is completely determined by the datum $(V, \Phi, V^\vee, \Phi^\vee, F_0)$, and the integer q. Thus we can think of this datum as a way of representing in GAP the essential information which determines a Chevalley group. Indeed, all properties of Chevalley groups can be computed from its Weyl datum: symbols representing unipotent characters, conjugacy classes, and finally the whole character table of $\mathbf{G}^F$.

It turns out that an interesting part of the objects attached to this datum depends only on (V, W, F_0): the order of the maximal tori, the "fake degrees", the order of $\mathbf{G}^F$, Deligne-Lusztig induction in terms of "almost characters", symbols representing unipotent characters and Deligne-Lusztig induction, etc... (see, e.g., [BMM93a]). It is thus possible to extend their construction to non-crystallographic groups (or even to more general complex reflection groups); this is why we did not include Φ in the definition of a Coxeter coset.

However, in GAP we will always have the whole datum $(V, \Phi, V^\vee, \Phi^\vee, F_0)$. We assume that we have chosen a Borel subgroup of $\mathbf{G}$ containing $\mathbf{T}$. This defines an order on the roots, and thus a basis Π of Φ as well as a basis $\Pi^\vee$ of $\Phi^\vee$. Any element of WF_0 induces a permutation of the roots, and there is a unique element $\phi \in WF_0$, which we call the **reduced element** in the coset, which preserves the set of positive roots. This element is stored in the component `phi` of the coset, and can be used to test isomorphism of cosets. The coset WF_0 is completely defined by the permutation `F0Perm` of the roots induced by F_0 when $\mathbf{G}$ is semi-simple (in which case Φ generates V). So in this case we just need to specify `F0Perm` when defining the coset.

We should mention also a special case of Chevalley groups which does not exactly fit the above description: the Ree and Suzuki groups. In these cases, the group is defined as $\mathbf{G}^F$ where F is not a Frobenius endomorphism, but an isogeny such that either F^2 or F^3 is a Frobenius endomorphism. Here, F still defines an endomorphism of V which normalizes W (and induces the automorphism of order 2 of W — W is of type G_2, B_2 or F_4), but this endomorphism is no longer q times an endomorphism of finite order; however, up to some power of q, F_0 still takes a form independent of q (but here, q is a power of a fixed prime p equal to 2 or 3 depending on the group considered). This has not yet been implemented.

11.1 CoxeterCoset

`CoxeterCoset(` *W* `[,` *F0Mat* `] )`

`CoxeterCoset(` *W* `[,` *F0Perm* `] )`

This function returns a Coxeter coset as a GAP object. The argument *W* must be a Coxeter group (created by `CoxeterGroup` or `ReflectionSubgroup`). In the first form the argument *F0Mat* must be an invertible matrix with `Rank(`*W*`)` rows, representing an automorphism F_0 of the root system of the parent of W. In the second form *F0Perm* is a permutation which describes the images of the simple roots under F_0 (and only these images are used). Of course this form is only allowed if the semisimple rank of W equals the rank (i.e., the simple roots are a basis of V). If there is no second argument the default for *F0Mat* is the identity matrix.

`CoxeterCoset` returns a record from which we document the following components:

`isDomain`, `isFinite`
: true

`coxeter`
: the Coxeter group W

`F0Mat`
: the matrix acting on V which represents F_0

`F0Perm`
: the permutation on the roots of W induced by `F0Mat`

`phi`
: the shortest element in the coset (as permutation)

`w1`
: `phi/F0Perm`

In the first example we create a Coxeter coset corresponding to the general unitary groups $GU_3(q)$ over finite fields with q elements.

```
gap> W := CoxeterGroup( [ [ 1, -1, 0 ], [ 0, 1, -1 ] ],
>                       [ [ 1, -1, 0 ], [ 0, 1, -1 ] ] );;
gap> gu3 := CoxeterCoset( W, -IdentityMat( 3 ) );
CoxeterCoset(CoxeterGroup([ [ 1, -1, 0 ], [ 0, 1, -1 ] ],
[ [ 1, -1, 0 ], [ 0, 1, -1 ] ]),
[ [ -1, 0, 0 ], [ 0, -1, 0 ], [ 0, 0, -1 ] ])
gap> F4 := CoxeterGroup( "F", 4 );;
gap> D4 := ReflectionSubgroup( F4, [ 1, 2, 16, 48 ] );
ReflectionSubgroup(CoxeterGroup("F", 4), [ 1, 2, 9, 16 ])
gap> PrintDynkinDiagram( D4 );
D4   9
      \
       1 - 16
      /
     2
gap> 3D4 := CoxeterCoset( D4, (2,9,16) );
CoxeterCoset(ReflectionSubgroup(CoxeterGroup("F", 4),
[ 1, 2, 9, 16 ]), ( 2, 9,16))
```

These functions require the package "chevie"(see ??).

11.2 CoxeterSubCoset

`CoxeterSubCoset(` WF, r, `[`w`] )`

Returns the reflection subcoset of the Coxeter coset WF generated by the reflections with roots specified by r. r is a list of indices specifying a subset of the roots of W where W is the Coxeter group `CoxeterGroup(` WF`)`. If specified, w must be an element of W such that `w*WF.FOPerm` normalizes the subroot system generated by r. If absent, the default value for w is `()`. It is an error, if `w*WF.FOPerm` does not normalize the subsystem.

```
gap> CoxeterSubCoset( CoxeterCoset( CoxeterGroup( "A", 2 ), (1,2) ),
>                                                           [ 1 ] );
Error, must give w, such that w * WF.FOPerm normalizes subroot system.
 in
CoxeterSubCoset(CoxeterCoset(CoxeterGroup("A", 2), (1,2)), [ 1 ])
 called from main loop
brk>
gap> f4coset := CoxeterCoset( CoxeterGroup( "F", 4 ) );
CoxeterCoset(CoxeterGroup("F", 4))
gap> w := RepresentativeOperation( CoxeterGroup( f4coset ),
>                    [ 1, 2, 9, 16 ], [ 1, 9, 16, 2], OnTuples );;
gap> 3d4again := CoxeterSubCoset( f4coset, [ 1, 2, 9, 16], w );
CoxeterSubCoset(CoxeterCoset(CoxeterGroup("F", 4)), [ 1, 2, 9, 16 ],
( 2, 9,16)( 3, 4,31)( 5,11,18)( 6,13,10)( 7,27,28)( 8,15,12)(14,22,20)
(17,19,21)(26,33,40)(29,35,42)(30,37,34)(32,39,36)(38,46,44)
(41,43,45))
gap> PrintDynkinDiagram( 3d4again );
phi acts as ( 2, 9,16) on the component below
D4   9
```

```
      \
       1 - 2
      /
     16
```

This function requires the package "chevie"(see ??).

11.3 Functions on Coxeter cosets

`CoxeterGroup(` *WF* `)`
returns the Coxeter group of which *WF* is a coset.

Quite a few functions defined for domains, permutation groups or Coxeter groups have been implemented to work with Coxeter cosets.

`Elements`, `Random`, `Representative`, `Size`, `in`
these functions use the corresponding functions for `CoxeterGroup(` *WF* `)`.

`ConjugacyClasses(` *WF* `)`
returns the conjugacy classes of the Coxeter coset *WF* (see also the introduction of this Chapter). Let W be `CoxeterGroup(`*WF*`)`. Then the classes are defined to be the W-orbits on WF_0, where W acts by conjugation (they coincide with the WF_0-orbits, WF_0 acting by the conjugation); by the translation $w \mapsto w\phi^{-1}$ they are sent to the ϕ-conjugacy classes of W.

`PositionClass(` *WF* `,` x `)`
for any element x in *WF* this returns the number `i` such that x is an element of `ConjugacyClasses(`*WF*`)[i]` (to work fast, the classification of Coxeter groups is used).

`FusionConjugacyClasses(` *WF1*`,` *WF* `)`
works in the same way as for groups. See the section `CoxeterSubCoset`.

`Print(` *WF* `)`
if *WF*`.name` is bound then this is printed, else this function prints the coset in a form which can be input back into GAP.

`InductionTable(` *HF*`,` *WF* `)`

works in the same way as for groups. It gives the induction table from the Coxeter subcoset *HF* to the Coxeter coset *WF*. If HwF_0 is a Coxeter subcoset of WF_0, restriction of characters is defined as restriction of functions from WF_0 to HwF_0, and induction as the adjoint map for the natural scalar product $\langle f,g\rangle = \frac{1}{\#W}\sum_{v\in W} f(vF_0)\overline{g}(vF_0)$.
If the Coxeter coset *WF* represents the reductive group $\mathbf{G}$, and *HF* corresponds to a Levi subgroup $\mathbf{L}$, then the induction from *HF* to *WF* describes the Lusztig induction of uniform unipotent almost characters from $\mathbf{L}$ to $\mathbf{G}$.

Harish-Chandra induction in the basis of almost characters:

```
gap> WF := CoxeterCoset( CoxeterGroup( "A", 4 ), (1,4)(2,3) );
CoxeterCoset(CoxeterGroup("A", 4), (1,4)(2,3))
gap> Display( InductionTable( CoxeterSubCoset( WF, [ 2, 3 ] ), WF ) );
      | 111 21 3
________________
```

```
11111 |    1  . .
2111  |    .  1 .
221   |    1  . .
311   |    1  . 1
32    |    .  . 1
41    |    .  1 .
5     |    .  . 1
```

Lusztig induction from a diagonal Levi:

```
gap> HF := CoxeterSubCoset( WF, [1, 2],
>               LongestCoxeterElement( CoxeterGroup( WF ) ) );;
gap> Display( InductionTable( HF, WF ) );

      | 111 21  3
-----------------
11111 |  -1  .  .
2111  |  -2 -1  .
221   |  -1 -2  .
311   |   1  2 -1
32    |   . -2  1
41    |   .  1 -2
5     |   .  .  1
```

A descent of scalars:

```
gap> W := CoxeterCoset( CoxeterGroup( "A", 2, "A", 2 ), (1,3)(2,4) );
CoxeterCoset(CoxeterGroup("A", 2, "A", 2), (1,3)(2,4))
gap> Display( InductionTable( CoxeterSubCoset( W, [ 1, 3 ] ), W ) );
    | 11 2
----------
111 |  1 .
21  |  1 1
3   |  . 1
```

`CartanName( WF )`

returns a string which describes the isomorphism type of the group $W \rtimes \langle F\rangle$, associated to WF, as described in the introduction of this Chapter. An orbit of $\phi = WF$.`phi` on the components is put in brackets if of length k greater than 1, and is preceded by the order of phi^k on it, if this is not 1. For example `"2(A2xA2)"` denotes 2 components of type A_2 permuted by F_0, and such that phi^2 induces the non-trivial diagram automorphism on any of them, while `3D4` denotes an orbit of length 1 on which phi is of order 3.

```
gap> W := CoxeterCoset( CoxeterGroup( "A", 2, "G", 2, "A", 2 ),
>                                                     (1,5,2,6) );
CoxeterCoset(CoxeterGroup("A", 2, "G", 2, "A", 2), (1,5,2,6))
gap> CartanName( W );
"2(A2xA2)xG2"
```

`PrintDynkinDiagram( WF )`

this is a purely descriptive routine (as was already the case for finite Coxeter groups themselves). It prints the Dynkin diagram of `CoxeterGroup(` *WF* `)` together with the information how *WF*`.phi` acts on it.

```
gap> W := CoxeterCoset( CoxeterGroup( "A", 2, "A", 2 ), (1,3,2,4) );
CoxeterCoset(CoxeterGroup("A", 2, "A", 2), (1,3,2,4))
gap> PrintDynkinDiagram( W );
phi permutes the next 2 components
phi^2 acts as (1,2) on the component below
A2    1 - 2
A2    3 - 4
```

`ChevieClassInfo(` *WF* `)`, see the explicit description in 11.5.

`ChevieCharInfo`
This function returns additional information on the irreducible characters, see 4.10 for more details.

`CharParams(` *WF* `)`

`CharName(` *WF* `)`

Note that some functions for elements of a Coxeter group work naturally for elements of a Coxeter coset: `CoxeterWord`, `PermCoxeterWord`, `CoxeterLength`, `ReducedInCoxeterCoset`, `LeftDescentSet`, `RightDescentSet`, etc...

11.4 CartanType for Coxeter cosets

`CartanType(` *WF* `)`

returns the type of the Coxeter coset *WF*. This consists of a list of records, one for each orbit of *WF*`.phi` on the irreducible components of the Dynkin diagram of `CoxeterGroup(`*WF*`)`, which have two fields:

`orbit`
is a list of types of the irreducible components in the orbit. These types are the same as returned by the function `CartanType` for an irreducible untwisted Coxeter group (see `CartanType` in chapter 2): a couple `[`*type*`,`*indices*`]` (a triple for type $I_2(n)$). The components are ordered according to the action of *WF*`.phi`, so *WF*`.phi` maps the generating permutations with indices in the first type to indices in the second type in the same order as stored in the type, etc ...
`phi`
if k is the number of irreducible components in the orbit, this is the permutation which describes the action of *WF*`.phi`^{k} on the simple roots of the first irreducible component in the orbit.

```
gap> W := CoxeterCoset( CoxeterGroup( "A", 2, "A", 2 ), (1,3,2,4) );
CoxeterCoset(CoxeterGroup("A", 2, "A", 2), (1,3,2,4))
gap> CartanType( W );
[ rec(
      orbit := [ [ "A", [ 1, 2 ] ], [ "A", [ 3, 4 ] ] ],
      phi := (1,2) ) ]
```

This function requires the package "chevie"(see ??).

11.5 ChevieClassInfo for Coxeter cosets

`ChevieClassInfo(` *WF* `)`

returns information about the conjugacy classes of the Coxeter coset WF. The result is a record with three components: **classtext** contains a list of reduced words for the representatives in `ConjugacyClasses(`WF`)`, **classnames** contains corresponding names for the classes, and **classparams** gives corresponding parameters for the classes. Let W be the Coxeter group `CoxeterGroup(`WF`)`. In the case where $-1 \notin W$, i.e., $\phi = -w_0$, they are obtained by multiplying by w_0 a set of representatives of *maximal* length of the classes of W.

```
gap> W := CoxeterGroup( "D", 4 );;
gap> ChevieClassInfo( CoxeterCoset( W, (1,2,4) ) );
rec(
  classtext := [ [ 1 ], [  ], [ 1, 2, 3, 1, 2, 3 ], [ 3 ], [ 1, 3 ],
      [ 1, 2, 3, 1, 2, 4, 3, 2 ], [ 1, 2, 3, 2 ] ],
  classparams :=
   [ [ "C_3" ], [ "~A_2" ], [ "C_3+A_1" ], [ "~A_2+A_1" ], [ "F_4" ],
      [ "~A_2+A_2" ], [ "F_4(a_1)" ] ],
  classnames :=
   [ "C_3", "~A_2", "C_3+A_1", "~A_2+A_1", "F_4", "~A_2+A_2",
      "F_4(a_1)" ] )
```

This function requires the package "chevie"(see ??).

11.6 CharTable for Coxeter cosets

`CharTable(` *WF* `)`

This function returns the character table of the Coxeter coset WF (see also the introduction of this Chapter). We call "characters"of the Coxeter coset WF with corresponding Coxeter group W the restriction to WF_0 of a set containing one extension of each F_0-invariant character of W to the semidirect product of W with the cyclic group generated by F_0. (We choose, following Lusztig, in each case one non-canonical extension, called the preferred extension.)

The returned record contains almost all components present in the character table of a Coxeter group. But if F_0 is not trivial then there are no components **powermap** (since powers of elements in the coset need not be in the coset) and **orders** (if you really need them, use **MatXPerm** to determine the order of elements in the coset).

```
gap> W := CoxeterCoset( CoxeterGroup( "D", 4 ), (1,2,4) );
CoxeterCoset(CoxeterGroup("D", 4), (1,2,4))
gap> Display( CharTable( W ) );
W( 3D4 )

       2   2    2        2         2   2        3        3
       3   1    1        1         .   .        1        1

         C_3 ~A_2 C_3+A_1 ~A_2+A_1 F_4 ~A_2+A_2 F_4(a_1)
```

```
    .4          1    1       1        1   1        1        1
    .1111      -1    1       1       -1   1        1        1
    .22         .    2       2        .  -1       -1       -1
    11.2        .    .       .        .  -1        3        3
    1.3         1    1      -1       -1   .       -2        2
    1.111      -1    1      -1        1   .       -2        2
    1.21        .    2      -2        .   .        2       -2
```

This function requires the package "chevie" (see ??).

11.7 Frobenius

`Frobenius(` *WF* `)(` *word* `)`

`Frobenius(` *WF* `)(` *permutation* `)`

`Frobenius(` *WF* `)(` *rec* `)`

Given a Coxeter coset *WF*, `Frobenius(`*WF*`)` returns a function which makes *WF*`.F0Perm` act on its argument which may be either a CoxeterWord, a permutation, a braid or a Hecke element. For instance, if H is an Hecke algebra for `CoxeterGroup(WF)`, to compute $(T_w F_0)^2$ in that algebra one can do `Basis(H,"T")(w)*Frobenius(WF)((Basis(H,"T")(w)))`.

This function requires the package "chevie" (see ??).

11.8 PhiFactors

`PhiFactors(` *WF* `)`

Let W be the Coxeter group corresponding to the Coxeter coset *WF*, and let V be the vector space of dimension `W.rank` on which W acts as a reflection group. Let $f_1, \ldots, f_n$ be the basic invariants of W on the symmetric algebra SV of V. The matrix `WF.F0Mat` has the f_i as eigenvectors. The corresponding eigenvalues, sorted in order of increasing degrees of the f_i are called the **factors** of F_0 acting on V.

```
gap> W := CoxeterGroup( "E", 6 );; WF := CoxeterCoset( W );
CoxeterCoset(CoxeterGroup("E", 6))
gap> phi := PermCoxeterWord( W,
>       [ 6, 5, 4, 2, 3, 1, 4, 3, 5, 4, 2, 6, 5, 4, 3, 1 ] );;
gap> HF := CoxeterSubCoset( WF, [ 2..5 ], phi );;
gap> PrintDynkinDiagram( HF );
phi acts as (2,3,5) on the component below
D4   2
      \
       4 - 5
      /
     3
gap> PhiFactors( HF );
[ E(3), E(3)^2, 1, E(3), E(3)^2, 1 ]
gap> ReflectionDegrees( CoxeterGroup( HF ) );
[ 1, 1, 2, 4, 4, 6 ]
```

This function requires the package "chevie" (see ??).

Chapter 12

Hecke cosets

"Hecke cosets" are $H\phi$ where H is a Hecke algebra of some Coxeter group W on which the reduced element ϕ acts by $\phi(T_w) = T_{\phi(w)}$. This corresponds to the action of the Frobenius automorphism on the commuting algebra of the induced of the trivial representation from the rational points of some F-stable Borel subgroup to $\mathbf{G}^F$.

```
gap> W := CoxeterGroup( "A", 2 );;
gap> q := X( Rationals );; q.name := "q";;
gap> HF := Hecke( CoxeterCoset( W, (1,2) ), q^2, q );
Hecke(CoxeterCoset(CoxeterGroup("A", 2), (1,2)),[ q^2, q^2 ],[ q, q ])
gap> Display( CharTable( HF ) );
H(2A2)

         2      1   1   .
         3      1   .   1

              111  21   3
        2P    111 111   3
        3P    111  21 111

111          -1   1  -1
21         -2q^3   0   q
3           q^6   1 q^2
```

We do not yet have a satisfying theory of character tables for these cosets (the equivalent of `HeckeClassPolynomials` has not yet been proven to exist). We hope that future releases of CHEVIE will contain better versions of such character tables.

12.1 Hecke for Coxeter cosets

`Hecke(` *WF* `,` *H* `)`

`Hecke(` *WF* `,` *params* `)`

Construct a Hecke coset a Coxeter coset WF and an Hecke algebra associated to the Coxeter-Group of WF. The second form is equivalent to `Hecke(` *WF* `, Hecke(CoxeterGroup(`*WF*`),` *params*`))`.

This function requires the package "chevie"(see ??).

12.2 Operations and functions for Hecke cosets

`Hecke`
: returns the untwisted Hecke algebra corresponding to the Hecke coset.

`CoxeterCoset`
: returns the Coxeter coset corresponding to the Hecke coset.

`CoxeterGroup`
: returns the untwisted Coxeter group corresponding to the Hecke coset.

`Print`
: prints the Hecke coset in a form which can be read back into GAP.

`CharTable`
: returns the character table of the Hecke coset.

These functions require the package "chevie"(see ??).

Chapter 13

Appendix – utility functions for the CHEVIE package

The functions described below, used in various parts of the CHEVIE package, are of a general nature and should really be included in other parts of the GAP library. We include them here for the moment for the commodity of the reader.

13.1 InductionTable

`InductionTable(` *W1* `,` *W* `)`

`InductionTable` computes the decomposition of the induced characters from the subgroup *W1* into irreducible characters of *W*. The rows correspond to the characters of the parent group, the columns to those of the subgroup. What is returned is actually a record with several fields:`scalar` contains the induction table proper, and there is a `Display` method. The other fields contain labeling information taken from the character tables of *W1* and *W* when it exists.

```
gap> W := Group( [ (1,2), (2,3), (3,4) ], () );
Group( (1,2), (2,3), (3,4) )
gap> H:=Subgroup( W, [ (1,2), (3,4) ] );
Subgroup( Group( (1,2), (2,3), (3,4) ), [ (1,2), (3,4) ] )
gap> W.name := "W";; H.name := "H";; # to avoid warnings
gap> Display( InductionTable( H, W ) );
    | X.1 X.2 X.3 X.4
_____________________________
X.1 |   1   .   .   .
X.2 |   .   .   .   1
X.3 |   1   .   .   1
X.4 |   .   1   1   1
X.5 |   1   1   1   .
gap> W := CoxeterGroup( "G", 2 );;
gap> H := ReflectionSubgroup( W, [ 1, 4 ] );
ReflectionSubgroup(CoxeterGroup("G", 2), [ 1, 4 ])
```

```
gap> CartanName( H );
"A1x~A1"
gap> t := InductionTable( H, W );
InductionTable( ReflectionSubgroup(CoxeterGroup("G", 2),
[ 1, 4 ]), CoxeterGroup("G", 2))
gap> Display( t );
            | 11,11 11,2 2,11 2,2
________________________________________
phi_{1,0}   |     .    .    .   1
phi_{1,6}   |     1    .    .   .
phi_{1,3}'  |     .    1    .   .
phi_{1,3}'' |     .    .    1   .
phi_{2,1}   |     .    1    1   .
phi_{2,2}   |     1    .    .   1
```

If one does not want to see the whole induction table, one can specify the characters of the subgroup and of the parent group by giving a second argument to `Display`. This second argument is a record with optional components `charsGroup` and `charsSubgroup`, to which one has to assign the lists of rows and columns which should be printed.

```
gap> Display( t,rec( charsGroup := [5], charsSubgroup := [2,3] ) );
Induction from A1x~A1 into G2
          | 11,2 2,11
______________________________
phi_{2,1} |    1    1
```

13.2 CharRepresentationWords

`CharRepresentationWords(` *rep* `,` *elts* `)`

given a list *rep* of matrices corresponding to generators and a list *elts* of words in the generators it returns the list of traces of the corresponding representation on the elements in *elts*.

```
gap> H := Hecke(CoxeterGroup( "F", 4 ));;
gap> r := ChevieClassInfo( Group( H ) ).classtext;;
gap> t := HeckeReflectionRepresentation( H );;
gap> CharRepresentationWords( t, r );
[ 4, -4, 0, 1, -1, 0, 1, -1, -2, 2, 0, 2, -2, -1, 1, 0, 2, -2, -1, 1,
  0, 0, 2, -2, 0 ]
```

13.3 PositionClass

`PositionClass(` G, c `)`

G must be a domain for which `ConjugacyClasses` is defined and c must be an element of G. This functions returns a positive integer `i` such that `c in ConjugacyClasses( G )[i]`.

```
gap> G := Group( (1,2)(3,4), (1,2,3,4,5) );;
gap> ConjugacyClasses( G );
[ ConjugacyClass( Group( (1,2)(3,4), (1,2,3,4,5) ), () ),
  ConjugacyClass( Group( (1,2)(3,4), (1,2,3,4,5) ), (3,4,5) ),
```

```
    ConjugacyClass( Group( (1,2)(3,4), (1,2,3,4,5) ), (2,3)(4,5) ),
    ConjugacyClass( Group( (1,2)(3,4), (1,2,3,4,5) ), (1,2,3,4,5) ),
    ConjugacyClass( Group( (1,2)(3,4), (1,2,3,4,5) ), (1,2,3,5,4) ) ]
gap>  g := Random( G );
(1,2,5,4,3)
gap> PositionClass( G, g );
5
```

13.4 PointsAndRepresentativesOrbits

`PointsAndRepresentativesOrbits(` *G*`[,` *m*`] )`

returns a pair *[orb, rep]* where *orb* is a list of the orbits of the permutation group G on `[ 1..LargestMovedPoint(` G `) ]` and *rep* is a list of list of elements of G such that `rep[i][j]` applied to `orb[i][1]` yields `orb[i][j]` for all i, j. If the optional argument m is given, then `LargestMovedPoint(` G `)` is replaced by the integer m.

```
gap> G := Group( (1,7)(2,3)(5,6)(8,9)(11,12),
>                (1,5)(2,8)(3,4)(7,11)(9,10) );;
gap> PointsAndRepresentativesOrbits( G );
[ [ [ 1, 7, 5, 11, 6, 12 ], [ 2, 3, 8, 4, 9, 10 ] ],
  [ [ (), ( 1, 7)( 2, 3)( 5, 6)( 8, 9)(11,12),
          ( 1, 5)( 2, 8)( 3, 4)( 7,11)( 9,10),
          ( 1,11,12, 7, 5, 6)( 2, 4, 3, 8,10, 9),
          ( 1, 6, 5, 7,12,11)( 2, 9,10, 8, 3, 4),
          ( 1,12)( 2, 4)( 3, 9)( 6, 7)( 8,10) ],
      [ (), ( 1, 7)( 2, 3)( 5, 6)( 8, 9)(11,12),
          ( 1, 5)( 2, 8)( 3, 4)( 7,11)( 9,10),
          ( 1,11,12, 7, 5, 6)( 2, 4, 3, 8,10, 9),
          ( 1, 6, 5, 7,12,11)( 2, 9,10, 8, 3, 4),
          ( 1, 6)( 2,10)( 4, 8)( 5,11)( 7,12) ] ] ]
```

13.5 DirectSumMat

`DirectSumMat(` *mat1*`, ... ,` *matn* `)`

returns the block diagonal direct sum of the matrices *mat1*, ..., *matn*.

```
gap> C1 := [ [   2,  -1,   0,   0 ],
>            [  -1,   2,  -1,   0 ],
>            [   0,  -1,   2,  -1 ],
>            [   0,   0,  -1,   2 ] ];;
gap> C2 := [ [   2,   0,  -1,   0 ],
>            [   0,   2,  -1,   0 ],
>            [  -1,  -1,   2,  -1 ],
>            [   0,   0,  -1,   2 ] ];;
gap> d := DirectSumMat( C1, C2 );;
gap> PrintArray( d );
[ [   2,  -1,   0,   0,   0,   0,   0,   0 ],
  [  -1,   2,  -1,   0,   0,   0,   0,   0 ],
```

```
      [  0, -1,  2, -1,  0,  0,  0,  0 ],
      [  0,  0, -1,  2,  0,  0,  0,  0 ],
      [  0,  0,  0,  0,  2,  0, -1,  0 ],
      [  0,  0,  0,  0,  0,  2, -1,  0 ],
      [  0,  0,  0,  0, -1, -1,  2, -1 ],
      [  0,  0,  0,  0,  0,  0, -1,  2 ] ]
```

One can also use a computed list of matrices as an argument; the function call then reads `ApplyFunc(DirectSumMat, [`*mat1*`, ... ,` *matn*`] )`.

13.6 DecomposedMat

`DecomposedMat(` *mat* `)`

finds if the square matrix *mat* admits a block decomposition.

Define a graph G with vertices `[1..Length(mat)]` and with an edge between `i` and `j` if either `mat[i][j]` or `mat[j][i]` is non-zero. `DecomposedMat` return a list of lists `l` such that `l[1]`,`l[2]`, etc.. are the vertices in each connected component of `G`. In other words, the matrices `mat{l[1]}{l[1]}`,`mat{l[2]}{l[2]}`, etc... are blocks of the matrix *mat*.

```
gap> m := [ [  0,  0,  0,  1 ],
>           [  0,  0,  1,  0 ],
>           [  0,  1,  0,  0 ],
>           [  1,  0,  0,  0 ] ];;
gap> DecomposedMat( m );
[ [ 1, 4 ], [ 2, 3 ] ]
gap> PrintArray( m{[ 1, 4 ]}{[ 1, 4 ]});
[ [  0,  1 ],
  [  1,  0 ] ]
```

13.7 IsDiagonalMat

`IsDiagonalMat(` *mat* `)`

mat must be a matrix. This function returns `true` if all entries `mat[i][j]` with `i<>j` are equal to `0*mat[i][j]` and `false` otherwise.

```
gap> a := [ [ 1, 2 ], [ 3, 1 ] ];;
gap> IsDiagonalMat( a );
false
```

13.8 IsLowerTriangularMat

`IsLowerTriangularMat(` *mat* `)`

mat must be a matrix. This function returns `true` if all entries `mat[i][j]` with `j>i` are equal to `0*mat[i][j]` and `false` otherwise.

```
gap> a := [ [ 1, 2 ], [ 3, 1 ] ];;
gap> IsLowerTriangularMat( a );
false
gap> a[1][2] := 0;;
gap> IsLowerTriangularMat( a );
true
```

13.9 IsNormalizing

`IsNormalizing(` *lst*`,` *mat* `)`

returns true or false according to whether the matrix *mat* leaves the vectors in *lst* as a set invariant, i.e., `Set(l * M) = Set( l )`.

```
gap> a := [ [ 1, 2 ], [ 3, 1 ] ];;
gap> l := [ [ 1, 0 ], [ 0, 1 ], [ 1, 1 ], [ 0, 0 ] ];;
gap> l * a;
[ [ 1, 2 ], [ 3, 1 ], [ 4, 3 ], [ 0, 0 ] ]
gap> IsNormalizing( l, a );
false
```

13.10 SublistUnbnd

`SublistUnbnd(` *l*`,` *ind* `)`

Sublist of a list with possibly unbound entries.

The writing of this function was prompted by the fact that if *l* has some unbound entries, `l{ind}` returns an error message instead of doing what is expected (which is what this routine does).

```
gap> l := [ 1, , 2, , , 3 ];;
gap> SublistUnbnd( l, [ 1..4 ] );
[ 1,, 2 ]
```

If you use `l{[ 1..4 ]}`, you get an error message.

13.11 PermListList

`PermListList(` *a*`,` *b* `)`

return a permutation `p` such that `Permuted(b,p)=a`, where *a* and *b* are two lists. It returns false if no such permutation exists.

```
gap> a := [ 1, 2, 3, 4 ];;
gap> b := [ 2, 3, 1, 4 ];;
gap> p := PermListList( a, b );
(1,2,3)
gap> Permuted( b, p );
[ 1, 2, 3, 4 ]
gap> PermListList( a, [ 3, 2, 1, 5 ] );
false
```

13.12 Coefficient

`Coefficient(` *a*`,` *b* `)`

`generic` routine which looks if *a* has a `Coefficient` method in its operations record and then returns `a.operations.Coefficient(a,b)`.

13.13 IntListToString

`IntListToString(` *part*`, [`*brackets*`] )`

part must be a list of positive integers. If all of them are smaller than 10 then a string of digits corresponding to the entries of *part* is returned. If an entry is ≥ 10 then the elements of *part* are converted to strings, concatenated with separating commas and the result surrounded by brackets. By default `()` brackets are used. This may be changed by giving as second argument a length two string specifying another kind of brackets.

```
gap> IntListToString( [ 4, 2, 2, 1, 1 ] );
"42211"
gap> IntListToString( [ 14, 2, 2, 1, 1 ] );
"(14,2,2,1,1)"
gap> IntListToString( [ 14, 2, 2, 1, 1 ], "{}" );
"{14,2,2,1,1}"
```

13.14 DoublePartitionToString

`DoublePartitionToString(` *pair* `)`

converts the double partition *pair* to a string where the two partitions are separated by a point.

```
gap> d := DoublePartitions( 3 );
[ [ [ 1 ], [ 1, 1 ] ], [ [ 1 ], [ 2 ] ], [ [ 1, 1 ], [ 1 ] ],
  [ [ 1, 1, 1 ], [  ] ], [ [ 2 ], [ 1 ] ], [ [ 2, 1 ], [  ] ],
  [ [ 3 ], [  ] ], [ [  ], [ 1, 1, 1 ] ], [ [  ], [ 2, 1 ] ],
  [ [  ], [ 3 ] ] ]
gap> for i in d do
>      Print( DoublePartitionToString( i )," ");
> od; Print("\n");
1.11  1.2  11.1  111.  2.1  21.  3.  .111  .21  .3
```

Bibliography

[AL82] D. Alvis and G. Lusztig. The representations and generic degrees of the Hecke algebra of type H_4. *J. reine angew. Math.*, 336:201–212, 1982. Correction: ibid. **449**, 217–218 (1994).

[Alv87] D. Alvis. The left cells of the Coxeter group of type H_4. *J. Algebra*, 107:160–168, 1987.

[BC72] C. T. Benson and C. W. Curtis. On the degrees and rationality of certain characters of finite Chevalley groups. *Trans. Amer. Math. Soc.*, 165:251–273, 1972.

[Ben76] M. Benard. Schur indices and splitting fields of the unitary reflection groups. *J. Algebra*, 38:318–342, 1976.

[BM93] M. Broué and G. Malle. Zyklotomische Heckealgebren. In *Représentations unipotentes génériques et blocs des groupes réductifs fini* [BMM93b], pages 119–189.

[BMM93a] M. Broué, G. Malle, and J. Michel. Generic blocks of finite reductive groups. In *Représentations unipotentes génériques et blocs des groupes réductifs fini* [BMM93b], pages 7–92.

[BMM93b] M. Broué, G. Malle, and J. Michel. *Représentations unipotentes génériques et blocs des groupes réductifs finis*, volume 212 of *Astérisque*. Société Mathématique de France, 1993.

[Bou68] N. Bourbaki. *Groupes et algèbres de Lie, Ch. 4–6.* Hermann, Paris, 1968. Masson, Paris: 1981.

[Bri71] E. Brieskorn. Die Fundamentalgruppe des Raumes der regulären Orbits einer endlichen komplexen Spiegelungsgruppe. *Invent. Math.*, 12:57–61, 1971.

[BS72] E. Brieskorn and K. Saito. Artin-Gruppen und Coxeter-Gruppen. *Invent. Math.*, 17:245–271, 1972.

[Cab96] Marc Cabanes, editor. *Finite Reductive Groups, Related Structures and Representations*, volume 141 of *Progress in Mathematics*. Birkhäuser, Basel, 1996.

[Car72] R. W. Carter. Conjugacy classes in the Weyl group. *Compositio Math.*, 25:1–59, 1972.

[Car85] R. W. Carter. *Finite groups of Lie type: Conjugacy classes and complex characters.* Wiley, New York, 1985.

[Car86] R. W. Carter. Representation theory of the 0–Hecke algebra. *J. Algebra*, 104:89–103, 1986.

[CCN+85] J. H. Conway, R. T. Curtis, S. P. Norton, R. A. Parker, and R. A. Wilson. *Atlas of Finite Groups.* Oxford University Press, 1985.

[Cha92] R. Charney. Artin groups of finite type are bi-automatic. *Math. Ann.*, 292:671–683, 1992.

[CR87] C. W. Curtis and I. Reiner. *Methods of representation theory, vol. I, II.* John Wiley, New York, 1981/1987.

[Del72] P. Deligne. Les immeubles des groupes de tresses généralisés. *Invent. Math.*, 17:273–302, 1972.

[Deo89] V.V. Deodhar. A note on subgroups generated by reflections in Coxeter groups. *Arch. Math.*, 53:543–546, 1989.

[DuC91] F. DuCloux. *Coxeter Version 1.0.* Université de Lyon, France, 1991.

[Dye90] M. Dyer. Reflection subgroups of Coxeter systems. *J. Algebra*, 135:57–73, 1990.

[Gar69] F. A. Garside. The braid groups and other groups. *Quart. J. Math. Oxford, 2^{nd} Ser*, 20:235–254, 1969.

[Gec94] M. Geck. On the character values of Iwahori-Hecke algebras of exceptional type. *Proc. London Math. Soc.*, 68:51–76, 1994.

[Gec95] M. Geck. *Beiträge zur Darstellungstheorie von Iwahori–Hecke Algebren*, volume 11 of *Aachener Beiträge zur Mathematik.* Verlag der Augustinus Buchhandlung, Aachen, 1995.

[GHL+96] M. Geck, G. Hiss, F. Lübeck, G. Malle, and G. Pfeiffer. CHEVIE-a system for computing and processing generic character tables for finite groups of lie type. *AAECC*, 7:175–210, 1996.

[GK96] M. Geck and S. Kim. Bases for the Bruhat–Chevalley order on all finite Coxeter groups. Preprint, 1996.

[GM97] M. Geck and J. Michel. On "good" elements in the conjugacy classes of finite Coxeter groups and their eigenvalues on the irreducible representations of Iwahori-Hecke algebras. *Proc. London Math. Soc.*, 1997. to appear.

[GP93] M. Geck and G. Pfeiffer. On the irreducible characters of Hecke algebras. *Advances in Math.*, 102:79–94, 1993.

[HR94] T. Halverson and A. Ram. Murnaghan-Nakayama rules for characters of Iwahori-Hecke algebras of classical type. Preprint, 1994.

[Hum90] J. E. Humphreys. *Reflections groups and Coxeter groups*, volume 29 of *Cambridge studies in advanced Math.* Cambridge University Press, 1990.

[KL79] D. A. Kazhdan and G. Lusztig. Representations of Coxeter groups and Hecke algebras. *Invent. Math.*, 53:165–184, 1979.

[Lus81] G. Lusztig. On a theorem of Benson and Curtis. *J. Algebra*, 71:490–498, 1981.

[Lus85] G. Lusztig. *Characters of reductive groups over a finite field*, volume 107 of *Annals of Math. Studies*. Princeton University Press, 1985.

[Mal96] G. Malle. Degrés relatifs des algèbres cyclotomiques associées aux groupes de réflexions complexes de dimension deux. In Cabanes [Cab96].

[Pfe94a] G. Pfeiffer. Character tables of Weyl groups in gap. *Bayreuther Math. Schr.*, 47:165–222, 1994.

[Pfe94b] G. Pfeiffer. Young characters on Coxeter basis elements of Iwahori-Hecke algebras and a Murnaghan-Nakayama formula. *J. Algebra*, 168:525–535, 1994.

[Pfe96] G. Pfeiffer. Character values of Iwahori-Hecke algebras of type B. In Cabanes [Cab96].

[Ram91] A. Ram. A Frobenius formula for the characters of the Hecke algebras. *Invent. Math.*, 106:461–488, 1991.

[Spr81] T. A. Springer. *Linear algebraic groups*. Birkhäuser, Basel, 1981.

[ST54] G. C. Shephard and J. A. Todd. Finite unitary reflection groups. *Canad. J. Math.*, 6:274–304, 1954.

[Ste89] J. R. Stembridge. On the eigenvalues of representations of reflection groups and wreath products. *Pacific J. Math.*, 140:353–396, 1989.