LAGUNA

Lie AlGebras and UNits of group Algebras

Version 3.8.0

24 September 2017

Victor Bovdi
Alexander Konovalov
Richard Rossmanith
Csaba Schneider

Victor Bovdi Email: vbovdi@science.unideb.hu
Address: Institute of Mathematics and Informatics
University of Debrecen
P.O.Box 12, Debrecen, H-4010 Hungary

Alexander Konovalov Email: alexander.konovalov@st-andrews.ac.uk
Homepage: https://alexk.host.cs.st-andrews.ac.uk
Address: School of Computer Science
University of St Andrews
Jack Cole Building, North Haugh,
St Andrews, Fife, KY16 9SX, Scotland

Csaba Schneider Email: csaba.schneider@sztaki.hu
Homepage: http://www.sztaki.hu/~schneider
Address: Informatics Laboratory
Computer and Automation Research Institute
The Hungarian Academy of Sciences
1111 Budapest, Lagymanyosi u. 11, Hungary

Abstract

The title "LAGUNA" stands for "Lie AlGebras and UNits of group Algebras". This is the new name of the GAP4 package LAG, which is thus replaced by LAGUNA.

LAGUNA extends the GAP functionality for computations in group rings. Besides computing some general properties and attributes of group rings and their elements, LAGUNA is able to perform two main kinds of computations. Namely, it can verify whether a group algebra of a finite group satisfies certain Lie properties; and it can calculate the structure of the normalized unit group of a group algebra of a finite p-group over the field of p elements.

Copyright

Acknowledgements

Some of the features of LAGUNA were already included in the GAP4 package LAG written by the third author, Richard Rossmanith. The three other authors first would like to thank Greg Gamble for maintaining LAG and for upgrading it from version 2.0 to version 2.1, and Richard Rossmanith for allowing them to update and extend the LAG package. We are also grateful to Wolfgang Kimmerle for organizing the workshop "Computational Group and Group Ring Theory" (University of Stuttgart, 28–29 November, 2002), which allowed us to meet and have fruitful discussions that led towards the final LAGUNA release.

We are all very grateful to the members of the GAP team: Thomas Breuer, Willem de Graaf, Alexander Hulpke, Stefan Kohl, Steve Linton, Frank Lübeck, Max Neunhöffer and many other colleagues for helpful comments and advise. We acknowledge very much Herbert Pahlings for communicating the package and the referee for careful testing LAGUNA and useful suggestions.

A part of the work on upgrading LAG to LAGUNA was done in 2002 during Alexander Konovalov's visits to Debrecen, St Andrews and Stuttgart Universities. He would like to express his gratitude to Adalbert Bovdi and Victor Bovdi, Colin Campbell, Edmund Robertson and Steve Linton, Wolfgang Kimmerle, Martin Hertweck and Stefan Kohl for their warm hospitality, and to the NATO Science Fellowship Program, to the London Mathematical Society and to the DAAD for the support of these visits.

Contents

1 Introduction **4**
- 1.1 General aims . . . 4
- 1.2 General computations in group rings . . . 4
- 1.3 Computations in the normalized unit group . . . 5
- 1.4 Computing Lie properties of the group algebra . . . 5
- 1.5 Installation and system requirements . . . 5

2 A sample calculation with LAGUNA **6**

3 The basic theory behind LAGUNA **12**
- 3.1 Notation and definitions . . . 12
- 3.2 p-modular group algebras . . . 13
- 3.3 Polycyclic generating set for V . . . 13
- 3.4 Computing the canonical form . . . 14
- 3.5 Computing a power commutator presentation for V . . . 15
- 3.6 Verifying Lie properties of FG . . . 15

4 LAGUNA functions **16**
- 4.1 General functions for group algebras . . . 16
- 4.2 Operations with group algebra elements . . . 18
- 4.3 Important attributes of group algebras . . . 23
- 4.4 Computations with the unit group . . . 27
- 4.5 The Lie algebra of a group algebra . . . 34
- 4.6 Other commands . . . 43

References **46**

Index **47**

Chapter 1

Introduction

1.1 General aims

LAGUNA – Lie AlGebras and UNits of group Algebras – is the new name of the GAP4 package LAG. The LAG package arose as a byproduct of the third author's PhD thesis [Ros97]. Its first version was ported to GAP4 and was brought into the standard GAP4 package format during his visit to St Andrews in September 1998.

The main objective of LAG is to deal with Lie algebras associated with some associative algebras, and, in particular, Lie algebras of group algebras. Using LAG it is possible to verify some properties or calculate certain Lie ideals of such Lie algebras very efficiently, due to their special structure. In the current version of LAGUNA the main part of the Lie algebra functionality is heavily built on the previous LAG releases.

The GAP4 package LAGUNA also extends the GAP functionality for calculations with units of modular group algebras. In particular, using this package, one can check whether an element of such a group algebra is invertible. LAGUNA also contains an implementation of an efficient algorithm to calculate the (normalized) unit group of the group algebra of a finite p-group over the field of p elements. Thus, the present version of LAGUNA provides a part of the functionality of the SISYPHOS program, which was developed by Martin Wursthorn to study the modular isomorphism problem; see [Wur93].

The corresponding functions of LAGUNA use the same algorithmic and theoretical approach as those in SISYPHOS. The reason why we reimplemented the normalised unit group algorithms in the LAGUNA package is that SISYPHOS has no interface to GAP4, and, even in GAP3, it is cumbersome to use the SISYPHOS output for further computation with the normalised unit group. For instance, using SISYPHOS with its GAP3 interface, it is difficult to embed a finite p-group into the normalized unit group of its group algebra over the field of p elements, but this can easily be done with LAGUNA.

1.2 General computations in group rings

The LAGUNA package provides a set of functions to carry out some basic computations with a group ring and its elements. Among other things, LAGUNA provides elementary functions to compute such basic notions as support, length, trace and augmentation of an element. For modular group algebras of finite p-groups LAGUNA is able to calculate the power-structure of the augmentation ideal, which is useful for the construction of the normalised unit group; see Sections 4.1–4.3 for more details.

1.3 Computations in the normalized unit group

One of the aims of the LAGUNA package is to carry out efficient computations in the normalised unit group of the group algebra FG of a finite p-group G over the field F of p elements. If U is the unit group of FG then it is easy to see that U is the direct product of F^* and $V(FG)$, where F^* is the multiplicative group of F, and $V(FG)$ is the group of normalised units. A unit of FG of the form $\alpha_1 \cdot g_1 + \alpha_2 \cdot g_2 + \cdots + \alpha_k \cdot g_k$ with $\alpha_i \in F$ and $g_i \in G$ is said to be normalised if the sum $\alpha_1 + \alpha_2 + \cdots + \alpha_k$ is equal to 1.

It is well-known that the normalised unit group V has order $|F|^{|G|-1}$, and so V is a finite p-group. Thus computing V efficiently means to compute a polycyclic presentation for V. For the theory of polycyclic presentations refer to [Sim94, Chapter 9]. For this computation we use an algorithm that was also used in the SISYPHOS package. For a brief description see Chapter 3. The functions that compute the structure of the normalised unit group are described in Section 4.4.

1.4 Computing Lie properties of the group algebra

The functions that are used to compute Lie properties of p-modular group algebras were already included in the previous versions of LAG. The bracket operation $[\cdot,\cdot]$ on a p-modular group algebra FG is defined by $[a,b] = ab - ba$. It is well-known and very easy to check that $(FG,+,[\cdot,\cdot])$ is a Lie algebra. Then we may ask what kind of Lie algebra properties are satisfied by FG. The results in [LR86], [PPS73], and [Ros00] give fast, practical algorithms to check whether the Lie algebra FG is abelian, nilpotent, soluble, centre-by-metabelian, etc. The functions that implement these algorithms are described in Section 4.5.

1.5 Installation and system requirements

LAGUNA does not use external binaries and, therefore, works without restrictions on the type of the operating system. It is designed for GAP4.4 and no compatibility with previous releases of GAP4 is guaranteed.

To use the LAGUNA online help it is necessary to install the GAP4 package GAPDoc by Frank Lübeck and Max Neunhöffer, which is available from the GAP site or from `http://www.math.rwth-aachen.de/~Frank.Luebeck/GAPDoc/`.

LAGUNA is distributed in standard formats (`zoo`, `tar.gz`, `tar.bz2`, `-win.zip`) and can be obtained from `http://www.cs.st-andrews.ac.uk/~alexk/laguna/`. To unpack the archive `laguna-X.X.X.zoo` you need the program `unzoo`, which can be obtained from the GAP homepage `http://www.gap-system.org/` (see section 'Distribution'). To install LAGUNA, copy this archive into the `pkg` subdirectory of your GAP4.4 installation. The subdirectory `laguna` will be created in the `pkg` directory after the following command:

```
unzoo -x laguna-X.X.X.zoo
```

Chapter 2

A sample calculation with LAGUNA

Before explaining the theory behind the LAGUNA package we present a sample calculation to show the reader what LAGUNA is able to compute. We will carry out some calculations in the group algebra of the dihedral group of order 16 over the field of two elements. First we create this modular group algebra.

Example

```
gap> K := GF( 2 );
GF(2)
gap> G := DihedralGroup( 16 );
<pc group of size 16 with 4 generators>
gap> KG := GroupRing( K, G );
<algebra-with-one over GF(2), with 4 generators>
```

The group algebra KG has some properties and attributes that are direct consequences of its definition. These can be checked very quickly.

Example

```
gap> IsGroupAlgebra( KG );
true
gap> IsPModularGroupAlgebra( KG );
true
gap> IsFModularGroupAlgebra( KG );
true
gap> UnderlyingGroup( KG );
<pc group of size 16 with 4 generators>
gap> LeftActingDomain( KG );
GF(2)
```

Since KG is naturally a group algebra, the information provided by LeftActingDomain can also be obtained using two other functions as follows.

Example

```
gap> UnderlyingRing( KG );
GF(2)
gap> UnderlyingField( KG );
GF(2)
```

Let us construct a certain element of the group algebra. For example, we take a minimal generating system of the group G and find the corresponding elements in KG.

Example

```
gap> MinimalGeneratingSet( G );
[ f1, f2 ]
gap> l := List( last, g -> g^Embedding( G, KG ) );
[ (Z(2)^0)*f1, (Z(2)^0)*f2 ]
```

Now we construct an element x as follows.

Example

```
gap> a :=l[1]; b:=l[2]; # a and b are images of group generators in KG
(Z(2)^0)*f1
(Z(2)^0)*f2
gap> e := One( KG );    # for convenience, we denote the identity by e
(Z(2)^0)*<identity> of ...
gap> x := ( e + a ) * ( e + b );
(Z(2)^0)*<identity> of ...+(Z(2)^0)*f1+(Z(2)^0)*f2+(Z(2)^0)*f1*f2
```

We may investigate some of the basic properties of our element.

Example

```
gap> Support( x );
[ <identity> of ..., f1, f2, f1*f2 ]
gap> CoefficientsBySupport( x );
[ Z(2)^0, Z(2)^0, Z(2)^0, Z(2)^0 ]
gap> Length( x );
4
gap> TraceOfMagmaRingElement( x );
Z(2)^0
```

We can also calculate the augmentation of x, which is defined as the sum of its coefficients.

Example

```
gap> Augmentation( x );
0*Z(2)
gap> IsUnit( KG, x );
false
```

Since the augmentation of x is zero, x is not invertible, but 1+x is. This is again very easy to check.

Example

```
gap> y := e + x;
(Z(2)^0)*f1+(Z(2)^0)*f2+(Z(2)^0)*f1*f2
gap> IsUnit( KG, y );
true
```

LAGUNA can calculate the inverse of 1+x very quickly.

Example

```
gap> y^-1;
(Z(2)^0)*f1+(Z(2)^0)*f2+(Z(2)^0)*f3+(Z(2)^0)*f4+(Z(2)^0)*f1*f2+(Z(2)^
0)*f1*f3+(Z(2)^0)*f1*f4+(Z(2)^0)*f2*f4+(Z(2)^0)*f1*f2*f4+(Z(2)^0)*f2*f3*f4+(
Z(2)^0)*f1*f2*f3*f4
gap> y * y^-1;
(Z(2)^0)*<identity> of ...
```

We may also want to check whether y is symmetric, that is, whether it is invariant under the classical involution; or whether it is unitary, that is, whether the classical involution inverts y. We find that y is neither.

Example

```
gap> Involution( y );
(Z(2)^0)*f1+(Z(2)^0)*f1*f2+(Z(2)^0)*f2*f3*f4
gap> y = Involution( y );
false
gap> IsSymmetric( y );
false
gap> y * Involution( y );
(Z(2)^0)*<identity> of ...+(Z(2)^0)*f2+(Z(2)^0)*f2*f3*f4
gap> IsUnitary( y );
false
```

Now we calculate some important ideals of KG. First we obtain the augmentation ideal which is the set of elements with augmentation zero. In our case the augmentation ideal of KG coincides with the radical of KG, and this is taken into account in LAGUNA.

Example

```
gap> AugmentationIdeal( KG );
<two-sided ideal in <algebra-with-one over GF(2), with 4 generators>,
  (dimension 15)>
gap> RadicalOfAlgebra( KG ) = AugmentationIdeal( KG );
true
```

It is well-known that the augmentation ideal of KG is a nilpotent ideal. Using Jennings' theory on dimension subgroups, we can obtain its nilpotency index without immediate calculation of its powers. This is implemented in LAGUNA.

Example

```
gap> AugmentationIdealNilpotencyIndex( KG );
9
```

On the other hand, we can also calculate the powers of the augmentation ideal.

Example

```
gap> AugmentationIdealPowerSeries( KG );
[ <algebra of dimension 15 over GF(2)>, <algebra of dimension 13 over GF(2)>,
  <algebra of dimension 11 over GF(2)>, <algebra of dimension 9 over GF(2)>,
  <algebra of dimension 7 over GF(2)>, <algebra of dimension 5 over GF(2)>,
  <algebra of dimension 3 over GF(2)>, <algebra of dimension 1 over GF(2)>,
  <algebra over GF(2)> ]
```

We see that the length of this list is exactly the nilpotency index of the augmentation ideal of KG.

Now let's work with the unit group of KG. First we calculate the normalized unit group, which is the set of elements with augmentation one. The generators of the unit group are obtained as explained in Chapter 3. This can be computed very quickly, but further computation with this group is very inefficient.

Example

```
gap> V := NormalizedUnitGroup( KG );
<group of size 32768 with 15 generators>
```

In order to make our computation in the normalised unit group efficient, we calculate a power-commutator presentation for this group.

Example

```
gap> W := PcNormalizedUnitGroup( KG );
<pc group of size 32768 with 15 generators>
```

GAP has many efficient and practical algorithms for groups given by a power-commutator presentation. In order to use these algorithms to carry out computation in the normalised unit group, we need to set up isomorphisms between the outputs of NormalizedUnitGroup and PcNormalizedUnitGroup.

The first isomorphism maps NormalizedUnitGroup(KG) onto the polycyclically presented PcNormalizedUnitGroup(PC). Let's find the images of the elements of the group G in W.

Example

```
gap> t := NaturalBijectionToPcNormalizedUnitGroup( KG );
MappingByFunction( <group of size 32768 with 15 generators>, <pc group of size\
 32768 with 15 generators>, function( x ) ... end )
gap> List( AsList( G ), x -> ( x^Embedding( G, KG ) )^t );
[ <identity> of ..., f2, f1, f3, f7, f1*f2*f3, f2*f3, f2*f7, f1*f3, f1*f7,
  f3*f7, f1*f2*f7, f1*f2*f3*f7, f2*f3*f7, f1*f3*f7, f1*f2 ]
```

The second isomorphism is the inverse of the first.

Example

```
gap> f := NaturalBijectionToNormalizedUnitGroup( KG );
[ f1, f2, f3, f4, f5, f6, f7, f8, f9, f10, f11, f12, f13, f14, f15 ] ->
[ (Z(2)^0)*f2, (Z(2)^0)*f1, (Z(2)^0)*f3,
  (Z(2)^0)*f1+(Z(2)^0)*f2+(Z(2)^0)*f1*f2,
  (Z(2)^0)*f2+(Z(2)^0)*f3+(Z(2)^0)*f2*f3,
  (Z(2)^0)*f1+(Z(2)^0)*f3+(Z(2)^0)*f1*f3, (Z(2)^0)*f4,
  (Z(2)^0)*f1+(Z(2)^0)*f2+(Z(2)^0)*f3+(Z(2)^0)*f1*f2+(Z(2)^0)*f1*f3+(Z(2)^
    0)*f2*f3+(Z(2)^0)*f1*f2*f3, (Z(2)^0)*f2+(Z(2)^0)*f4+(Z(2)^0)*f2*f4,
  (Z(2)^0)*f1+(Z(2)^0)*f4+(Z(2)^0)*f1*f4,
  (Z(2)^0)*f3+(Z(2)^0)*f4+(Z(2)^0)*f3*f4,
  (Z(2)^0)*f1+(Z(2)^0)*f2+(Z(2)^0)*f4+(Z(2)^0)*f1*f2+(Z(2)^0)*f1*f4+(Z(2)^
    0)*f2*f4+(Z(2)^0)*f1*f2*f4, (Z(2)^0)*f2+(Z(2)^0)*f3+(Z(2)^0)*f4+(Z(2)^
    0)*f2*f3+(Z(2)^0)*f2*f4+(Z(2)^0)*f3*f4+(Z(2)^0)*f2*f3*f4,
  (Z(2)^0)*f1+(Z(2)^0)*f3+(Z(2)^0)*f4+(Z(2)^0)*f1*f3+(Z(2)^0)*f1*f4+(Z(2)^
    0)*f3*f4+(Z(2)^0)*f1*f3*f4, (Z(2)^0)*f1+(Z(2)^0)*f2+(Z(2)^0)*f3+(Z(2)^
    0)*f4+(Z(2)^0)*f1*f2+(Z(2)^0)*f1*f3+(Z(2)^0)*f1*f4+(Z(2)^0)*f2*f3+(Z(2)^
    0)*f2*f4+(Z(2)^0)*f3*f4+(Z(2)^0)*f1*f2*f3+(Z(2)^0)*f1*f2*f4+(Z(2)^
    0)*f1*f3*f4+(Z(2)^0)*f2*f3*f4+(Z(2)^0)*f1*f2*f3*f4 ]
```

For example, we may calculate the conjugacy classes of the group `W`, and then map their representatives back into the group algebra.

Example

```
gap> cc := ConjugacyClasses( W );;
gap> Length( cc );
848
gap> Representative( cc[ Length( cc ) ] );
f1*f2*f4*f6*f12*f15
gap> last^f;
(Z(2)^0)*<identity> of ...+(Z(2)^0)*f2+(Z(2)^0)*f4+(Z(2)^0)*f1*f2+(Z(2)^
0)*f1*f3+(Z(2)^0)*f1*f4+(Z(2)^0)*f2*f3+(Z(2)^0)*f2*f4+(Z(2)^0)*f3*f4+(Z(2)^
0)*f1*f2*f3+(Z(2)^0)*f1*f3*f4
```

Having a power-commutator presentation of the normalised unit group, we may use the full power of the GAP functionality for such groups. For example, the lower central series can be calculated very quickly.

Example

```
gap> LowerCentralSeries( W );
[ <pc group of size 32768 with 15 generators>,
  Group([ f3, f5*f8*f10*f12*f13*f14*f15, f6*f8*f12*f14*f15, f7, f9*f12,
      f10*f14, f11*f13, f13*f14, f14*f15 ]),
  Group([ f7, f9*f12, f10*f15, f11*f15, f13*f15, f14*f15 ]),
  Group([ f11*f15, f13*f15, f14*f15 ]), Group([ <identity> of ... ]) ]
```

Let's now compute, for instance, a minimal system of generators of the centre of the normalised unit group. First we carry out the computation in the group which is determined by the power-commutator presentation, then we map the result into our group algebra.

Example

```
gap> C := Centre( W );
Group([ f3*f5*f13*f15, f7, f15, f13*f15, f14*f15, f11*f13*f14*f15 ])
gap> m := MinimalGeneratingSet( C );
[ f7*f13*f14*f15, f13*f14*f15, f7*f11*f14*f15, f15, f3*f5*f14 ]
gap> List( m, g -> g^f );
[ (Z(2)^0)*<identity> of ...+(Z(2)^0)*f3+(Z(2)^0)*f1*f2+(Z(2)^0)*f3*f4+(Z(2)^
    0)*f1*f2*f3+(Z(2)^0)*f1*f2*f4+(Z(2)^0)*f1*f2*f3*f4,
  (Z(2)^0)*f3+(Z(2)^0)*f4+(Z(2)^0)*f1*f2+(Z(2)^0)*f3*f4+(Z(2)^0)*f1*f2*f3+(
    Z(2)^0)*f1*f2*f4+(Z(2)^0)*f1*f2*f3*f4, (Z(2)^0)*<identity> of ...+(Z(2)^
    0)*f2+(Z(2)^0)*f3+(Z(2)^0)*f1*f2+(Z(2)^0)*f2*f3+(Z(2)^0)*f2*f4+(Z(2)^
    0)*f3*f4+(Z(2)^0)*f1*f2*f3+(Z(2)^0)*f1*f2*f4+(Z(2)^0)*f2*f3*f4+(Z(2)^
    0)*f1*f2*f3*f4, (Z(2)^0)*f1+(Z(2)^0)*f2+(Z(2)^0)*f3+(Z(2)^0)*f4+(Z(2)^
    0)*f1*f2+(Z(2)^0)*f1*f3+(Z(2)^0)*f1*f4+(Z(2)^0)*f2*f3+(Z(2)^0)*f2*f4+(
    Z(2)^0)*f3*f4+(Z(2)^0)*f1*f2*f3+(Z(2)^0)*f1*f2*f4+(Z(2)^0)*f1*f3*f4+(Z(2)^
    0)*f2*f3*f4+(Z(2)^0)*f1*f2*f3*f4, (Z(2)^0)*<identity> of ...+(Z(2)^0)*f1+(
    Z(2)^0)*f3+(Z(2)^0)*f1*f3+(Z(2)^0)*f1*f4+(Z(2)^0)*f2*f3+(Z(2)^0)*f2*f4+(
    Z(2)^0)*f3*f4+(Z(2)^0)*f1*f3*f4 ]
```

We finish our example by calculating some properties of the Lie algebra associated with `KG`. This example needs no further explanation.

Example

```
gap> L := LieAlgebra( KG );
#I  LAGUNA package: Constructing Lie algebra ...
<Lie algebra of dimension 16 over GF(2)>
gap> D := LieDerivedSubalgebra( L );
#I  LAGUNA package: Computing the Lie derived subalgebra ...
<Lie algebra of dimension 9 over GF(2)>
gap> LC := LieCentre( L );
<Lie algebra of dimension 7 over GF(2)>
gap> LieLowerNilpotencyIndex( KG );
5
gap> LieUpperNilpotencyIndex( KG );
5
gap> IsLieAbelian( L );
false
gap> IsLieSolvable( L );
#I  LAGUNA package: Checking Lie solvability ...
true
gap> IsLieMetabelian( L );
false
gap> IsLieCentreByMetabelian( L );
true
```

Chapter 3

The basic theory behind LAGUNA

In this chapter we describe the theory that is behind the algorithms used by LAGUNA.

3.1 Notation and definitions

Let G be a group and F a field. Then the *group algebra* FG consists of the set of formal linear combinations of the form

$$\sum_{g\in G} \alpha_g g, \qquad \alpha_g \in F$$

where all but finitely many of the α_g are zero. The group algebra FG is an F-algebra with the obvious operations. Clearly, $\dim FG = |G|$.

The *augmentation homomorphism* $\chi : FG \to F$ is defined by

$$\chi\left(\sum_{g\in G} \alpha_g g\right) = \sum_{g\in G} \alpha_g.$$

It is easy to see that χ is indeed a homomorphism onto F. The kernel of χ is called the *augmentation ideal* of FG. The augmentation ideal is denoted $A(FG)$, or simply A when there is no danger of confusion. It follows from the isomorphism theorems that $\dim A(FG) = \dim FG - 1 = |G| - 1$. Another way to write the augmentation ideal is

$$A(FG) = \left\{ \sum_{g\in G} \alpha_g g \mid \sum_{g\in G} \alpha_g = 0 \right\}.$$

An invertible element of FG is said to be a *unit*. Clearly the elements of G and the non-zero elements of F are units. The set of units in FG is a group with respect to the multiplication of FG. The *unit group* of FG is denoted $U(FG)$ or simply U when there is no risk of confusion. A unit u is said to be *normalised* if $\chi(u) = 1$. The set of normalised units forms a subgroup of the unit group, and is referred to as the *normalised unit group*. The normalised unit group of FG is denoted $V(FG)$, or simply V. It is easy to prove that $U(FG) = F^* \times V(FG)$ where F^* denotes the multiplicative group of F.

3.2 p-modular group algebras

A group algebra FG is said to be p-modular if F is the field of characteristic p, and G is a finite p-group. A lot of information about the structure of p-modular group algebras can be found in [HB82, Chapter VIII]. In a p-modular group algebra we have that an element u is a unit if and only if $\chi(u) \neq 0$. Hence the normalised unit group V consists of all elements of FG with augmentation 1. In other words V is a coset of the augmentation ideal, namely $V = 1 + A$. This also implies that $|V| = |A| = |F|^{|G|-1}$, and so V is a finite p-group.

One of the aims of the LAGUNA package is to compute a power-commutator presentation for the normalised unit group in the case when G is a finite p-group and F is a field of p elements. Such a presentation is given by generators $y_1, \ldots, y_{|G|-1}$ and two types of relations: $y_i^p = (y_{i+1})^{\alpha_{i,i+1}} \cdots (y_{|G|-1})^{\alpha_{i,|G|-1}}$ for $1 \leq i \leq |G|-1$, and $[y_j, y_i] = (y_{j+1})^{\alpha_{j,i,j+1}} \cdots (y_{|G|-1})^{\alpha_{j,i,|G|-1}}$ for $1 \leq i < j \leq |G|-1$, where the exponents $\alpha_{i,k}$ and $\alpha_{i,j,k}$ are elements of the set $\{0, \ldots, p-1\}$. Having such a presentation, it is possible to carry out efficient computations in the finite p-group V; see [Sim94, Chapter 9].

3.3 Polycyclic generating set for V

Let G be a finite p-group and F the field of p elements. Our aim is to construct a power-commutator presentation for $V = V(FG)$. We noted earlier that $V = 1 + A$, where A is the augmentation ideal. We use this piece of information and construct a polycyclic generating set for V using a suitable basis for A. Before constructing this generating set, we note that A is a nilpotent ideal in FG. In other words there is some c such that $A^c \neq 0$ but $A^{c+1} = 0$. Hence we can consider the following series of ideals in A:

$$A \rhd A^2 \rhd \cdots \rhd A^c \rhd A^{c+1} = 0.$$

It is clear that a quotient $A^i/A^{i+1}$of this chain has trivial multiplication, that is, such a quotient is a nil-ring. The chain A^i gives rise to a series of normal subgroups in V:

$$V = 1 + A \rhd 1 + A^2 \rhd \cdots \rhd 1 + A^c \rhd 1 + A^{c+1} = 1.$$

It is easy to see that the chain $1 + A^i$ is central, that is, $(1 + A^i)/(1 + A^{i+1}) \leq Z((1 + A)/(1 + A^{i+1}))$.

Now we show how to compute a basis for A^i that gives a polycyclic generating set for $1 + A^i$. Let

$$G = G_1 \rhd G_2 \rhd \cdots \rhd G_k \rhd G_{k+1} = 1$$

be the *Jennings series* of G. That is, $G_{i+1} = [G_i, G]G_{j^p}$ where j is the smallest non-negative integer such that $j \geq i/p$. For all $i \leq k$ select elements $x_{i,1}, \ldots, x_{i,l_i}$ of G_i such that $\{x_{i,1}G_{i+1}, \ldots, x_{i,l_i}G_{i+1}\}$ is a minimal generating set for the elementary abelian group G_i/G_{i+1}. For the Jennings series it may happen that $G_i = G_{i+1}$ for some i. In this case we choose an empty generating set for the quotient G_i/G_{i+1} and $l_i = 0$. Then the set $x_{1,1}, \ldots, x_{1,l_1}, \ldots, x_{k,1}, \ldots, x_{k,l_k}$ is said to be a *dimension basis* for G. The *weight* of a dimension basis element $x_{i,j}$ is i.

A non-empty product

$$u = (x_{1,1} - 1)^{\alpha_{1,1}} \cdots (x_{1,l_1} - 1)^{\alpha_{1,l_1}} \cdots (x_{k,1} - 1)^{\alpha_{k,1}} \cdots (x_{k,l_k} - 1)^{\alpha_{k,l_k}}$$

where $0 \leq \alpha_{i,j} \leq p-1$ is said to be *standard*. Clearly, a standard product is an element of the augmentation ideal A. The weight of the standard product u is

$$\sum_{i=1}^{k} i(\alpha_{i,1} + \cdots + \alpha_{i,l_i}).$$

The total number of standard products is $|G|-1$.

LEMMA ([HB82, Theorem VIII.2.6]). For $i \leq c$, the set S_i of standard products of weight at least i forms a basis for A^i. Moreover, the set $1+S_i = \{1+s \mid s \in S_i\}$ is a polycyclic generating set for $1+A^i$. In particular $1+S_1$ is a polycyclic generating set for V.

A basis for A consisting of the standard products is referred to as a *weighted basis*. Note that a weighted basis is a basis for the augmentation ideal, and not for the whole group algebra.

Let $x_1,\ldots,x_{|G|-1}$ denote the standard products where we choose the indices so that the weight of x_i is not larger than the weight of x_{i+1} for all i, and set $y_i = 1+x_i$. Then every element v of V can be uniquely written in the form

$$v = y_1^{\alpha_1}\cdots(y_{|G|-1})^{\alpha_{|G|-1}}, \quad \alpha_1,\ldots,\alpha_{|G|-1} \in \{0,\ldots,p-1\}.$$

This expression is called the *canonical form* of v. We note that by adding a generator of F^* to the set $y_1,\ldots,y_{|G|-1|}$ we can obtain a polycyclic generating set for the unit group U.

3.4 Computing the canonical form

We show how to compute the canonical form of a normalised unit with respect to the polycyclic generating set $y_1,\ldots,y_{|G|-1}$. We use the following elementary lemma.

LEMMA. Let $i \leq c$ and suppose that $w \in A^i$. Assume that $x_{s_i}, x_{s_i+1}\ldots,x_{r_i}$ are the standard products with weight i and for $s_i \leq j \leq r_i$ set $y_j = 1+x_j$. Then for all $\alpha_{s_i},\ldots,\alpha_{r_i} \in \{0,\ldots,p-1\}$ we have that

$$w \equiv \alpha_{s_i}x_{s_i} + \cdots + \alpha_{r_i}x_{r_i} \mod A^{i+1}$$

if an only if

$$1+w \equiv (y_{s_i})^{\alpha_{s_i}}\cdots(y_{r_i})^{\alpha_{r_i}} \mod 1+A^{i+1}.$$

Suppose that w is an element of the augmentation ideal A and $1+w$ is a normalised unit. Let $x_1,\ldots,x_{r_1}$ be the standard products of weight 1, and let $y_1,\ldots,y_{r_1}$ be the corresponding elements in the polycyclic generating set. Then using the previous lemma, we find $\alpha_1,\ldots,\alpha_{r_1}$ such that

$$w \equiv \alpha_1x_1 + \cdots + \alpha_{r_1}x_{r_1} \mod A^2,$$

and so

$$1+w \equiv (y_1)^{\alpha_1}\cdots(y_{r_1})^{\alpha_{r_1}} \mod 1+A^2.$$

Now we have that $1+w = (y_1)^{\alpha_1}\cdots(y_{r_1})^{\alpha_{r_1}}(1+w_2)$ for some $w_2 \in A^2$. Then suppose that $x_{s_2}, x_{s_2+1},\ldots,x_{r_2}$ are the standard products of weight 2. We find $\alpha_{s_2},\ldots,\alpha_{r_2}$ such that

$$w_2 \equiv \alpha_{s_2}x_{s_2} + \cdots + \alpha_{r_2}x_{r_2} \mod A^3.$$

Then the lemma above implies that

$$1+w_2 \equiv (y_{s_2})^{\alpha_{s_2}}\cdots(y_{r_2})^{\alpha_{r_2}} \mod 1+A^3.$$

Thus $1+w_2 = (y_{s_2})^{\alpha_{s_2}}\cdots(y_{r_2})^{\alpha_{r_2}}(1+w_3)$ for some $w_3 \in A^3$, and so $1+w = (y_1)^{\alpha_1}\cdots(y_{r_1})^{\alpha_{r_1}}(y_{s_2})^{\alpha_{s_2}}\cdots(y_{r_2})^{\alpha_{r_2}}(1+w_3)$. We repeat this process, and after c steps we obtain the canonical form for the element $1+w$.

3.5 Computing a power commutator presentation for V

Using the procedure in the previous section, it is easy to compute a power commutator presentation for the normalized unit group V of a p-modular group algebra over the field of p elements. First we compute the polycyclic generating sequence $y_1, \ldots, y_{|G|-1}$ as in Section 3.3. Then for each y_i and for each y_j, y_i such that $i < j$ we compute the canonical form for y_i^p and $[y_j, y_i]$ as described in Section 3.4.

Once a power-commutator presentation for V is constructed, it is easy to obtain a polycyclic presentation for the unit group U by adding an extra central generator y corresponding to a generator of the cyclic group F^* and enforcing that $y^{p-1} = 1$.

3.6 Verifying Lie properties of FG

If FG is a group algebra then one can consider the Lie bracket operation defined by $[a,b] = ab - ba$. Then it is well-known that FG with respect to the scalar multiplication, the addition, and the bracket operation becomes a Lie algebra over F. This Lie algebra is also denoted FG. Some Lie properties of such Lie algebras can be computed very efficiently. In particular, it can be verified whether the Lie algebra FG is nilpotent, soluble, metabelian, centre-by-metabelian. Fast algorithms that achieve these goals are described in [LR86], [PPS73], and [Ros00].

Chapter 4

LAGUNA functions

4.1 General functions for group algebras

4.1.1 IsGroupAlgebra

▷ `IsGroupAlgebra(`*KG*`)` (property)

A group ring over a field is called a group algebra. For a group ring *KG*, `IsGroupAlgebra` returns `true`, if the underlying ring of *KG* is a field; `false` is returned otherwise. This property will be set automatically for every group ring created by the function `GroupRing`.

Example

```
gap> IsGroupAlgebra( GroupRing( GF( 2 ), DihedralGroup( 16 ) ) );
true
gap> IsGroupAlgebra( GroupRing( Integers, DihedralGroup( 16 ) ) );
false
```

4.1.2 IsFModularGroupAlgebra

▷ `IsFModularGroupAlgebra(`*KG*`)` (property)

A group algebra KG over a field K is called *modular*, if the characteristic of the field K divides the order of some element in G. For a group algebra *KG* of a finite group G, `IsModularGroupAlgebra` returns `true`, if *KG* is modular according to this definition; `false` is returned otherwise. This property will be set automatically for every group algebra, created by the function `GroupRing`.

Example

```
gap> IsFModularGroupAlgebra( GroupRing( GF( 2 ), SymmetricGroup( 6 ) ) );
true
gap> IsFModularGroupAlgebra( GroupRing( GF( 2 ), CyclicGroup( 3 ) ) );
false
```

4.1.3 IsPModularGroupAlgebra

▷ `IsPModularGroupAlgebra(`*KG*`)` (property)

A group algebra KG is said to be p-modular, if K is a field of characteristic p and G is a finite p-group for the same prime p. For a group algebra *KG* of a finite group G, `IsPModularGroupAlgebra` returns `true`, if *KG* is p-modular according to this definition; `false` is returned otherwise. This property will be set automatically for every group algebra, created by the function `GroupRing`.

Example

```
gap> IsPModularGroupAlgebra( GroupRing( GF( 2 ), DihedralGroup( 16 ) ) );
true
gap> IsPModularGroupAlgebra( GroupRing( GF( 2 ), SymmetricGroup( 6 ) ) );
false
```

4.1.4 UnderlyingGroup (of a group ring)

▷ `UnderlyingGroup(`*KG*`)` (attribute)

Returns: the underlying group of a group ring

This attribute stores the underlying group of a group ring *KG*. In fact, it refers to the attribute `UnderlyingMagma` which returns the same result, and was introduced for group rings for convenience, and for teaching purposes.

Example

```
gap> KG := GroupRing( GF ( 2 ), DihedralGroup( 16 ) );
<algebra-with-one over GF(2), with 4 generators>
gap> G := UnderlyingGroup( KG );
<pc group of size 16 with 4 generators>
```

4.1.5 UnderlyingRing

▷ `UnderlyingRing(`*KG*`)` (attribute)

Returns: the underlying ring of a group ring

This attribute stores the underlying ring of a group ring *KG*. In fact, it refers to the attribute `LeftActingDomain` which returns the same result, and was introduced for group rings for convenience, and for teaching purposes.

Example

```
gap> KG := GroupRing( GF( 2 ), DihedralGroup( 16 ) );
<algebra-with-one over GF(2), with 4 generators>
gap> UnderlyingRing( KG );
GF(2)
```

4.1.6 UnderlyingField

▷ `UnderlyingField(KG)` (attribute)

Returns: the underlying field of a group algebra

This attribute stores the underlying field of a group algebra `KG`. In fact, it refers to the attribute `LeftActingDomain` which returns the same result, and was introduced for group algebras for convenience, and for teaching purposes.

Example

```
gap> KG := GroupRing( GF( 2 ), DihedralGroup( 16 ) );
<algebra-with-one over GF(2), with 4 generators>
gap> UnderlyingField( KG );
GF(2)
```

4.2 Operations with group algebra elements

4.2.1 Support

▷ `Support(x)` (attribute)

Returns: support of x as a list of elements of the underlying group

Returns the support of a group ring element x. The support of a non-zero element $x = \alpha_1 \cdot g_1 + \alpha_2 \cdot g_2 + \cdots + \alpha_k \cdot g_k$ of a group ring is the list of elements $g_i \in G$ for which the coefficient α_i is non-zero. The support of the zero element of a group ring is defined to be the empty list. This method is also applicable to elements of magma rings.

Example

```
# First we create an element x to use in in the series of examples.
# We map the minimal generating system of the group G to its group algebra
# and denote their images as a and b
gap> G:=DihedralGroup(16);; KG:=GroupRing(GF(2),G);;
gap> l := List( MinimalGeneratingSet( G ), g -> g^Embedding( G, KG ) );
[ (Z(2)^0)*f1, (Z(2)^0)*f2 ]
gap> a := l[1]; b := l[2]; e := One( KG ); # we denote the identity by e
(Z(2)^0)*f1
(Z(2)^0)*f2
(Z(2)^0)*<identity> of ...
gap> x := ( e + a ) * ( e + b );
(Z(2)^0)*<identity> of ...+(Z(2)^0)*f1+(Z(2)^0)*f2+(Z(2)^0)*f1*f2
gap> Support( x );
[ <identity> of ..., f1, f2, f1*f2 ]
```

4.2.2 CoefficientsBySupport

▷ `CoefficientsBySupport(x)` (attribute)

Returns: coefficients of support elements as list of elements of the underlying ring

Returns a list that contains the coefficients corresponding to the elements of `Support( x )` in the same order as the elements appear in `Support( x )`. This method is also applicable to elements of magma rings.

Example

```
gap> x;
(Z(2)^0)*<identity> of ...+(Z(2)^0)*f1+(Z(2)^0)*f2+(Z(2)^0)*f1*f2
gap> CoefficientsBySupport( x );
[ Z(2)^0, Z(2)^0, Z(2)^0, Z(2)^0 ]
```

4.2.3 TraceOfMagmaRingElement

▷ `TraceOfMagmaRingElement(x)` (attribute)

Returns: an element of the underlying ring

Returns the trace of a group ring element x. By definition, the trace of an element $x = \alpha_1 \cdot 1 + \alpha_2 \cdot g_2 + \cdots + \alpha_k \cdot g_k$ is equal to α_1, that is, the coefficient of the identity element in G. The trace of the zero element is zero. This method is also applicable to elements of magma rings.

Example

```
gap> x;
(Z(2)^0)*<identity> of ...+(Z(2)^0)*f1+(Z(2)^0)*f2+(Z(2)^0)*f1*f2
gap> TraceOfMagmaRingElement( x );
Z(2)^0
```

4.2.4 Length

▷ `Length(x)` (attribute)

The length of an element of a group ring x is defined as the number of elements in its support. This method is also applicable to elements of magma rings.

Example

```
gap> x;
(Z(2)^0)*<identity> of ...+(Z(2)^0)*f1+(Z(2)^0)*f2+(Z(2)^0)*f1*f2
gap> Length( x );
4
```

4.2.5 Augmentation

▷ `Augmentation(x)` (attribute)

Returns: the sum of coefficients of a group ring element

The augmentation of a group ring element $x = \alpha_1 \cdot g_1 + \alpha_2 \cdot g_2 + \cdots + \alpha_k \cdot g_k$ is the sum of its coefficients $\alpha_1 + \alpha_2 + \cdots + \alpha_k$. The method is also applicable to elements of magma rings.

Example

```
gap> x;
(Z(2)^0)*<identity> of ...+(Z(2)^0)*f1+(Z(2)^0)*f2+(Z(2)^0)*f1*f2
gap> Augmentation( x );
```

```
0*Z(2)
```

4.2.6 PartialAugmentations

▷ `PartialAugmentations(KG, x)` (operation)

Returns: a list of partial augmentations and a list of conjugacy class representatives

The partial augmentation of an element $x = \alpha_1 \cdot g_1 + \alpha_2 \cdot g_2 + \cdots + \alpha_k \cdot g_k$ of the group ring KG, corresponding to the conjugacy class of an element g from the underlying group G is the sum of coefficients α_i taken over all g_i such that g_i is conjugated to g. The function returns a list of two lists, the first one is a list of partial augmentations, and the second is a list of representatives of appropriate conjugacy classes of elements of the group G.

Example

```
gap> y := x + a*b^2;
(Z(2)^0)*<identity> of ...+(Z(2)^0)*f1+(Z(2)^0)*f2+(Z(2)^0)*f1*f2+(Z(2)^
0)*f1*f3
gap> PartialAugmentations( KG, y );
[ [ Z(2)^0, 0*Z(2), Z(2)^0, Z(2)^0 ], [ <identity> of ..., f1, f2, f1*f2 ] ]
```

4.2.7 Involution

▷ `Involution(x[[, f], s])` (operation)

Returns: an element of a group ring

Let KG be a group ring, f be a homomorphism from the group G to the unit group of the ring K. Furthermore, let s be a mapping $G \to G$, such that s^2 is the identity mapping on G and for every element $g \in G$ $f(g * s(g))$ equals $f(s(g) * g)$ and equals the identity element of the ring K. Then the involution of KG induced by f and s is defined by $\alpha_1 \cdot g_1 + \alpha_2 \cdot g_2 + \cdots + \alpha_k \cdot g_k \mapsto \alpha_1 \cdot f(g_1) \cdot s(g_1) + \alpha_2 \cdot f(g_2) \cdot s(g_2) + \cdots + \alpha_k \cdot f(g_k) \cdot s(g_k)$.

The method returns the image of x under the involution of KG induced by f and s. If the mapping f is omitted, f is assumed to map everything to the identity element of the ring K. If both mappings are omitted, it returns the result of so-called classical involution, induced by the mapping $x \mapsto x^{-1}$.

Example

```
gap> x;
(Z(2)^0)*<identity> of ...+(Z(2)^0)*f1+(Z(2)^0)*f2+(Z(2)^0)*f1*f2
gap> Involution( x );
(Z(2)^0)*<identity> of ...+(Z(2)^0)*f1+(Z(2)^0)*f1*f2+(Z(2)^0)*f2*f3*f4
gap> l := List( MinimalGeneratingSet( G ), g -> g^Embedding( G, KG ) );
[ (Z(2)^0)*f1, (Z(2)^0)*f2 ]
gap> List( l, Involution ); # check how involution acts on elements of G
[ (Z(2)^0)*f1, (Z(2)^0)*f2*f3*f4 ]
gap> List( l, g -> g^-1 );
[ (Z(2)^0)*f1, (Z(2)^0)*f2*f3*f4 ]
```

4.2.8 IsSymmetric

▷ `IsSymmetric(x)` (attribute)

An element of a group ring is called *symmetric* if it is fixed under the classical involution. This property is checked here.

Example

```
gap> IsSymmetric( x );
false
gap> IsSymmetric( x * Involution( x ) );
true
```

4.2.9 IsUnitary

▷ `IsUnitary(x)` (attribute)

A unit of a group ring is called unitary if the classical involution inverts it. This property is checked here.

Example

```
gap> IsUnitary(x);
false
gap> l:=List(MinimalGeneratingSet(G),g -> g^Embedding(G,KG));
[ (Z(2)^0)*f1, (Z(2)^0)*f2 ]
gap> List(l,IsUnitary); # check that elements of G are unitary
[ true, true ]
```

4.2.10 IsUnit

▷ `IsUnit([KG, ]x)` (method)

This method improves a standard GAP functionality for modular group algebras.

In the two-argument version the method returns `true` if x is an invertible element of the modular group algebra KG and `false` otherwise. This can be done very quickly by checking whether the augmentation of the element x is non-zero.

If the first argument is omitted, then LAGUNA constructs the group H generated by the support of x, and, if this group is a finite p-group, then checks whether the coefficients of x belong to a field F of characteristic p. If this is the case, then `IsUnit( FH, x )` is called; otherwise, standard GAP method is used.

Example

```
gap> x;
(Z(2)^0)*<identity> of ...+(Z(2)^0)*f1+(Z(2)^0)*f2+(Z(2)^0)*f1*f2
gap> IsUnit( KG, x ); # clearly, is not a unit due to augmentation zero
false
gap> y := One( KG ) + x; # this should give a unit
```

```
(Z(2)^0)*f1+(Z(2)^0)*f2+(Z(2)^0)*f1*f2
gap> IsUnit( KG, y );
true
```

4.2.11 InverseOp

▷ `InverseOp(x)` (method)

Returns: the inverse element of an element of a group ring

This method improves a standard GAP functionality for modular group algebras. It calculates the inverse of a group algebra element. The user can also invoke this function by typing `x^-1` .

Example

```
gap> y;
(Z(2)^0)*f1+(Z(2)^0)*f2+(Z(2)^0)*f1*f2
gap> y^-1;
(Z(2)^0)*f1+(Z(2)^0)*f2+(Z(2)^0)*f3+(Z(2)^0)*f4+(Z(2)^0)*f1*f2+(Z(2)^
0)*f1*f3+(Z(2)^0)*f1*f4+(Z(2)^0)*f2*f4+(Z(2)^0)*f1*f2*f4+(Z(2)^0)*f2*f3*f4+(
Z(2)^0)*f1*f2*f3*f4
gap> y * y^-1;
(Z(2)^0)*<identity> of ...
```

4.2.12 BicyclicUnitOfType1

▷ `BicyclicUnitOfType1(`*`KG, `*`a, g`*`)` (operation)

▷ `BicyclicUnitOfType2(`*`KG, `*`a, g`*`)` (operation)

Returns: an element of a group ring

let a be an element of order n of a group G. We put $\alpha = 1+a+a^2+...+a^{n-1}$. Then $(a-1)*g*\alpha$ and $\alpha*g*(a-1)$ are nilpotent of index two for any element g of the group G not containing in the normalizer $N_G(\langle a\rangle)$, and the units $u_{a,g} = 1+(a-1)*g*\alpha$ and $v_{a,g} = 1+\alpha*g*(a-1)$ are called *bicyclic units* of the 1st and 2nd type respectively. Note that $u_{a,g}$ and $v_{a,g}$ may coincide for some a and g, but in general this does not hold. In the three-argument version these methods construct bicyclic units of both types when `a` and g are elements of the underlying group G of a group ring *KG*. The two-argument version accepts images of elements `a` and g from the underlying group in the group ring KG obtained using the mapping `Embedding( G, KG )`. Note that it is not actually checked that g is not contained in $N_G(\langle a\rangle)$, because this is verified in `BicyclicUnitGroup` (4.4.13).

Example

```
gap> G := SmallGroup(32,6);
<pc group of size 32 with 5 generators>
gap> KG := GroupRing( GF(2), G );
<algebra-with-one over GF(2), with 5 generators>
gap> g := MinimalGeneratingSet( G );
[ f1, f2 ]
gap> g[1] in Normalizer( G, Subgroup( G, [g[2]] ) );
false
gap> g[2] in Normalizer( G, Subgroup( G, [g[1]] ) );
false
```

```
gap> g := List( g, x -> x^Embedding( G, KG ) );
[ (Z(2)^0)*f1, (Z(2)^0)*f2 ]
gap> BicyclicUnitOfType1(g[1],g[2]) = BicyclicUnitOfType2(g[1],g[2]);
false
```

4.2.13 BassCyclicUnit

▷ BassCyclicUnit(*ZG,]g, k*) (operation)

Returns: an element of a group ring

Let g be an element of order n of the group G, and $1 < k < n$ be such that k and n are coprime, then k^Phi(n) is congruent to 1 modulo n. The unit

$$b(g,k) = \left(\sum_{j=0}^{k-1} g^j\right)^{\varphi(n)} + \frac{1-k^{\varphi(n)}}{n}\hat{g},$$

where $\hat{g} = g + g^2 + \ldots + g^n$, is called a *Bass cyclic unit* of the integral group ring *ZG*.

The three-argument version constructs the Bass cyclic unit $b(g,k)$ for the element g from the underlying group G of the group ring *ZG*. The two-argument version accepts the image of g in the group ring ZG obtained using the mapping Embedding(G, KG).

Remark that when G is a finite nilpotent group, the group generated by the Bass cyclic units contain a subgroup of finite index in the centre of the unit group of *ZG* [JPS96].

Example

```
gap> S := SymmetricGroup( 5 );;
gap> ZS := GroupRing( Integers, S );;
gap> f := Embedding( S, ZS );;
gap> BassCyclicUnit( ZS, (1,3,2,5,4) , 3 );
(1)*()+(-2)*(1,2,4,3,5)+(-2)*(1,3,2,5,4)+(3)*(1,4,5,2,3)+(1)*(1,5,3,4,2)
gap> BassCyclicUnit( (1,3,2,5,4)^f, 3 );
(1)*()+(-2)*(1,2,4,3,5)+(-2)*(1,3,2,5,4)+(3)*(1,4,5,2,3)+(1)*(1,5,3,4,2)
```

4.3 Important attributes of group algebras

4.3.1 AugmentationHomomorphism

▷ AugmentationHomomorphism(*KG*) (attribute)

Returns: a homomorphism from a group ring to the underlying ring

The mapping which maps an element of a group ring KG to its augmentation is a homomorphism from KG onto the ring K; see Augmentation (4.2.5). This attribute stores this homomorphism for the group ring *KG*.

Please note that for calculation of the augmentation of an element of a group ring the user is strongly recommended to use Augmentation (4.2.5) which works much faster than AugmentationHomomorphism.

Example

```
gap> F := GF( 2 ); G := SymmetricGroup( 3 ); FG := GroupRing( F, G );
```

```
GF(2)
Sym( [ 1 .. 3 ] )
<algebra-with-one over GF(2), with 2 generators>
gap> e := Embedding( G,FG );
<mapping: SymmetricGroup( [ 1 .. 3 ] ) -> AlgebraWithOne( GF(2), ... ) >
gap> x := (1,2)^e; y := (1,3)^e;
(Z(2)^0)*(1,2)
(Z(2)^0)*(1,3)
gap> a := AugmentationHomomorphism( FG );
[ (Z(2)^0)*(1,2,3), (Z(2)^0)*(1,2) ] -> [ Z(2)^0, Z(2)^0 ]
gap> x^a; y^a; ( x + y )^a; # this is slower
Z(2)^0
Z(2)^0
0*Z(2)
gap> Augmentation(x); Augmentation(y); Augmentation( x + y ); # this is faster
Z(2)^0
Z(2)^0
0*Z(2)
```

4.3.2 AugmentationIdeal

▷ `AugmentationIdeal(`*KG*`)` (attribute)

Returns: an ideal of a group ring

If KG is a group ring, then its augmentation ideal A is generated by all elements of the form $g-1$, where $g \in G \setminus \{ 1 \}$. The augmentation ideal consists of all elements of FG with augmentation 0; see `Augmentation` (4.2.5). This method changes a standard GAP functionality for modular group algebras and returns the augmentation ideal of a modular group algebra *KG*.

Example

```
gap> KG := GroupRing( GF( 2 ), DihedralGroup( 16 ) );
<algebra-with-one over GF(2), with 4 generators>
gap> AugmentationIdeal( KG );
<two-sided ideal in <algebra-with-one over GF(2), with 4 generators>,
  (dimension 15)>
```

4.3.3 RadicalOfAlgebra

▷ `RadicalOfAlgebra(`*KG*`)` (attribute)

Returns: an ideal of a group algebra

This method improves a standard GAP functionality for modular group algebras of finite p-groups. Since in this case the radical of the group algebra coincides with its augmentation ideal, this method simply checks if the algebra *KG* is a p-modular group algebra, and, if yes, it returns the augmentation ideal; otherwise, the standard GAP method will be used.

Example

```
gap> KG := GroupRing( GF( 2 ), DihedralGroup( 16 ) );
<algebra-with-one over GF(2), with 4 generators>
gap> RadicalOfAlgebra( KG );
```

```
<two-sided ideal in <algebra-with-one over GF(2), with 4 generators>,
  (dimension 15)>
gap> RadicalOfAlgebra( KG ) = AugmentationIdeal( KG );
true
```

4.3.4 WeightedBasis

▷ `WeightedBasis(KG)` (attribute)

Returns: a record of two components: weighted basis elements and their weights

The argument *KG* must be a p-modular group algebra.

For a group algebra KG, let A denote the augmentation ideal, and assume that c is the smallest number such that $A^c = 0$. Then a weighted basis of KG is some basis $b_1, \ldots, b_n$ for the augmentation ideal A, for which there are indices $i_1 = 1, \ldots, i_{c-1}$ such that $b_{i_k}, \ldots, b_n$ is a basis for A^k. The weight of an element b_i of a weighted basis is the unique integer w such that b_i belongs to w-th power of A but does not belong to its $(w+1)$-th power.

Note that this function actually constructs a basis for the *augmentation ideal* of *KG* and not for *KG* itself. Since the augmentation ideal has co-dimension 1 in KG, a basis for KG can be easily obtained by adjoining the identity element of the group.

The method returns a record whose basis entry is the basis and the weights entry is a list of the corresponding weights the of basis elements. See Section 3.3 for more details.

Example

```
gap> KG := GroupRing( GF( 2 ), ElementaryAbelianGroup( 4 ) );
<algebra-with-one over GF(2), with 2 generators>
gap> WeightedBasis( KG );
rec(
  weightedBasis := [ (Z(2)^0)*<identity> of ...+(Z(2)^0)*f2,
      (Z(2)^0)*<identity> of ...+(Z(2)^0)*f1,
      (Z(2)^0)*<identity> of ...+(Z(2)^0)*f1+(Z(2)^0)*f2+(Z(2)^0)*f1*f2 ],
  weights := [ 1, 1, 2 ] )
```

4.3.5 AugmentationIdealPowerSeries

▷ `AugmentationIdealPowerSeries(KG)` (attribute)

Returns: a list of ideals of a group algebra

The argument *KG* is a p-modular group algebra. The method returns a list whose elements are the terms of the augmentation ideal filtration of *KG*, that is `AugmentationIdealPowerSeries(A)[i]` is the i-th power of the augmentation ideal of *KG*.

Example

```
gap> KG := GroupRing( GF( 2 ), DihedralGroup( 16 ) );
<algebra-with-one over GF(2), with 4 generators>
gap> AugmentationIdealPowerSeries( KG );
[ <algebra of dimension 15 over GF(2)>, <algebra of dimension 13 over GF(2)>,
  <algebra of dimension 11 over GF(2)>, <algebra of dimension 9 over GF(2)>,
  <algebra of dimension 7 over GF(2)>, <algebra of dimension 5 over GF(2)>,
  <algebra of dimension 3 over GF(2)>, <algebra of dimension 1 over GF(2)>,
```

```
  <algebra over GF(2)> ]
gap> Length(last);
9
```

4.3.6 AugmentationIdealNilpotencyIndex

▷ AugmentationIdealNilpotencyIndex(*KG*) (attribute)

For the p-modular group algebra *KG* the method returns the smallest number n such that $A^n = 0$, where A is the augmentation ideal of *KG*. This can be done using Jenning's theory without the explicit calculations of the powers of the augmentation ideal.

Example

```
gap> KG := GroupRing( GF( 2 ), DihedralGroup( 16 ) );
<algebra-with-one over GF(2), with 4 generators>
gap> AugmentationIdealNilpotencyIndex( KG );
9
```

4.3.7 AugmentationIdealOfDerivedSubgroupNilpotencyIndex

▷ AugmentationIdealOfDerivedSubgroupNilpotencyIndex(*KG*) (attribute)

For the p-modular group algebra *KG* this attribute stores the nilpotency index of the augmentation ideal of KG' where G' denotes the derived subgroup of G.

Example

```
gap> KG := GroupRing( GF( 2 ), DihedralGroup( 16 ) );
<algebra-with-one over GF(2), with 4 generators>
gap> AugmentationIdealOfDerivedSubgroupNilpotencyIndex( KG );
4
gap> D := DerivedSubgroup( UnderlyingGroup( KG ) );
Group([ f3, f4 ])
gap> KD := GroupRing( GF( 2 ), D );
<algebra-with-one over GF(2), with 2 generators>
gap> AugmentationIdealNilpotencyIndex( KD );
4
```

4.3.8 LeftIdealBySubgroup

▷ LeftIdealBySubgroup(*KG*, *H*) (operation)
▷ RightIdealBySubgroup(*KG*, *H*) (operation)
▷ TwoSidedIdalBySubgroup(*KG*, *H*) (operation)

Returns: an ideal of a group ring

Let *KG* be a group ring of a group G over the ring K, and *H* be a subgroup of G. Then the set $J_l(H)$ of all elements of *KG* of the form

$$\sum_{h \in H} x_h(h-1)$$

is the left ideal in *KG* generated by all elements $h-1$ with h in H. The right ideal $J_r(H)$ is defined analogously. These operations are used to consrtuct such ideals, taking into account the fact, that the ideal $J_l(H)$ is two-sided if and only if *H* is normal in G. An attempt of constructing two-sided ideal for a non-normal subgroup *H* will lead to an error message.

Example

```
gap> KG := GroupRing( GF(2), DihedralGroup(16) );
<algebra-with-one over GF(2), with 4 generators>
gap> G := DihedralGroup(16);
<pc group of size 16 with 4 generators>
gap> KG := GroupRing( GF(2), G );
<algebra-with-one over GF(2), with 4 generators>
gap> D := DerivedSubgroup( G );
Group([ f3, f4 ])
gap> LeftIdealBySubgroup( KG, D );
<two-sided ideal in <algebra-with-one over GF(2), with 4 generators>,
  (dimension 12)>
gap> H := Subgroup( G, [ GeneratorsOfGroup(G)[1] ]);
Group([ f1 ])
gap> IsNormal( G, H );
false
gap> LeftIdealBySubgroup( KG, H );
<left ideal in <algebra-with-one over GF(2), with 4 generators>, (dimension 8
 )>
```

4.4 Computations with the unit group

4.4.1 NormalizedUnitGroup

▷ NormalizedUnitGroup(*KG*) (attribute)

Returns: a group generated by group algebra elements

Determines the normalized unit group of a p-modular group algebra *KG* over the field of p elements. Returns the normalized unit group as the group generated by certain elements of *KG*; see Section 3.3 for more details.

For efficient computations the user is recommended to use PcNormalizedUnitGroup (4.4.2).

Example

```
gap> KG := GroupRing( GF( 2 ), DihedralGroup( 16 ) );
<algebra-with-one over GF(2), with 4 generators>
gap> V := NormalizedUnitGroup( KG );
<group of size 32768 with 15 generators>
gap> u := GeneratorsOfGroup( V )[4];
(Z(2)^0)*f1+(Z(2)^0)*f2+(Z(2)^0)*f1*f2
```

4.4.2 PcNormalizedUnitGroup

▷ PcNormalizedUnitGroup(*KG*) (attribute)

Returns: a group given by power-commutator presentation

The argument *KG* is a *p*-modular group algebra over the field of *p* elements. PcNormalizedUnitGroup returns the normalized unit group of *KG* given by a power-commutator presentation. The generators in this polycyclic presentation correspond to the weighted basis elements of *KG*. For more details, see Section 3.3.

Example

```
gap> W := PcNormalizedUnitGroup( KG );
<pc group of size 32768 with 15 generators>
gap> w := GeneratorsOfGroup( W )[4];
f4
```

4.4.3 NaturalBijectionToPcNormalizedUnitGroup

▷ NaturalBijectionToPcNormalizedUnitGroup(*KG*) (attribute)

Returns: a homomorphism of groups

The normalised unit group of a *p*-modular group algebra *KG* over the field of *p* elements can be computed using two methods, namely NormalizedUnitGroup (4.4.1) and PcNormalizedUnitGroup (4.4.2). These two methods return two different objects, and they can be used for different types of computations. The elements of NormalizedUnitGroup(KG) are represented in their natural group algebra representation, and hence they can easily be identified in the group algebra. However, the more quickly constructed NormalizedUnitGroup(KG) is often not suitable for further fast calculations. Hence one will have to use PcNormalizedUnitGroup(KG) if one wants to find some group theoretic properties of the normalized unit group. This method returns the bijection from NormalizedUnitGroup(*KG*) onto PcNormalizedUnitGroup(*KG*). This bijection can be used to map the result of a computation in PcNormalizedUnitGroup(*KG*) into NormalizedUnitGroup(*KG*).

Example

```
gap> f := NaturalBijectionToPcNormalizedUnitGroup( KG );
MappingByFunction( <group of size 32768 with 15 generators>, <pc group of size\
 32768 with 15 generators>, function( x ) ... end )
gap> u := GeneratorsOfGroup( V )[4];;
gap> u^f;
f4
gap> GeneratorsOfGroup( V )[4]^f = GeneratorsOfGroup( W )[4];
true
```

4.4.4 NaturalBijectionToNormalizedUnitGroup

▷ NaturalBijectionToNormalizedUnitGroup(*KG*) (attribute)

Returns: a homomorphism of groups

For a *p*-modular group algebra *KG* over the field of *p* elements this function returns the inverse of the mapping NaturalBijectionToPcNormalizedUnitGroup (4.4.3)

Example

```
gap> t := NaturalBijectionToNormalizedUnitGroup(KG);;
```

```
gap> w := GeneratorsOfGroup(W)[4];;
gap> w^t;
(Z(2)^0)*f1+(Z(2)^0)*f2+(Z(2)^0)*f1*f2
gap> GeneratorsOfGroup( W )[4]^t = GeneratorsOfGroup( V )[4];
true
```

4.4.5 Embedding (from group to unit group)

▷ `Embedding(`*H*`, `*V*`)` (operation)

Returns: a homomorphism from an underlying group to a normalized unit group in pc-presentation

Let *H* be a subgroup of a group *G* and *V* be the normalized unit group of the group algebra *KG* given by the power-commutator presentation (see `PcNormalizedUnitGroup` (4.4.2). Then `Embedding( H, V )` returns the homomorphism from *H* to *V*, which is the composition of `Embedding( H, KG )` and `NaturalBijectionToPcNormalizedUnitGroup( KG )`.

Example

```
gap> G := DihedralGroup( 16 );
<pc group of size 16 with 4 generators>
gap> KG := GroupRing( GF( 2 ), G );
<algebra-with-one over GF(2), with 4 generators>
gap> V:=PcNormalizedUnitGroup( KG );
<pc group of size 32768 with 15 generators>
gap> ucs := UpperCentralSeries( V );
[ <pc group of size 32768 with 15 generators>,
  <pc group of size 4096 with 12 generators>,
  Group([ f3*f5*f13*f15, f7, f15, f13*f15, f14*f15, f11*f13*f14*f15, f12,
      f9*f12, f10 ]),
  Group([ f3*f5*f13*f15, f7, f15, f13*f15, f14*f15, f11*f13*f14*f15 ]),
  Group([  ]) ]
gap> f := Embedding( G, V );
[ f1, f2, f3, f4 ] -> [ f2, f1, f3, f7 ]
gap> G1 := Image( f, G );
Group([ f2, f1, f3, f7 ])
gap> H := Intersection( ucs[2], G1 ); # compute intersection in V(KG)
Group([ f3, f7, f3*f7 ])
gap> T:=PreImage( f, H );             # find its preimage in G
Group([ f3, f4, f3*f4 ])
gap> IdGroup( T );
[ 4, 1 ]
```

4.4.6 Units

▷ `Units(`*KG*`)` (attribute)

Returns: the unit group of a group ring

This improves a standard GAP functionality for modular group algebras of finite *p*-groups over the field of *p* elements. It returns the unit group of *KG* as a direct product of `Units(K)` and

NormalizedUnitGroup(KG), where the latter is generated by certain elements of *KG*; see Chapter 3 for more details.

Example

```
gap> U := Units( KG );
#I  LAGUNA package: Computing the unit group ...
<group of size 32768 with 15 generators>
gap> GeneratorsOfGroup( U )[5]; # now elements of U are already in KG
(Z(2)^0)*f2+(Z(2)^0)*f3+(Z(2)^0)*f2*f3
gap> FH := GroupRing( GF(3), SmallGroup(27,3) );
<algebra-with-one over GF(3), with 3 generators>
gap> T := Units( FH );
#I  LAGUNA package: Computing the unit group ...
<group of size 5083731656658 with 27 generators>
gap> x := GeneratorsOfGroup( T )[1];
DirectProductElement( [ Z(3), (Z(3)^0)*<identity> of ... ] )
gap> x in FH;
false
gap> x[1] * x[2] in FH; # how to get the corresponding element of FH
true
```

4.4.7 PcUnits

▷ PcUnits(*KG*) (attribute)

Returns: a group given by power-commutator presentation

Returns the unit group of *KG* as a direct product of Units(K) and PcNormalizedUnitGroup(KG), where the latter is a group given by a polycyclic presentation. See Section 3.4 for more details.

Example

```
gap> W := PcUnits( KG );
<pc group of size 32768 with 15 generators>
gap> GeneratorsOfGroup( W )[5];
f5
gap> FH := GroupRing( GF(3), SmallGroup(27,3) );
<algebra-with-one over GF(3), with 3 generators>
gap> T := PcUnits(FH);
<group of size 5083731656658 with 27 generators>
gap> x := GeneratorsOfGroup( T )[2];
DirectProductElement( [ Z(3)^0, f1 ] )
```

4.4.8 IsGroupOfUnitsOfMagmaRing

▷ IsGroupOfUnitsOfMagmaRing(*U*) (property)

This property will be automatically set true, if *U* is a group generated by some units of a magma ring, including Units(KG) and NormalizedUnitgroup(KG). Otherwise this property will not be bound.

Example

```
gap> IsGroupOfUnitsOfMagmaRing( NormalizedUnitGroup( KG ) );
true
gap> IsGroupOfUnitsOfMagmaRing( Units( KG ) );
true
```

4.4.9 IsUnitGroupOfGroupRing

▷ `IsUnitGroupOfGroupRing(`*U*`)` (property)

This property will be automatically set `true`, if *U* is the unit group of a *p*-modular group algebra, obtained either by `Units(KG)` or by `PcUnits(KG)`. Otherwise this property will not be bound.

Example

```
gap> IsUnitGroupOfGroupRing( Units( KG ) );
true
gap> IsUnitGroupOfGroupRing( PcUnits( KG ) );
true
```

4.4.10 IsNormalizedUnitGroupOfGroupRing

▷ `IsNormalizedUnitGroupOfGroupRing(`*U*`)` (property)

This property will be automatically set `true`, if *U* is the normalized unit group of a *p*-modular group algebra, obtained either by `NormalizedUnitGroup(KG)` or by `PcNormalizedUnitGroup(KG)`. Otherwise this property will not be bound.

Example

```
gap> IsNormalizedUnitGroupOfGroupRing( NormalizedUnitGroup( KG ) );
true
gap> IsNormalizedUnitGroupOfGroupRing( PcNormalizedUnitGroup( KG ) );
true
```

4.4.11 UnderlyingGroupRing

▷ `UnderlyingGroupRing(`*U*`)` (attribute)

Returns: a group ring

If *U* is the (normalized) unit group of a *p*-modular group algebra *KG* obtained using one of the functions `Units(KG)`, `PcUnits(KG)`, `NormalizedUnitGroup(KG)` or `PcNormalizedUnitGroup(KG)`, then the attribute `UnderlyingGroupRing` stores *KG*.

Example

```
gap> UnderlyingGroupRing( Units( KG ) );
<algebra-with-one of dimension 16 over GF(2)>
gap> UnderlyingGroupRing( PcUnits( KG ) );
```

```
<algebra-with-one of dimension 16 over GF(2)>
gap> UnderlyingGroupRing( NormalizedUnitGroup( KG ) );
<algebra-with-one of dimension 16 over GF(2)>
gap> UnderlyingGroupRing( PcNormalizedUnitGroup( KG ) );
<algebra-with-one of dimension 16 over GF(2)>
```

4.4.12 UnitarySubgroup

▷ `UnitarySubgroup(`*U*`)` (attribute)

Returns: the subgroup of the unit group

Let *U* be the normalized unit group of a group ring in either natural (see `NormalizedUnitGroup` (4.4.1)) or power-commutator (see `PcNormalizedUnitGroup` (4.4.2)) presentation. The attribute stores the unitary subgroup of *U*, generated by all unitary units of *U* (see `IsUnitary` (4.2.9)). The method is straightforward, so it is not recommended to run it for large groups.

Example

```
gap> KG := GroupRing( GF( 2 ), DihedralGroup( 8 ) );
<algebra-with-one over GF(2), with 3 generators>
gap> U := NormalizedUnitGroup( KG );
<group of size 128 with 7 generators>
gap> HU := UnitarySubgroup( U );
<group with 5 generators>
gap> IdGroup( HU );
[ 64, 261 ]
gap> V := PcNormalizedUnitGroup( KG );
<pc group of size 128 with 7 generators>
gap> HV := UnitarySubgroup( V );
Group([ f1, f2, f5, f6, f7 ])
gap> IdGroup( HV );
[ 64, 261 ]
gap> Image(NaturalBijectionToPcNormalizedUnitGroup( KG ), HU ) = HV;
true
```

4.4.13 BicyclicUnitGroup

▷ `BicyclicUnitGroup(`*U*`)` (attribute)

Returns: the subgroup of the unit group, generated by bicyclic units

Let *U* be the normalized unit group of a group ring in either natural (see `NormalizedUnitGroup` (4.4.1)) or power-commutator (see `PcNormalizedUnitGroup` (4.4.2)) presentation. The attribute stores the subgroup of *U*, generated by all bicyclic units $u_{g,h}$ and $v_{g,h}$ (see `BicyclicUnitOfType1` (4.2.12) and `BicyclicUnitOfType2` (4.2.12)), where g and h run over the elements of the underlying group, and h do not belongs to the normalizer of $\langle g \rangle$ in G.

Example

```
gap> KG := GroupRing( GF( 2 ), DihedralGroup( 8 ) );
<algebra-with-one over GF(2), with 3 generators>
gap> U := NormalizedUnitGroup( KG );
```

```
<group of size 128 with 7 generators>
gap> BU := BicyclicUnitGroup( U );
<group with 2 generators>
gap> IdGroup( BU );
[ 4, 2 ]
gap> V := PcNormalizedUnitGroup( KG );
<pc group of size 128 with 7 generators>
gap> BV := BicyclicUnitGroup( V );
Group([ f5*f6, f6*f7 ])
gap> IdGroup( BV );
[ 4, 2 ]
gap> Image( NaturalBijectionToPcNormalizedUnitGroup( KG ), BU ) = BV;
true
```

4.4.14 GroupBases

▷ `GroupBases(`*KG*`)` (attribute)

Returns: a list of lists of group rings elements

The subgroup B of the normalized unit group of the group algebra KG is called a *group basis*, if the elements of B are linearly independent over the field K and $KB = KG$. If *KG* is a p-modular group algebra, then `GroupBases` returns a list of representatives of the conjugacy classes of the group bases of the group algebra *KG* in its normalised unit group.

Example

```
gap> D8 := DihedralGroup( 8 );
<pc group of size 8 with 3 generators>
gap> K := GF(2);
GF(2)
gap> KD8 := GroupRing( GF( 2 ), D8 );
<algebra-with-one over GF(2), with 3 generators>
gap> gb := GroupBases( KD8 );;
gap> Length( gb );
32
gap> gb[1];
[ (Z(2)^0)*<identity> of ..., (Z(2)^0)*f3,
  (Z(2)^0)*f1*f2+(Z(2)^0)*f2*f3+(Z(2)^0)*f1*f2*f3,
  (Z(2)^0)*f2+(Z(2)^0)*f1*f2+(Z(2)^0)*f1*f2*f3,
  (Z(2)^0)*<identity> of ...+(Z(2)^0)*f2+(Z(2)^0)*f3+(Z(2)^0)*f2*f3+(Z(2)^
    0)*f1*f2*f3, (Z(2)^0)*f2+(Z(2)^0)*f1*f3+(Z(2)^0)*f2*f3,
  (Z(2)^0)*<identity> of ...+(Z(2)^0)*f2+(Z(2)^0)*f3+(Z(2)^0)*f1*f2+(Z(2)^
    0)*f2*f3, (Z(2)^0)*f1+(Z(2)^0)*f2+(Z(2)^0)*f2*f3 ]
gap> Length( last );
8
```

4.5 The Lie algebra of a group algebra

4.5.1 LieAlgebraByDomain

▷ `LieAlgebraByDomain(A)` (method)

This method takes a group algebra as its argument, and constructs its associated Lie algebra in which the product is the bracket operation: $[a,b] = ab - ba$. It is recommended that the user never calls this method. The Lie algebra for an associative algebra should normally be created using `LieAlgebra( A )`. When `LieAlgebra` is first invoked, it constructs the Lie algebra for A using `LieAlgebraByDomain`. After that it stores this Lie algebra and simply returns it if `LieAlgebra` is called again.

Example

```
gap> G := SymmetricGroup(3);; FG := GroupRing( GF( 2 ), G );
<algebra-with-one over GF(2), with 2 generators>
gap> L := LieAlgebra( FG );
#I  LAGUNA package: Constructing Lie algebra ...
<Lie algebra over GF(2)>
```

4.5.2 IsLieAlgebraByAssociativeAlgebra

▷ `IsLieAlgebraByAssociativeAlgebra(L)` (Category)

This category signifies that the Lie algebra L was constructed as the Lie algebra associated with an associative algebra (this piece of information cannot be obtained later).

Example

```
gap> KG := GroupRing( GF(3), DihedralGroup(16) );
<algebra-with-one over GF(3), with 4 generators>
gap> L := LieAlgebra ( KG );
#I  LAGUNA package: Constructing Lie algebra ...
<Lie algebra over GF(3)>
gap> IsLieAlgebraByAssociativeAlgebra( L );
true
```

4.5.3 UnderlyingAssociativeAlgebra

▷ `UnderlyingAssociativeAlgebra(L)` (attribute)

Returns: the underlying associative algebra of a Lie algebra

If a Lie algebra L is constructed from an associative algebra, then it remembers this underlying associative algebra as one of its attributes.

Example

```
gap> KG := GroupRing( GF(2), DihedralGroup(16) );
<algebra-with-one over GF(2), with 4 generators>
gap> L := LieAlgebra ( KG );
```

```
#I  LAGUNA package: Constructing Lie algebra ...
<Lie algebra over GF(2)>
gap> UnderlyingAssociativeAlgebra( L );
<algebra-with-one over GF(2), with 4 generators>
gap> last = KG;
true
```

4.5.4 NaturalBijectionToLieAlgebra

▷ NaturalBijectionToLieAlgebra(*A*) (attribute)

Returns: a mapping

The natural linear bijection between the (isomorphic, but not equal) underlying vector spaces of an associative algebra *A* and its associated Lie algebra is stored as an attribute of *A*. Note that this is a vector space isomorphism between two algebras, but not an algebra isomorphism.

Example

```
gap> F := GF( 2 ); G := SymmetricGroup( 3 ); FG := GroupRing( F, G );
GF(2)
Sym( [ 1 .. 3 ] )
<algebra-with-one over GF(2), with 2 generators>
gap> t := NaturalBijectionToLieAlgebra( FG );;
#I  LAGUNA package: Constructing Lie algebra ...
gap> a := Random( FG );
(Z(2)^0)*(1,2,3)+(Z(2)^0)*(1,3,2)+(Z(2)^0)*(1,3)
gap> a * a;                       # product in the associative algebra
(Z(2)^0)*()+(Z(2)^0)*(1,2,3)+(Z(2)^0)*(1,3,2)
gap> b := a^t;
LieObject( (Z(2)^0)*(1,2,3)+(Z(2)^0)*(1,3,2)+(Z(2)^0)*(1,3) )
gap> b * b; # product in the Lie algebra (commutator) - must be zero!
LieObject( <zero> of ... )
```

4.5.5 NaturalBijectionToAssociativeAlgebra

▷ NaturalBijectionToAssociativeAlgebra(*L*) (attribute)

This is the inverse of the previous linear bijection, stored as an attribute of the Lie algebra *L*.

Example

```
gap> G := SymmetricGroup(3); FG := GroupRing( GF( 2 ), G );
Sym( [ 1 .. 3 ] )
<algebra-with-one over GF(2), with 2 generators>
gap> L := LieAlgebra( FG );
#I  LAGUNA package: Constructing Lie algebra ...
<Lie algebra over GF(2)>
gap> s := NaturalBijectionToAssociativeAlgebra( L );;
gap> InverseGeneralMapping( s ) = NaturalBijectionToLieAlgebra( FG );
true
```

4.5.6 IsLieAlgebraOfGroupRing

▷ `IsLieAlgebraOfGroupRing(`*L*`)` (property)

If a Lie algebra *L* is constructed from an associative algebra which happens to be in fact a group ring, it has many nice properties that can be used for fast algorithms, so this information is stored as a property.

Example

```
gap> F := GF( 2 ); G := SymmetricGroup( 3 ); FG := GroupRing( F, G );
GF(2)
Sym( [ 1 .. 3 ] )
<algebra-with-one over GF(2), with 2 generators>
gap> L := LieAlgebra( FG );
#I  LAGUNA package: Constructing Lie algebra ...
<Lie algebra over GF(2)>
gap> IsLieAlgebraOfGroupRing( L );
true
```

4.5.7 UnderlyingGroup (of Lie algebra of a group ring)

▷ `UnderlyingGroup(`*L*`)` (attribute)

Returns: the underlying group

The underlying group of a Lie algebra *L* that is constructed from a group ring is defined as the underlying group of this group ring; see `UnderlyingGroup` (4.1.4).

Example

```
gap> F := GF( 2 ); G := SymmetricGroup( 3 ); FG := GroupRing( F, G );
GF(2)
Sym( [ 1 .. 3 ] )
<algebra-with-one over GF(2), with 2 generators>
gap> L := LieAlgebra( FG );
#I  LAGUNA package: Constructing Lie algebra ...
<Lie algebra over GF(2)>
gap> UnderlyingGroup( L );
Sym( [ 1 .. 3 ] )
gap> LeftActingDomain( L );
GF(2)
```

4.5.8 Embedding (from group to Lie algebra)

▷ `Embedding(`*U*`, `*L*`)` (operation)

Returns: a mapping, which is a composition of two mappings

Let *FG* be a group ring, let *U* be a submagma of *G*, and let *L* be the Lie algebra associated with *FG*. Then `Embedding(`*U*`, `*L*` )` returns the obvious mapping from *U* to *L* (as the composition of the mappings `Embedding( `*U*`, FG )` and `NaturalBijectionToLieAlgebra( FG )`).

Example

```
gap> F := GF( 2 ); G := SymmetricGroup( 3 ); FG := GroupRing( F, G );
GF(2)
Sym( [ 1 .. 3 ] )
<algebra-with-one over GF(2), with 2 generators>
gap> L := LieAlgebra( FG );
#I  LAGUNA package: Constructing Lie algebra ...
<Lie algebra over GF(2)>
gap> f := Embedding( G, L );;
gap> (1,2)^f + (1,3)^f;
LieObject( (Z(2)^0)*(1,2)+(Z(2)^0)*(1,3) )
```

4.5.9 LieCentre

▷ `LieCentre(`*L*`)` (method)

Returns: a Lie algebra

The centre of the Lie algebra associated with a group ring corresponds to the centre of the underlying group ring, and it can be calculated very fast by considering the conjugacy classes of the group. This method returns the centre of *L* using this idea.

Example

```
gap> G := SmallGroup( 256, 400 ); FG := GroupRing( GF( 2 ), G );
<pc group of size 256 with 8 generators>
<algebra-with-one over GF(2), with 8 generators>
gap> L := LieAlgebra( FG );
#I  LAGUNA package: Constructing Lie algebra ...
<Lie algebra over GF(2)>
gap> C := LieCentre( L );
<Lie algebra of dimension 28 over GF(2)>
gap> D := LieDerivedSubalgebra( L );
#I  LAGUNA package: Computing the Lie derived subalgebra ...
<Lie algebra of dimension 228 over GF(2)>
gap> c := Dimension( C ); d := Dimension( D ); l := Dimension( L );
28
228
256
gap> c + d = l; # This is always the case for Lie algebras of group algebras!
true
```

4.5.10 LieDerivedSubalgebra

▷ `LieDerivedSubalgebra(`*L*`)` (method)

Returns: a Lie algebra

If *L* is the Lie algebra associated with a group ring, then this method returns the Lie derived subalgebra of *L*. This can be done very fast using the conjugacy classes of the underlying group.

Example

```
gap> G := SmallGroup( 256, 400 ); FG := GroupRing( GF( 2 ), G );
```

```
<pc group of size 256 with 8 generators>
<algebra-with-one over GF(2), with 8 generators>
gap> L := LieAlgebra( FG );
#I  LAGUNA package: Constructing Lie algebra ...
<Lie algebra over GF(2)>
gap> C := LieCentre( L );
<Lie algebra of dimension 28 over GF(2)>
gap> D := LieDerivedSubalgebra( L );
#I  LAGUNA package: Computing the Lie derived subalgebra ...
<Lie algebra of dimension 228 over GF(2)>
gap> l := Dimension( L ); c := Dimension( C ); d := Dimension( D );
256
28
228
gap> c + d = l; # This is always the case for Lie algebras of group algebras!
true
```

4.5.11 IsLieAbelian

▷ `IsLieAbelian(`*L*`)` (method)

The Lie algebra L of an associative algebra A is Lie abelian, if and only if A is abelian, so this method refers to `IsAbelian( A )`.

Example

```
gap> G := SymmetricGroup( 3 ); FG := GroupRing( GF( 2 ), G);
Sym( [ 1 .. 3 ] )
<algebra-with-one over GF(2), with 2 generators>
gap> L := LieAlgebra( FG );
#I  LAGUNA package: Constructing Lie algebra ...
<Lie algebra over GF(2)>
gap> IsAbelian( G );
false
gap> IsAbelian( L );    # This command should not be used for Lie algebras!
true
gap> IsLieAbelian( L ); # Instead, IsLieAbelian is the correct command.
false
```

4.5.12 IsLieSolvable

▷ `IsLieSolvable(`*L*`)` (method)

In [PPS73] Passi, Passman, and Sehgal have classified all groups G such that the Lie algebra associated with the group ring is solvable. This method uses their classification, making it considerably faster than the more elementary method which just calculates Lie commutators.

Example

```
gap> G := SmallGroup( 256, 400 ); FG := GroupRing( GF( 2 ), G );
```

```
<pc group of size 256 with 8 generators>
<algebra-with-one over GF(2), with 8 generators>
gap> L := LieAlgebra( FG );
#I  LAGUNA package: Constructing Lie algebra ...
<Lie algebra over GF(2)>
gap> IsLieSolvable( L );                        # This is very fast.
#I  LAGUNA package: Checking Lie solvability ...
true
gap> List( LieDerivedSeries( L ), Dimension ); # This is very slow.
#I  LAGUNA package: Computing the Lie derived subalgebra ...
[ 256, 228, 189, 71, 0 ]
```

4.5.13 IsLieNilpotent

▷ IsLieNilpotent(*L*) (method)

In [PPS73] Passi, Passman, and Sehgal have classified all groups G such that the Lie algebra associated with the group ring is Lie nilpotent. This method uses their classification, making it considerably faster than the more elementary method which just calculates Lie commutators.

Example

```
gap> G := SmallGroup( 256, 400 ); FG := GroupRing( GF( 2 ), G );
<pc group of size 256 with 8 generators>
<algebra-with-one over GF(2), with 8 generators>
gap> L := LieAlgebra( FG );
#I  LAGUNA package: Constructing Lie algebra ...
<Lie algebra over GF(2)>
gap> IsLieNilpotent( L );                          # This is very fast.
#I  LAGUNA package: Checking Lie nilpotency ...
true
gap> List( LieLowerCentralSeries( L ), Dimension ); # This is very slow.
#I  LAGUNA package: Computing the Lie derived subalgebra ...
[ 256, 228, 222, 210, 191, 167, 138, 107, 76, 54, 29, 15, 6, 0 ]
```

4.5.14 IsLieMetabelian

▷ IsLieMetabelian(*L*) (property)

In [LR86] Levin and Rosenberger have classified all groups G such that the Lie algebra associated with the group ring is Lie metabelian. This method uses their classification, making it considerably faster than the more elementary method which just calculates Lie commutators.

Example

```
gap> G := SmallGroup( 256, 400 ); FG := GroupRing( GF( 2 ), G );
<pc group of size 256 with 8 generators>
<algebra-with-one over GF(2), with 8 generators>
gap> L := LieAlgebra( FG );
#I  LAGUNA package: Constructing Lie algebra ...
```

```
<Lie algebra over GF(2)>
gap> IsLieMetabelian( L );
false
```

4.5.15 IsLieCentreByMetabelian

▷ IsLieCentreByMetabelian(*L*) (property)

In [Ros02] the third author of this package classified all groups G such that the Lie algebra associated with the group ring is Lie centre-by-metabelian. This method uses the classification, making it considerably faster than the more elementary method which just calculates Lie commutators.

Example

```
gap> G := SymmetricGroup( 3 ); FG := GroupRing( GF( 2 ), G );
Sym( [ 1 .. 3 ] )
<algebra-with-one over GF(2), with 2 generators>
gap> L := LieAlgebra( FG );
#I  LAGUNA package: Constructing Lie algebra ...
<Lie algebra over GF(2)>
gap> IsLieMetabelian( L );
false
gap> IsLieCentreByMetabelian( L );
true
```

4.5.16 CanonicalBasis

▷ CanonicalBasis(*L*) (method)

Returns: basis of a Lie algebra

The canonical basis of a group algebra FG is formed by the elements of G. Here L is the Lie algebra associated with FG, and the method returns the images of the elements of G in L.

Example

```
gap> G := SymmetricGroup( 3 ); FG := GroupRing( GF( 2 ), G );
Sym( [ 1 .. 3 ] )
<algebra-with-one over GF(2), with 2 generators>
gap> L := LieAlgebra( FG );
#I  LAGUNA package: Constructing Lie algebra ...
<Lie algebra over GF(2)>
gap> B := CanonicalBasis( L );
CanonicalBasis( <Lie algebra of dimension 6 over GF(2)> )
gap> Elements( B );
[ LieObject( (Z(2)^0)*() ), LieObject( (Z(2)^0)*(2,3) ),
  LieObject( (Z(2)^0)*(1,2) ), LieObject( (Z(2)^0)*(1,2,3) ),
  LieObject( (Z(2)^0)*(1,3,2) ), LieObject( (Z(2)^0)*(1,3) ) ]
```

4.5.17 IsBasisOfLieAlgebraOfGroupRing

▷ IsBasisOfLieAlgebraOfGroupRing(*B*) (property)

A basis *B* has this property if the preimages of the basis vectors in the group algebra form a group. It can be verified if a basis has this property. This is important for the speed of the calculation of the structure constants table; see StructureConstantsTable (4.5.18).

Example

```
gap> G := SymmetricGroup( 3 ); FG := GroupRing( GF( 2 ), G );
Sym( [ 1 .. 3 ] )
<algebra-with-one over GF(2), with 2 generators>
gap> L := LieAlgebra( FG );
#I  LAGUNA package: Constructing Lie algebra ...
<Lie algebra over GF(2)>
gap> B := CanonicalBasis( L );
CanonicalBasis( <Lie algebra of dimension 6 over GF(2)> )
gap> IsBasisOfLieAlgebraOfGroupRing( B );
true
```

4.5.18 StructureConstantsTable

▷ StructureConstantsTable(*B*) (method)

A very fast implementation for calculating the structure constants table for the Lie algebra L associated with a group ring with respect to its canonical basis *B* using its special structure; see CanonicalBasis (4.5.16).

Example

```
gap> G := CyclicGroup( 2 ); FG := GroupRing( GF( 2 ), G );
<pc group of size 2 with 1 generators>
<algebra-with-one over GF(2), with 1 generators>
gap> L := LieAlgebra( FG );
#I  LAGUNA package: Constructing Lie algebra ...
<Lie algebra over GF(2)>
gap> B := CanonicalBasis( L );
CanonicalBasis( <Lie algebra of dimension 2 over GF(2)> )
gap> StructureConstantsTable( B );
#I  LAGUNA package: Computing the structure constants table ...
[ [ [ [  ], [  ] ], [ [  ], [  ] ] ], [ [ [  ], [  ] ], [ [  ], [  ] ] ], -1,
  0*Z(2) ]
```

4.5.19 LieUpperNilpotencyIndex

▷ LieUpperNilpotencyIndex(*KG*) (attribute)

In a modular group algebra KG the *upper Lie power series* is defined as follows: $KG^{(1)} = KG$, $KG^{(n+1)}$ is the associative ideal, generated by $[KG^{(n)}, KG]$. The upper Lie nilpotency index $t^L(G)$ of

the group algebra KG is defined to be the smallest number n such that $KG^{(n)} = 0$. It can be calculated very fast using Lie dimension subgroups [Sha91], that is, using only information about the underlying group; see `LieDimensionSubgroups` (4.6.4). This is why it is stored as an attribute of the group algebra *KG* rather than that of its associated Lie algebra.

Example

```
gap> KG := GroupRing( GF( 2 ), DihedralGroup( 16 ) );
<algebra-with-one over GF(2), with 4 generators>
gap> LieUpperNilpotencyIndex( KG );
5
```

4.5.20 LieLowerNilpotencyIndex

▷ `LieLowerNilpotencyIndex(`*KG*`)` (attribute)

In a modular group algebra KG the *lower Lie power series* is defined as follows: $KG^{[n]}$ is the associative ideal, generated by all (left-normed) Lie-products $[x_1, x_2, \ldots, x_n]$, $x_i \in KG$. The lower Lie nilpotency index $t_L(G)$ of the group algebra KG is defined to be the minimal smallest n such that $KG^{[n]} = 0$. In [Du92] the Jennings' conjecture was proved, which means that the nilpotency class of the normalized unit group of the modular group algebra KG is equal to $t_L(G) - 1$.

This allows to express lower Lie nilpotency index via the nilpotency class of the normalized unit group, and with its polycyclic presentation, provided by LAGUNA, this will be faster than elementary calculations with Lie commutators. As the previous attribute, this index is also stored as an attribute of the group algebra *KG*.

Example

```
gap> KG := GroupRing( GF( 2 ), DihedralGroup( 16 ) );
<algebra-with-one over GF(2), with 4 generators>
gap> LieLowerNilpotencyIndex( KG );
5
```

4.5.21 LieDerivedLength

▷ `LieDerivedLength(`*L*`)` (attribute)

Let L be a Lie algebra. The *Lie derived series* of L is defined as follows: $\delta^{[0]}(L) = L$ and $\delta^{[n]}(L) = [\delta^{[n-1]}(L), \delta^{[n-1]}(L)]$. L is called Lie solvable if there exists an integer m such that $\delta^{[m]}(L) = 0$. In this case the integer m is called the *Lie derived length* of L, and it is returned by this function.

Example

```
gap> KG := GroupRing( GF ( 2 ), DihedralGroup( 16 ) );;
gap> L := LieAlgebra( KG );
#I  LAGUNA package: Constructing Lie algebra ...
<Lie algebra over GF(2)>
gap> LieDerivedLength( L );
#I  LAGUNA package: Computing the Lie derived subalgebra ...
```

```
3
```

4.6 Other commands

4.6.1 SubgroupsOfIndexTwo

▷ `SubgroupsOfIndexTwo(G)` (attribute)

Returns a list of subgroups of G with index two. Such subgroups are important for the investigation of the Lie structure of the group algebra KG in the case of characteristic 2.

Example

```
gap> SubgroupsOfIndexTwo( DihedralGroup( 16 ) );
[ Group([ f3, f4, f1 ]), Group([ f3, f4, f2 ]), Group([ f3, f4, f1*f2 ]) ]
```

4.6.2 DihedralDepth

▷ `DihedralDepth(U)` (method)

For a finite 2-group `U`, the function returns its *dihedral depth*, which is defined to be the maximal number d such that `U` contains a subgroup isomorphic to the dihedral group of order 2^{d+1}.

Example

```
gap> KD8 := GroupRing( GF(2), DihedralGroup( 8 ) );
<algebra-with-one over GF(2), with 3 generators>
gap> UD8 := PcNormalizedUnitGroup( KD8 );
<pc group of size 128 with 7 generators>
gap> DihedralDepth( UD8 );
2
```

4.6.3 DimensionBasis

▷ `DimensionBasis(G)` (method)

Returns: record with two components: 'dimensionBasis' (list of group elements) and 'weights' (list of weights)

For a finite p-group `G`, returns its Jennings basis as it was described in Section 3.3.

Example

```
gap> G := DihedralGroup( 16 );
<pc group of size 16 with 4 generators>
gap> DimensionBasis( G );
rec( dimensionBasis := [ f1, f2, f3, f4 ], weights := [ 1, 1, 2, 4 ] )
```

4.6.4 LieDimensionSubgroups

▷ `LieDimensionSubgroups(G)` (attribute)

Returns: list of subgroups

For a finite p-group `G`, returns the series of its Lie dimension subgroups. The m-th Lie dimension subgroup $D_{(m)}$ is the intersection of the group G and $1+KG^{(m)}$, where $KG^{(m)}$ is the m-th term of the upper Lie power series of KG; see `LieUpperNilpotencyIndex` (4.5.19)

Example

```
gap> G := DihedralGroup( 16 );
<pc group of size 16 with 4 generators>
gap> LieDimensionSubgroups( G );
[ <pc group of size 16 with 4 generators>, Group([ f3, f4 ]), Group([ f4 ]),
  Group([ <identity> of ... ]) ]
```

4.6.5 LieUpperCodimensionSeries (for group ring)

▷ `LieUpperCodimensionSeries(KG)` (attribute)

▷ `LieUpperCodimensionSeries(G)` (attribute)

Returns: list of subgroups

A notion of upper Lie codimension subgroups was introduced in [CS06]. For a finite p-group `G`, C_i is the set of all elements g in `G`, such that the Lie commutator $[g,g_1,...,g_i]$ of the length $i+1$ is equal to zero for all $g_1,...,g_i$ from `G`, and $C_0=1$. By Du's theorem (see [Du92]), C_i coincides with the intersection of G and the i-th term of the upper central series $1=Z_0<Z_1<Z_2<...<Z_n=V(KG)$ of the normalized unit group $V(KG)$. This fact is used in LAGUNA to speed up computation of this series. Since $V(KG)$ is involved in computation, for the first time the argiment should be the group ring `KG`, but later you can also apply it to the group `G` itself.

Example

```
gap> G := DihedralGroup(16);
<pc group of size 16 with 4 generators>
gap> KG := GroupRing( GF(2), G );
<algebra-with-one over GF(2), with 4 generators>
gap> LieUpperCodimensionSeries( KG );
[ Group([ f1, f2, f3, f4 ]), Group([ f3, f4, f3*f4 ]), Group([ f4 ]),
  Group([ f4 ]), Group([  ]) ]
gap> LieUpperCodimensionSeries( G );
[ Group([ f1, f2, f3, f4 ]), Group([ f3, f4, f3*f4 ]), Group([ f4 ]),
  Group([ f4 ]), Group([  ]) ]
```

4.6.6 LAGInfo

▷ `LAGInfo` (info class)

`LAGInfo` is a special Info class for LAGUNA algorithms. It has 5 levels: 0, 1 (default), 2, 3 and 4. To change info level to `k`, use command `SetInfoLevel(LAGInfo, k)`.

Example

```
gap> SetInfoLevel( LAGInfo, 2 );
gap> KD8 := GroupRing( GF( 2 ), DihedralGroup( 8 ) );
<algebra-with-one over GF(2), with 3 generators>
gap> UD8 := PcNormalizedUnitGroup( KD8 );
#I  LAGInfo: Computing the pc normalized unit group ...
#I  LAGInfo: Calculating weighted basis ...
#I  LAGInfo: Calculating dimension basis ...
#I  LAGInfo: dimension basis finished !
#I  LAGInfo: Weighted basis finished !
#I  LAGInfo: Computing the augmentation ideal filtration...
#I  LAGInfo: Filtration finished !
#I  LAGInfo: finished, converting to PcGroup
<pc group of size 128 with 7 generators>
```

References

[CS06] F. Catino and E. Spinelli. Lie nilpotent group algebras and upper Lie codimension subgroups. *Comm. Algebra*, 34(10):3859–3873, 2006. 44

[Du92] X. K. Du. The centers of a radical ring. *Canad. Math. Bull.*, 35(2):174–179, 1992. 42, 44

[HB82] B. Huppert and N. Blackburn. *Finite groups. II*, volume 242 of *Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences]*. Springer-Verlag, Berlin, 1982. , AMD, 44. 13, 14

[JPS96] E. Jespers, M. M. Parmenter, and S. K. Sehgal. Central units of integral group rings of nilpotent groups. *Proc. Amer. Math. Soc.*, 124(4):1007–1012, 1996. 23

[LR86] F. Levin and G. Rosenberger. Lie metabelian group rings. In *Group and semigroup rings (Johannesburg, 1985)*, volume 126 of *North-Holland Math. Stud.*, pages 153–161. North-Holland, Amsterdam, 1986. 5, 15, 39

[PPS73] I. B. S. Passi, D. S. Passman, and S. K. Sehgal. Lie solvable group rings. *Canad. J. Math.*, 25:748–757, 1973. 5, 15, 38, 39

[Ros97] R. Rossmanith. *Centre-by-metabelian group algebras*. PhD thesis, Friedrich-Schiller-Universitat Jena, 1997. 4

[Ros00] R. Rossmanith. Lie centre-by-metabelian group algebras in even characteristic. I, II. *Israel J. Math.*, 115:51–75, 77–99, 2000. 5, 15

[Ros02] R. Rossmanith. Lie centre-by-metabelian group algebras over commutative rings. *J. Algebra*, 251(2):503–508, 2002. 40

[Sha91] A. Shalev. Lie dimension subgroups, Lie nilpotency indices, and the exponent of the group of normalized units. *J. London Math. Soc. (2)*, 43(1):23–36, 1991. 42

[Sim94] C. C. Sims. *Computation with finitely presented groups*, volume 48 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, Cambridge, 1994. 5, 13

[Wur93] M. Wursthorn. Isomorphisms of modular group algebras: an algorithm and its application to groups of order 2^6. *J. Symbolic Comput.*, 15(2):211–227, 1993. 4

Index

Augmentation, 19
augmentation homomorphism, 12
augmentation ideal, 12
AugmentationHomomorphism, 23
AugmentationIdeal, 24
AugmentationIdealNilpotencyIndex, 26
AugmentationIdealOfDerivedSubgroup-
NilpotencyIndex, 26
AugmentationIdealPowerSeries, 25

Bass cyclic unit, 23
BassCyclicUnit, 23
bicyclic unit, 22
BicyclicUnitGroup, 32
BicyclicUnitOfType1, 22
BicyclicUnitOfType2, 22

CanonicalBasis, 40
CoefficientsBySupport, 18

DihedralDepth, 43
dimension basis, 13
DimensionBasis, 43

Embedding
from group to Lie algebra, 36
from group to unit group, 29

group algebra, 12
GroupBases, 33

InverseOp, 22
Involution, 20
IsBasisOfLieAlgebraOfGroupRing, 41
IsFModularGroupAlgebra, 16
IsGroupAlgebra, 16
IsGroupOfUnitsOfMagmaRing, 30
IsLieAbelian, 38
IsLieAlgebraByAssociativeAlgebra, 34
IsLieAlgebraOfGroupRing, 36
IsLieCentreByMetabelian, 40
IsLieMetabelian, 39
IsLieNilpotent, 39
IsLieSolvable, 38
IsNormalizedUnitGroupOfGroupRing, 31
IsPModularGroupAlgebra, 17
IsSymmetric, 21
IsUnit, 21
IsUnitary, 21
IsUnitGroupOfGroupRing, 31

Jennings series, 13

LAGInfo, 44
LAGUNA package, 2
LeftIdealBySubgroup, 26
Length, 19
Lie derived length, 42
Lie derived series, 42
LieAlgebraByDomain, 34
LieCentre, 37
LieDerivedLength, 42
LieDerivedSubalgebra, 37
LieDimensionSubgroups, 44
LieLowerNilpotencyIndex, 42
LieUpperCodimensionSeries
for group, 44
for group ring, 44
LieUpperNilpotencyIndex, 41
lower Lie power series, 42

modular group algebra, 16

NaturalBijectionToAssociativeAlgebra,
35
NaturalBijectionToLieAlgebra, 35
NaturalBijectionToNormalizedUnitGroup,
28
NaturalBijectionToPcNormalizedUnit-
Group, 28
normalised unit, 12

normalised unit group, 12
NormalizedUnitGroup, 27

p-modular group algebra, 13
partial augmentation, 20
PartialAugmentations, 20
PcNormalizedUnitGroup, 27
PcUnits, 30
power-commutator presentation, 13

RadicalOfAlgebra, 24
RightIdealBySubgroup, 26

SISYPHOS package, 4
standard product, 13
StructureConstantsTable, 41
SubgroupsOfIndexTwo, 43
Support, 18
symmetric element, 21

TraceOfMagmaRingElement, 19
TwoSidedIdalBySubgroup, 26

UnderlyingAssociativeAlgebra, 34
UnderlyingField, 18
UnderlyingGroup
    of a group ring, 17
    of Lie algebra of a group ring, 36
UnderlyingGroupRing, 31
UnderlyingRing, 17
unit, 12
unit group, 12
unitary element, 21
UnitarySubgroup, 32
Units, 29
upper Lie power series, 41

weight, of dimension basis element, 13
WeightedBasis, 25