HeLP

Hertweck-Luthar-Passi method.

4.0

29 February 2024

Andreas Bächle

Leo Margolis

Andreas Bächle
Email: ABachle@gmx.net
Address: Vrije Universiteit Brussel
Vakgroep Wiskunde
Pleinlaan 2
1050 Brussels
Belgium

Leo Margolis
Email: Leo.Margolis@uam.es
Homepage: http://margollo.github.io
Address: Universidad Autonoma de Madrid
Campus Cantoblanco
Facultad de Ciencias
Departamento de Matematicas
28049 Madrid
Spain

Copyright

Acknowledgements

The authors are grateful to Sebastian Gutsche, Christof Söger and Max Horn for endowing GAP with a 4ti2-Interface and a normlaiz-Interface. We also would like to thank Gutsche and Söger for many very helpful discussions. We also want to give credits to the developers of the softwares 4ti2 and normaliz. Thanks go to David Avis for writing lrslib and answering our questions about it. We moreover thank Wolfgang Kimmerle for introducing us to the beautiful world of group rings. The development of this package was partially supported by the Research Foundation Flanders (FWO - Vlaanderen) and the DFG priority program SPP 1489 Algorithmic and Experimental Methods in Algebra, Geometry, and Number Theory.

Contents

Chapter 1

Introduction

The purpose of this package is to provide functionalities to work with torsion units in (integral) group rings. It implements a method that was developed by I.S. Luthar and I.B.S. Passi in [LP89] and which was extended by M. Hertweck in [Her07]. These names also constitute the name of the method, suggested by O. Konovalov: *He*rtweck-*L*uthar-*P*assi. The theory behind the method is briefly described in Chapter 5 and also in the survey article [BM18].

The package uses the software 4ti2 [tt] and/or normaliz [BIR^{+}] and hence is only properly working on systems which have at least one of the two installed. normaliz is usually automatically installed with the GAP-package NormalizInterface, see the documentation of that package for details. For more information on 4ti2 and to download it, please visit www.4ti2.de. To interact with these external software the package makes use of the 4ti2-Interface and normaliz-Interface written by S. Gutsche, C. Söger and M. Horn. The 4ti2-Interface in turn uses the package IO, that needs a C-part to be compiled; see the readme-file or the documentation of the IO-package for details. The package also provides the possibility to use redund from the lrslib software [Avi], to remove redundant inequalities before solving the system, which might speed up the computations significantly when 4ti2 is used. However, it is not a requirement to have lrslib installed. If the above mentioned preconditions are fulfilled, one just has to copy the HeLP-package in the GAP pkg-directory. Now the package can be loaded by typing `LoadPackage("HeLP");` during a GAP-session. If the HeLP-package doesn't work properly on your computer, you might want to check Section 6.1 for some trouble shooting.

Chapter 2

The main functions

2.1 Zassenhaus Conjecture

This function checks whether the Zassenhaus Conjecture ((ZC) for short, cf. Section 5.1) can be proved using the HeLP method with the data available in GAP.

2.1.1 HeLP_ZC

▷ `HeLP_ZC(OrdinaryCharacterTable|Group)` (function)

Returns: `true` if (ZC) can be solved using the given data, `false` otherwise

`HeLP_ZC` checks whether the Zassenhaus Conjecture can be solved for the given group using the HeLP method, the Wagner test and all character data available. The argument of the function can be either an ordinary character table or a group. In the second case it will first calculate the corresponding ordinary character table. If the group in question is nilpotent, the Zassenhaus Conjecture holds by a result of A. Weiss and the function will return `true` without performing any calculations.

If the group is not solvable, the function will check all orders which are divisors of the exponent of the group. If the group is solvable, it will only check the orders of group elements, as there can't be any torsion units of another order. The function will use the ordinary table and, for the primes p for which the group is not p-solvable, all p-Brauer tables which are available in GAP to produce as many constraints on the torsion units as possible. Additionally, the Wagner test is applied to the results, cf. Section 5.4. In case the information suffices to obtain a proof for the Zassenhaus Conjecture for this group the function will return `true` and `false` otherwise. The possible partial augmentations for elements of order k and all its powers will also be stored in the list entry `HeLP_sol[k]`.

The prior computed partial augmentations in `HeLP_sol` will not be used and will be overwritten. If you do not like the last fact, please use `HeLP_AllOrders` (3.3.1).

Example

```
gap> G := AlternatingGroup(5);
Alt( [ 1 .. 5 ] )
gap> HeLP_ZC(G);
true
gap> C := CharacterTable("A5");
CharacterTable( "A5" )
gap> HeLP_ZC(C);
true
gap> HeLP_sol;
[ [ [ [ 1 ] ] ], [ [ [ 1 ] ] ], [ [ [ 1 ] ] ],,
```

```
  [ [ [ 0, 1 ] ], [ [ 1, 0 ] ] ], [  ],,,, [  ],,,,, [  ],,,,,,,,,,,,,,, [  ]
 ]
gap> HeLP_PrintSolution();
Solutions for elements of order 2:
[ [         u ],
  [  [ "2a" ] ],
  [       --- ],
  [     [ 1 ] ] ]
Solutions for elements of order 3:
[ [         u ],
  [  [ "3a" ] ],
  [       --- ],
  [     [ 1 ] ] ]
Solutions for elements of order 5:
[ [               u ],
  [  [ "5a", "5b" ] ],
  [             --- ],
  [        [ 0, 1 ] ],
  [        [ 1, 0 ] ] ]
There are no admissible partial augmentations for elements of order 6.
There are no admissible partial augmentations for elements of order 10.
There are no admissible partial augmentations for elements of order 15.
There are no admissible partial augmentations for elements of order 30.
```

This is the classical example of Luthar and Passi to verify the Zassenhaus Conjecture for the alternating group of degree 5, cf. [LP89]. In the first call of HeLP_ZC this is checked using the character table computed by GAP using the given group, the second call uses the character table from the character table library. The entries of HeLP_sol are

- lists with entries 0 and 1 (at the spots 1, 2, 3 and 5, which correspond to torsion units that are conjugate to group elements),
- empty lists (at the spots 6, 10, 15 and 30, stating that there are no admissible partial augmentations for these orders),
- or are not bound (these orders were not checked as they don't divide the exponent of the group).

The function HeLP_PrintSolution (3.9.1) can be used to display the result in a pretty way.

Example
```
gap> C := CharacterTable( "A6" );
CharacterTable( "A6" )
gap> SetInfoLevel(HeLP_Info, 2);
gap> HeLP_ZC(C);
#I  Checking order 2.
#I  Checking order 3.
#I  Checking order 4.
#I  Checking order 5.
#I  Checking order 6.
#I  Checking order 10.
#I  Checking order 12.
#I  Checking order 15.
#I  Checking order 20.
```

```
#I  Checking order 30.
#I  Checking order 60.
#I  ZC can't be solved, using the given data, for the orders: [ 6 ].
false
gap> HeLP_sol[6];
[ [ [ 1 ], [ 0, 1 ], [ -2, 2, 1 ] ], [ [ 1 ], [ 1, 0 ], [ -2, 1, 2 ] ] ]
gap> HeLP_PrintSolution(6);
Solutions for elements of order 6:
[ [                   u^3,                  u^2,                       u ],
  [              [ "2a" ],        [ "3a", "3b" ],  [ "2a", "3a", "3b" ] ],
  [                   ---,                  ---,                     --- ],
  [                 [ 1 ],             [ 0, 1 ],          [ -2, 2, 1 ] ],
  [                 [ 1 ],             [ 1, 0 ],          [ -2, 1, 2 ] ] ]
gap> SetInfoLevel(HeLP_Info, 1);
```

This is the example M. Hertweck deals with in his article [Her08c]. The HeLP-method is not sufficient to verify the Zassenhaus Conjecture for this group. There are two tuples of possible partial augmentations for torsion units of order 6 which are admissible by the HeLP method. M. Hertweck used a different argument to eliminate these possibilities.

Example

```
gap> G := SmallGroup(48,30);;
gap> StructureDescription(G);
"A4 : C4"
gap> HeLP_ZC(G);
#I  ZC can't be solved, using the given data, for the orders: [ 4 ].
false
gap> Size(HeLP_sol[4]);
10
```

The group SmallGroup(48,30) is the smallest group for which the HeLP method does not suffice to prove the Zassenhaus Conjecture. However (ZC) was proved for this group in [HK06], Proposition 4.2.

Example

```
gap> C1 := CharacterTable(SymmetricGroup(5));
CharacterTable( Sym( [ 1 .. 5 ] ) )
gap> HeLP_ZC(C1);
#I  The Brauer tables for the following primes are not available: [ 2, 3, 5 ].
#I  ZC can't be solved, using the given data, for the orders: [ 4, 6 ].
false
gap> C2 := CharacterTable("S5");
CharacterTable( "A5.2" )
gap> HeLP_ZC(C2);
true
```

This example demonstrates the advantage of using the GAP character table library: Since GAP can't compute the Brauer tables from the ordinary table of S_5 in the current implementation, they are not used in the first calculation. But in the second calculation HeLP_ZC accesses the Brauer tables from the library and can prove the Zassenhaus Conjecture for this group, see [Her07], Section 5. This example might of course change as soon as GAP will be able to compute the needed Brauer tables.

Example

```
gap> C := CharacterTable("M11");
CharacterTable( "M11" )
```

```
gap> HeLP_ZC(C);
#I  ZC can't be solved, using the given data, for the orders: [ 4, 6, 8 ].
false
gap> HeLP_sol[12];
[  ]
gap> HeLP_PrintSolution(8);
Solutions for elements of order 8:
[ [       u^4,                          u^2,                            u ],
  [ [ "2a" ],               [ "2a", "4a" ],  [ "2a", "4a", "8a", "8b" ] ],
  [       ---,                          ---,                          --- ],
  [     [ 1 ],                   [ 0, 1 ],                [ 0, 0, 0, 1 ] ],
  [     [ 1 ],                   [ 0, 1 ],                [ 0, 0, 1, 0 ] ],
  [     [ 1 ],                   [ 0, 1 ],              [ 0, 2, -1, 0 ] ],
  [     [ 1 ],                   [ 0, 1 ],              [ 0, 2, 0, -1 ] ],
  [     [ 1 ],                  [ 2, -1 ],               [ 0, 0, 0, 1 ] ],
  [     [ 1 ],                  [ 2, -1 ],               [ 0, 0, 1, 0 ] ],
  [     [ 1 ],                  [ 2, -1 ],              [ 0, 2, -1, 0 ] ],
  [     [ 1 ],                  [ 2, -1 ],              [ 0, 2, 0, -1 ] ] ]
```

Comparing this example to the result in [BK07a] one sees, that the existence of elements of order 12 in $V(\mathbb{Z}M_{11})$ may not be eliminated using only the HeLP method. This may be done however by applying also the Wagner test, cf. Section 5.4 and the example for the function `HeLP_WagnerTest` (3.7.1).

This example also demonstrates, why also the partial augmentations of the powers of u must be stored (and not only the partial augmentations of u). To prove that all elements of order 8 in $V(\mathbb{Z}M_{11})$ are rationally conjugate to group elements, it is not enough to prove that all elements u of order 8 in $V(\mathbb{Z}M_{11})$ have all partial augmentations 1 and 0, as the fifth and sixth possibility from above still could exist in $V(\mathbb{Z}M_{11})$, which would not be rationally conjugate to group elements.

Example

```
gap> G := SmallGroup(144, 117);
<pc group of size 144 with 6 generators>
gap> HeLP_ZC(G);
true
```

This seems to contradict the result of [BHK+18]. This is due to the fact that the HeLP-package, from version 4.0 on, does include additional restrictions on partial augmentations, in particular a result of Hertweck on p-adic conjugacy which is relevant for this group, cf. Section 5.2.

2.2 Prime Graph Question

This function checks whether the Prime Graph Question ((PQ) for short, cf. Section 5.1) can be verified using the HeLP method with the data available in GAP.

2.2.1 HeLP_PQ

▷ `HeLP_PQ(OrdinaryCharacterTable|Group)` (function)

Returns: `true` if (PQ) can be solved using the given data, `false` otherwise

`HeLP_PQ` checks whether an affirmative answer for the Prime Graph Question for the given group can be obtained using the HeLP method, the Wagner restrictions and the data available. The argument of the function can be either an ordinary character table or a group. In the second case it will first calculate the corresponding ordinary character table. If the group in question is solvable, the Prime Graph Question has an affirmative answer by a result of W. Kimmerle and the function will return `true` without performing any calculations.

If the group is non-solvable, the ordinary character table and all p-Brauer tables for primes p for which the group is not p-solvable and which are available in GAP will be used to produce as many constraints on the torsion units as possible. Additionally, the Wagner test is applied to the results, cf. Section 5.4. In case the information suffices to obtain an affirmative answer for the Prime Graph Question, the function will return `true` and it will return `false` otherwise. Let p and q be distinct primes such that there are elements of order p and q in G but no elements of order pq. Then for k being p, q or pq the function will save the possible partial augmentations for elements of order k and its (non-trivial) powers in `HeLP_sol[k]`. The function also does not use the previously computed partial augmentations for elements of these orders but will overwrite the content of `HeLP_sol`. If you do not like the last fact, please use `HeLP_AllOrdersPQ` (3.3.2).

Example
```
gap> C := CharacterTable("A7");
CharacterTable( "A7" )
gap> HeLP_PQ(C);
true
```

The Prime Graph Question for the alternating group of degree 7 was first proved by M. Salim [Sal11].

Example
```
gap> C := CharacterTable("L2(19)");
CharacterTable( "L2(19)" )
gap> HeLP_PQ(C);
true
gap> HeLP_ZC(C);
#I  (ZC) can't be solved, using the given data, for the orders: [ 10 ].
false
gap> HeLP_sol[10];
[ [ [ 1 ], [ 0, 1 ], [ 0, -1, 1, 0, 1 ] ],
  [ [ 1 ], [ 0, 1 ], [ 0, 0, 0, 1, 0 ] ],
  [ [ 1 ], [ 1, 0 ], [ 0, 0, 0, 0, 1 ] ],
  [ [ 1 ], [ 1, 0 ], [ 0, 1, -1, 1, 0 ] ] ]
```

The HeLP method provides an affirmative answer to the Prime Graph Question for the group L2(19), although the method doesn't solve the Zassenhaus Conjecture for that group, as there are two sets of possible partial augmentations for units of order 10 left, which do not correspond to elements which are rationally conjugate to group elements. The Zassenhaus Conjecture for this group is proved in [BM17b].

Example
```
gap> C1 := CharacterTable(PSL(2,7));
CharacterTable( Group([ (3,7,5)(4,8,6), (1,2,6)(3,4,8) ]) )
gap> HeLP_PQ(C1);
#I  The Brauer tables for the following primes are not available: [ 2, 3, 7 ].
#I  PQ can't be solved, using the given data, for the orders: [ 6 ].
false
gap> C2 := CharacterTable("L2(7)");
CharacterTable( "L3(2)" )
gap> HeLP_PQ(C2);
true
```

This example demonstrates the advantage of using tables from the GAP character table library: Since GAP can not compute the Brauer tables corresponding to `C1` they are not used in the first calculation. But in the second calculation `HeLP_PQ` accesses the Brauer tables from the library and can prove the Prime Graph Question for this group, see [Her07], Section 6. This example might change, as soon as GAP will be able to compute the Brauer tables needed.

Example

```
gap> SetInfoLevel(HeLP_Info,2);
gap> C := CharacterTable("A6");
CharacterTable( "A6" )
gap> HeLP_PQ(C);
#I  Checking order 2.
#I  Checking order 3.
#I  Checking order 5.
#I  Checking order 6.
#I  Checking order 10.
#I  Checking order 15.
#I  PQ can't be solved, using the given data, for the orders: [ 6 ].
false
gap> SetInfoLevel(HeLP_Info,1);
```

The Prime Graph Question can not be confirmed for the alternating group of degree 6 with the HeLP-method. This group is handled in [Her08c] by other means.

Example

```
gap> C := CharacterTable("L2(49)");
CharacterTable( "L2(49)" )
gap> HeLP_PQ(C);
#I  The Brauer tables for the following primes are not available: [ 7 ].
#I  (PQ) can't be solved, using the given data, for the orders: [ 10, 15 ].
false
```

This example shows the limitations of the program. Using the Brauer table for the prime 7 one can prove (PQ) for PSL(2,49), but this data is not available in GAP at the moment. The fact that there are no torsion units of order 10 and 15 was proved in [Her07], Proposition 6.7. See also the example in Section 4.5. The other critical orders were handled in a more general context in [BM17a].

2.3 Spectrum Problem

This function checks whether the Spectrum Problem ((SP) for short, cf. Section 5.1) can be verified using the HeLP method with the data available in GAP.

2.3.1 HeLP_SP

▷ `HeLP_SP(`*`OrdinaryCharacterTable|Group`*`)` (function)

Returns: `true` if (SP) can be solved using the given data, `false` otherwise

`HeLP_SP` checks whether an affirmative answer for the Spectrum Problem for the given group can be obtained using the HeLP method, the Wagner restrictions and the data available. The argument of the function can be either an ordinary character table or a group. In the second case it will first calculate the corresponding ordinary character table. If the group in question is solvable, the Spectrum

Problem has an affirmative answer by a result of M. Hertweck and the function will return `true` without performing any calculations.

If the group is non-solvable, the ordinary character table and all p-Brauer tables for primes p for which the group is not p-solvable and which are available in GAP will be used to produce as many constraints on the torsion units as possible. Additionally, the Wagner test is applied to the results, cf. Section 5.4. In case the information suffices to obtain an affirmative answer for the Spectrum Problem, the function will return `true` and it will return `false` otherwise. The possible partial augmentations for elements of order k and all its powers will also be stored in the list entry `HeLP_sol[k]`.

The function also does not use the previously computed partial augmentations for elements of these orders but will overwrite the content of `HeLP_sol`. If you do not like the last fact, please use `HeLP_AllOrdersSP` (3.3.3).

The Prime Graph Question for the first Janko group was studied in [BJK11] and solved positively using the HeLP-method. We note that this is not possible for the Spectrum Problem.

Example

```
gap> C := CharacterTable("J1");
CharacterTable( "J1" )
gap> HeLP_PQ(C);
true
gap> HeLP_SP(C);
#I  (SP) can't be solved, using the given data, for the orders: [ 30 ].
false
```

The Spectrum Problem is still known to have a positive answer for the first Janko group by [BM21], Proposition 3.8.

2.4 Kimmerle Problem

This function checks whether the Kimmerle Problem ((KP) for short, cf. Section 5.1) can be verified using the HeLP method with the data available in GAP.

2.4.1 HeLP_KP

▷ `HeLP_KP(OrdinaryCharacterTable|Group)` (function)

Returns: `true` if (KP) can be solved using the given data, `false` otherwise

`HeLP_KP` checks whether an affirmative answer for the Kimmerle Problem for the given group can be obtained using the HeLP method, the Wagner restrictions and the data available. The argument of the function can be either an ordinary character table or a group. In the second case it will first calculate the corresponding ordinary character table. If the group in question is nilpotent, then even the Zassenahus Conjecture and so also the Kimmerle Problem has an affirmative answer by a result of Al Weiss and the function will return `true` without performing any calculations.

If the group is non-nilpotent, the ordinary character table and all p-Brauer tables for primes p for which the group is not p-solvable and which are available in GAP will be used to produce as many constraints on the torsion units as possible. Additionally, the Wagner test is applied to the results, cf. Section 5.4. In case the information suffices to obtain an affirmative answer for the Kimmerle Problem, the function will return `true` and it will return `false` otherwise. The possible partial augmentations for elements of order k and all its powers will also be stored in the list entry `HeLP_sol[k]`.

The function also does not use the previously computed partial augmentations for elements of these orders but will overwrite the content of HeLP_sol. If you do not like the last fact, please use HeLP_AllOrdersKP (3.3.4).

Example

```
gap> G := SmallGroup(216, 33);
<pc group of size 216 with 6 generators>
gap> HeLP_ZC(G);
#I  (ZC) can't be solved, using the given data, for the orders: [ 12 ].
false
gap> HeLP_KP(G);
true
gap> C := CharacterTable("L2(19)");
CharacterTable( "L2(19)" )
HeLP_ZC(C);
#I  (ZC) can't be solved, using the given data, for the orders: [ 10 ].
false
gap> HeLP_KP(C);
true
gap> C := CharacterTable("A7");
CharacterTable( "A7" )
gap> HeLP_KP(C);
#I  (KP) can't be solved, using the given data, for the orders: [ 4, 6 ].
false
gap> HeLP_sol[6];
[ [ [ 1 ], [ 0, 1 ], [ -2, 2, 1, 0 ] ], [ [ 1 ], [ 0, 1 ], [ 0, 1, -1, 1 ] ],
  [ [ 1 ], [ 0, 1 ], [ 2, -1, 1, -1 ] ], [ [ 1 ], [ 1, 0 ], [ -2, 1, 2, 0 ] ],
  [ [ 1 ], [ 1, 0 ], [ 0, 0, 0, 1 ] ], [ [ 1 ], [ 1, 0 ], [ 2, 0, 0, -1 ] ] ]
```

We see remark that units of order 6 are known to be rationally conjugate to trivial units by other methods [BM21]. For order 4 however, this remains open.

Chapter 3

Further functions

A short remark is probably in order on the three global variables the package is using: `HeLP_CT`, `HeLP_sol` and `HeLP_settings`. The first one stores the character table for which the last calculations were performed, the second one containing at the `k`'s spot the already calculated admissible partial augmentations of elements of order k (and its powers u^d for $d \neq k$ a divisor of k). If a function of the HeLP-package is called with a character table different from the one saved in `HeLP_CT` then the package tries to check if the character tables belong to the same group. This can be done in particular for tables from the ATLAS. If this check is successful the solutions already written in `HeLP_sol` are kept, otherwise this variable is reset. For a more detailed account see Sections 4.2, 5.2 and `HeLP_ChangeCharKeepSols` (3.4.1). In most situations, the user does not have to worry about this, the program will take care of it as far as possible. `HeLP_settings` is a varaible which is used to store some settings on how linear inequalities are solved by the package.

3.1 Checks for specific orders

3.1.1 HeLP_WithGivenOrder

▷ `HeLP_WithGivenOrder(`*`CharacterTable|ListOfClassFunctions, ord`*`)` (function)

Returns: List of admissible partial augmentations

Calculates the admissible partial augmentations for elements of order *`ord`* using only the data given in the first argument. The first argument can be an ordinary character table, a Brauer table, or a list of class functions, all having the same underlying character table. This function only uses the constraints of the HeLP method (from the class functions given), but does not apply the Wagner test 5.4. If the constraints allow only a finite number of solutions, these lists will be written in `HeLP_sol[ord]`. If for divisors d of *`ord`* solutions are already calculated and stored in `HeLP_sol[d]`, these will be used, otherwise the function `HeLP_WithGivenOrder` will first be applied to this order and the data given in the first argument.

Example
```
gap> C := CharacterTable("A5");
CharacterTable( "A5" )
gap> HeLP_WithGivenOrder(C, 5);
#I  Number of solutions for elements of order 5: 2; stored in HeLP_sol[5].
[ [ [ 0, 1 ] ], [ [ 1, 0 ] ] ]
gap> HeLP_PrintSolution(5);
Solutions for elements of order 5:
```

```
[ [                u ],.
  [  [ "5a", "5b" ] ],
  [              --- ],
  [        [ 0, 1 ] ],
  [        [ 1, 0 ] ] ]
```

Tests which partial augmentations for elements of order 5 are admissible.

Example

```
gap> C := CharacterTable("A6");
CharacterTable( "A6" )
gap> HeLP_WithGivenOrder(C, 4);
#I  Number of solutions for elements of order 4: 4; stored in HeLP_sol[4].
[ [ [ 1 ], [ -1, 2 ] ], [ [ 1 ], [ 2, -1 ] ], [ [ 1 ], [ 1, 0 ] ],
  [ [ 1 ], [ 0, 1 ] ] ]
gap> HeLP_sol[4];
[ [ [ 1 ], [ -1, 2 ] ], [ [ 1 ], [ 2, -1 ] ], [ [ 1 ], [ 1, 0 ] ],
  [ [ 1 ], [ 0, 1 ] ] ]
```

Two of the non-trivial partial augmentations can be eliminated by using the Brauer table modulo the prime 3:

Example

```
gap> HeLP_WithGivenOrder(C mod 3, 4);
#I  Number of solutions for elements of order 4: 2; stored in HeLP_sol[4].
[ [ [ 1 ], [ 1, 0 ] ], [ [ 1 ], [ 0, 1 ] ] ]
```

When using HeLP_ZC also the last remaining non-trivial partial augmentation disappears, as this function applies the Wagner test, cf. 5.4 and HeLP_WagnerTest (3.7.1):

Example

```
gap> HeLP_ZC(C);
#I  ZC can't be solved, using the given data, for the orders: [ 6 ].
false
gap> HeLP_sol[4]; HeLP_sol[6];
[ [ [ 1 ], [ 0, 1 ] ] ]
[ [ [ 1 ], [ 0, 1 ], [ -2, 2, 1 ] ], [ [ 1 ], [ 1, 0 ], [ -2, 1, 2 ] ] ]
```

The following example demonstrates how one can use lists of characters to obtain constraints for partial augmentations:

Example

```
gap> C := CharacterTable("L2(49).2_1");
CharacterTable( "L2(49).2_1" )
gap> HeLP_WithGivenOrder(Irr(C), 7);;
#I  Number of solutions for elements of order 7: 1; stored in HeLP_sol[7].
gap> HeLP_WithGivenOrder(Irr(C){[2]}, 14);
#I  The given data admit infinitely many solutions for elements of order 14.
gap> HeLP_WithGivenOrder(Irr(C){[44]}, 14);
#I  The given data admit infinitely many solutions for elements of order 14.
gap> HeLP_WithGivenOrder(Irr(C){[2,44]}, 14);
#I  Number of solutions for elements of order 14: 0; stored in HeLP_sol[14].
[  ]
```

Brauer tables can provide more restrictions on partial augmentations of certain torsion units:

Example
```
gap> C := CharacterTable("J1");
CharacterTable( "J1" )
gap> HeLP_WithGivenOrder(C, 6);;
#I  Number of solutions for elements of order 6: 73; stored in HeLP_sol[6].
gap> B := C mod 11;
BrauerTable( "J1", 11 )
gap> HeLP_WithGivenOrder(B, 6);;
#I  Number of solutions for elements of order 6: 6; stored in HeLP_sol[6].
gap> HeLP_WithGivenOrder(Irr(B){[2,3]}, 6);;
#I  Number of solutions for elements of order 6: 6; stored in HeLP_sol[6].
gap> HeLP_PrintSolution(6);
Solutions for elements of order 6:
[ [                  u^3,                    u^2,                         u ],
  [              [ "2a" ],               [ "3a" ],  [ "2a", "3a", "6a" ] ],
  [                  ---,                    ---,                       --- ],
  [                [ 1 ],                  [ 1 ],              [ -2, 0, 3 ] ],
  [                [ 1 ],                  [ 1 ],              [ 2, 0, -1 ] ],
  [                [ 1 ],                  [ 1 ],               [ 0, 0, 1 ] ],
  [                [ 1 ],                  [ 1 ],              [ -4, 3, 2 ] ],
  [                [ 1 ],                  [ 1 ],              [ 0, 3, -2 ] ],
  [                [ 1 ],                  [ 1 ],              [ -2, 3, 0 ] ] ]
```

The result of the previous example can be found in [BJK11]. The existence of such units was later excluded in [BM21].

When dealing with many variables using lists of characters instead of a complete character table might also speed up the calculations a lot, see Section 4.3.

Example
```
gap> C := CharacterTable("L2(27)");
CharacterTable( "L2(27)" )
gap> HeLP_WithGivenOrder(C,7);;
#I  Number of solutions for elements of order 7: 78; stored in HeLP_sol[7].
gap> SetInfoLevel(HeLP_Info,4);
gap> HeLP_WithGivenOrder(C,3*7);
#I      Solutions for order 3 not yet calculated.  Restart for this order.
#I  Number of solutions for elements of order 21: 0; stored in HeLP_sol[21].
[  ]
gap> SetInfoLevel(HeLP_Info,1);
```

`HeLP_WithGivenOrder` often needs to consider many cases. Set the info class HeLP_Info to a level 4 or higher to keep track of the progress, see Section 4.4 on info levels.

3.1.2 HeLP_WithGivenOrderAndPA

▷ `HeLP_WithGivenOrderAndPA(`*`CharacterTable|ListOfClassFunctions, ord, partaugs`*`)` (function)

Returns: List of admissible partial augmentations

Calculates the admissible partial augmentations for elements of order *ord* using only the data given in the first argument. The first argument can be an ordinary character table, a Brauer table, or a list of class functions, all having the same underlying character table. The function uses the partial augmentations for the powers u^d with d divisors of k different from 1 and k given in *partaugs*. Here, the d's have to be in a descending order (i.e. the orders of the u^d's are ascending). This function only uses the constraints of the HeLP method, but does not apply the Wagner test 5.4. Note that this function will not affect `HeLP_sol`.

Example

```
gap> G := SmallGroup(48,33);; StructureDescription(G);
"SL(2,3) : C2"
gap> C := CharacterTable(G);;
gap> HeLP_WithGivenOrder(C, 4);;
#I  Number of solutions for elements of order 4: 4; stored in HeLP_sol[4].
gap> HeLP_WithGivenOrder(C, 6);;
#I  Number of solutions for elements of order 6: 2; stored in HeLP_sol[6].
gap> HeLP_sol[4]; HeLP_sol[6];
[ [ [ 1, 0 ], [ 0, 1, 0, 0, 0 ] ], [ [ 1, 0 ], [ 0, 0, 0, 0, 1 ] ],
  [ [ 1, 0 ], [ 0, 0, 0, 1, 0 ] ], [ [ 1, 0 ], [ 0, 0, 1, 0, 0 ] ] ]
[ [ [ 1, 0 ], [ 0, 1 ], [ 0, 0, 0, 0, 1, 0 ] ],
  [ [ 1, 0 ], [ 1, 0 ], [ 0, 0, 0, 0, 0, 1 ] ] ]
gap> HeLP_WithGivenOrderAndPA(C, 12, [ [ 1, 0 ],  [ 0, 1 ], [ 0, 0, 0, 0, 1 ],
>      [ 0, 0, 0, 0, 1, 0 ] ]);
#I  Number of solutions for elements of order 12 with these partial augmentation
s for the powers: 1.
[ [ [ 1, 0 ], [ 0, 1 ], [ 0, 0, 0, 0, 1 ], [ 0, 0, 0, 0, 1, 0 ],
      [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0 ] ] ]
gap> HeLP_WithGivenOrderAndPA(C, 12, [ [ 1, 0 ],  [ 0, 1 ], [ 0, 0, 0, 1, 0 ],
>      [ 0, 0, 0, 0, 1, 0 ] ]);
#I  Number of solutions for elements of order 12 with these partial augmentation
s for the powers: 0.
[  ]
```

In the calls of `HeLP_WithGivenOrderAndPA` the function uses the following partial augmentations:

- `[ 1, 0 ]` for the element u^6 of order 2,
- `[ 0, 1 ]` for the element u^4 of order 3,
- `[ 0, 0, 0, 0, 1 ]` and `[ 0, 0, 0, 1, 0 ]` for the element u^3 of order 4 respectively,
- `[ 0, 0, 0, 0, 1, 0 ]` for the element u^2 of order 6.

3.1.3 HeLP_WithGivenOrderAllTables

▷ `HeLP_WithGivenOrderAllTables(`*`CharacterTable, ord`*`)` (function)

Returns: List of admissible partial augmentations

Calculates the admissible partial augmentations for elements of order *ord* using the given character table *CharacterTable* and all Brauer tables that can be obtained from it. *CharacterTable* can be an ordinary or a Brauer table. In any case, then given table will be used first to obtain a finite number of solutions (if the characteristic does not divide *ord*, otherwise the ordinary table will be used), with the other tables only checks will be performed to restrict the number of possible partial augmentations as much as possible. If certain Brauer tables are not avaialble, this will be printed if HeLP_Info is at least 1. This function only uses the constraints of the HeLP method, but does not apply the Wagner test 5.4. If the constraints allow only a finite number of solutions, these lists will be written in `HeLP_sol[ord]`. If for divisors d of *ord* solutions are already calculated and stored in

HeLP_sol[d], these will be used, otherwise the function HeLP_WithGivenOrder will first be applied to this order and the data given in the first argument.

3.1.4 HeLP_WithGivenOrderAndPAAllTables

▷ HeLP_WithGivenOrderAndPAAllTables(*CharacterTable, ord, partaugs*) (function)

Returns: List of admissible partial augmentations

Calculates the admissible partial augmentations for elements of order *ord* using the given character table *CharacterTable* and all other tables that can be obtained from it. *CharacterTable* can be an ordinary or a Brauer table. In any case, then given table will be used first to obtain a finite number of solutions (if the characteristic does not divide *ord*, otherwise the ordinary table will be used), with the other tables only checks will be performed to restrict the number of possible partial augmentations as much as possible. If certain Brauer tables are not avaialble, this will be printed if HeLP_Info is at least 1. The function uses the partial augmentations for the powers u^d with d divisors of k different from 1 and k given in *partaugs*. Here, the d's have to be in a descending order (i.e. the orders of the u^d's are ascending). This function only uses the constraints of the HeLP method, but does not apply the Wagner test 5.4. Note that this function will not affect HeLP_sol.

3.1.5 HeLP_WithGivenOrderAndPAAndSpecificSystem

▷ HeLP_WithGivenOrderAndPAAndSpecificSystem(*list, ord, partaugs[, b]*) (function)

Returns: List of admissible partial augmentations

Calculates the admissible partial augmentations for elements of order *ord* using only the data given in the first argument. The first argument is a list, which can contains as entries characters or pairs with first entry a character and second entrie an integer or a mixture of these. The first argument is understood as follows: If a character χ is not given in a pair all inequalities obtainable by this character are used. If it is given in a pair with the integer m the inequalities obtainable from the multiplicity of E(ord) taken to the power m as an eigenvalue of a representation affording χ are used. The function uses the partial augmentations for the powers u^d with d divisors of k different from 1 and k given in *partaugs*. Here, the d's have to be in a descending order (i.e. the orders of the u^d's are ascending). This function only uses the constraints of the HeLP method, but does not apply the Wagner test 5.4. Note that this function will not affect HeLP_sol.

Example
```
gap> C := CharacterTable("A5");
CharacterTable( "A5" )
gap> chi := Irr(C)[2];; psi := Irr(C)[4];
Character( CharacterTable( "A5" ), [ 4, 0, 1, -1, -1 ] )
gap> HeLP_WithGivenOrderAndPAAndSpecificSystem([[chi, 1], [chi, 2]],
>  5, [ ], true);
[ [ [ [ 0, 1 ] ], [ [ 1, 0 ] ] ], [ [ -3/5, 2/5 ], [ 2/5, -3/5 ] ], [ 3/5, 3/5 ] ]
gap> sol5 := HeLP_WithGivenOrderAndPAAndSpecificSystem([[chi, 1], [chi, 2]],
>  5, [ ]);
[ [ [ 0, 1 ] ], [ [ 1, 0 ] ] ]
```

The inequalities in the above examples are:

$$\frac{-3}{5}\varepsilon_{5a}(u)+\frac{2}{5}\varepsilon_{5b}(u)+\frac{3}{5}\in\mathbb{Z}_{\geq 0} \text{ and } \frac{2}{5}\varepsilon_{5a}(u)+\frac{-3}{5}\varepsilon_{5b}(u)+\frac{3}{5}\in\mathbb{Z}_{\geq 0}.$$

Continuing the above example:

Example

```
gap> HeLP_WithGivenOrderAndPAAndSpecificSystem([psi],
>  2*5, [[1], sol5[1][1]], true);
[ [  ], [ [ 0, -2/5, -2/5 ], [ 0, -1/10, -1/10 ], [ 0, 1/10, 1/10 ],
  [ 0, -1/10, -1/10 ], [ 0, 1/10, 1/10 ], [ 0, 2/5, 2/5 ],
  [ 0, 1/10, 1/10 ], [ 0, -1/10, -1/10 ], [ 0, 1/10, 1/10 ],
[ 0, -1/10, -1/10 ] ], [ 0, 1/2, 1/2, 1/2, 1/2, 0, 1/2, 1/2, 1/2, 1/2 ] ]
gap> HeLP_WithGivenOrderAndPAAndSpecificSystem([[psi, 0], [psi, 2], [psi, 5]],
>  2*5, [[1], sol5[2][1]], true);
[ [  ], [ [ 0, -2/5, -2/5 ], [ 0, 1/10, 1/10 ], [ 0, 2/5, 2/5 ] ], [ 0, 1/2, 0 ] ]
```

3.2 Checks for specific orders with s-constant characters

When considering elements of order st (in absence of elements of this order in the group ; in particular when trying to prove (PQ)) and there are several conjugacy classes of elements of order s, it might be useful to consider s-constant characters (cf. Section 5.5) to reduce the computational complexity.

3.2.1 HeLP_WithGivenOrderSConstant

▷ `HeLP_WithGivenOrderSConstant(`*`CharacterTable|ListOfClassFunctions, s, t`*`)` (function)

Returns: List of admissible "partial augmentations" or `"infinite"`

Calculates the admissible partial augmentations for elements u of order $s*t$ using only the s-constant class functions that are contained in the first argument. The first argument can be an ordinary character table, a Brauer table, or a list of class functions, all having the same underlying character table. `s` and `t` have to be different prime numbers, such that there are elements of order `s` and `t` in the group, but no elements of order $s*t$.

The function filters which class functions given in the first argument are constant on all conjugacy classes of elements of order `s`. For the element u^s of order `t` the partial augmentations given in `HeLP_sol[t]` are used. If they are not yet calculated, the function calculates them first, using the data given in the first argument and stores them in `HeLP_sol[t]`. This function only uses the constraints of the HeLP method, but does not apply the Wagner test 5.4. If these calculations allow an infinite number of solutions of elements of order st the function returns `"infinite"`, otherwiese it returns the finite list of solutions for elements of order $s*t$. The first entry of every solution is a list of the partial augmentations of u^s and the second entry is a list of the "partial augmentations" for u: the first entry of this list is the sum of the partial augmentations on all classes of elements of order `s` and the other entries are the partial augmentations on the classes of order `t`. Only in the case that the existence of units of order $s*t$ can be excluded by this function the variable `HeLP_sol[s*t]` will be affected and `HeLP_sol[s*t]` will be set to `[ ]`.

Example

```
gap> C := CharacterTable("A6");;
gap> HeLP_WithGivenOrder(C, 6);
#I  Number of solutions for elements of order 6: 2; stored in HeLP_sol[6].
[ [ [ 1 ], [ 0, 1 ], [ -2, 2, 1 ] ], [ [ 1 ], [ 1, 0 ], [ -2, 1, 2 ] ] ]
gap> HeLP_WithGivenOrderSConstant(C, 2, 3);
[ [ [ 0, 1 ], [ -2, 2, 1 ] ], [ [ 1, 0 ], [ -2, 1, 2 ] ] ]
gap> HeLP_WithGivenOrderSConstant(C, 3, 2);
[ [ [ 1 ], [ 3, -2 ] ] ]
```

Example

```
gap> C := CharacterTable("Sz(8)");;
gap> SetInfoLevel(HeLP_Info, 4);
gap> HeLP_WithGivenOrderSConstant(C, 7, 13);
#I    Partial augmentations for elements of order 13 not yet calculated.  Restar
t for this order.
#I    Number of non-trivial 7-constant characters in the list: 7.
[  ]
gap> SetInfoLevel(HeLP_Info, 1);
```

The last example can also be checked by using all characters in `C`, but this takes notably longer.

Example

```
gap> C := CharacterTable("Sz(32)");
CharacterTable( "Sz(32)" )
gap> L := Filtered(OrdersClassRepresentatives(C), x-> x = 31);; Size(L);
15          # I.e. HeLP_WithGivenOrder(C,31) would take hopelessly long
gap> HeLP_WithGivenOrderSConstant(C mod 2, 31, 5);
[  ]
gap> IsBound(HeLP_sol[31]);
false
```

We still have no clue about elements of order 31, but there are none of order 5*31.

3.2.2 HeLP_AddGaloisCharacterSums

▷ `HeLP_AddGaloisCharacterSums(CT)` (function)

Returns: List of characters

Given an ordinary character table *CT* the function calculates the orbits under the action of the Galois group and returns a list of characters containing the ones contained in *CT* and the ones obtained by summing up the Galois-orbits.

3.3 Checks for all orders

3.3.1 HeLP_AllOrders

▷ `HeLP_AllOrders(CharacterTable|Group)` (function)

Returns: `true` if (ZC) can be solved using the given data, `false` otherwise

This function does almost the same as `HeLP_ZC` (2.1.1). It checks whether the Zassenhaus Conjecture can be verified for a group, but does not compute the partial augmentations of elements of order k, if `HeLP_sol[k]` already exists. It does however verify the solutions given in `HeLP_sol` using all available tables for the group, see `HeLP_VerifySolution` (3.6.1). Thus some precalculations using e.g. `HeLP_WithGivenOrder` (3.1.1) are respected. In contrast to `HeLP_ZC` (2.1.1) this function also does not check whether the group is nilpotent to use the Weiss-result to have an immediate positive solution for (ZC).

This function is interesting if one wants to save time or possesses some information, which was not obtained using this package and was entered manually into `HeLP_sol`.

Example

```
gap> C := CharacterTable(PSL(2,7));
CharacterTable( Group([ (3,7,5)(4,8,6), (1,2,6)(3,4,8) ]) )
```

```
gap> HeLP_ZC(C);
#I  The Brauer tables for the following primes are not available: [ 2, 3, 7 ].
#I  (ZC) can't be solved, using the given data, for the orders: [ 6 ].
false
gap> HeLP_sol[6] := [ ];
[  ]
gap> HeLP_AllOrders(C);
true
```

3.3.2 HeLP_AllOrdersPQ

▷ HeLP_AllOrdersPQ(*CharacterTable|Group*) (function)

Returns: `true` if (PQ) can be solved using the given data, `false` otherwise

This function does almost the same as `HeLP_PQ` (2.2.1). It checks whether the Prime Graph Question can be verified for a group, but does not compute the partial augmentations of elements of order k, if `HeLP_sol[k]` already exists. Thus some precalculations using e.g. `HeLP_WithGivenOrder` (3.1.1) are respected. In contrast to `HeLP_PQ` (2.2.1) this function also does not check whether the group is solvable to use the Kimmerle-result to have an immediate positive solution for (PQ).

This function is interesting if one wants to save time or possesses some information, which was not obtained using this package and was entered manually into `HeLP_sol`.

Example

```
gap> C := CharacterTable("A12");
CharacterTable( "A12" )
gap> HeLP_WithGivenOrder(Irr(C){[2, 4, 7]}, 2);;
#I  Number of solutions for elements of order 2: 37; stored in HeLP_sol[2].
gap> HeLP_WithGivenOrderSConstant(C mod 3,11,2);
[  ]
gap> HeLP_WithGivenOrder(Irr(C mod 2){[2, 3, 4, 6]}, 3);;
#I  Number of solutions for elements of order 3: 99; stored in HeLP_sol[3].
gap> HeLP_WithGivenOrderSConstant(C mod 2, 11, 3);
[  ]
gap> HeLP_AllOrdersPQ(C);
true
```

Thus the Prime Graph Question holds for the alternating group of degree 12. Just using `HeLP_PQ(C)` would take hopelessly long.

3.3.3 HeLP_AllOrdersSP

▷ HeLP_AllOrdersSP(*CharacterTable|Group*) (function)

Returns: `true` if (SP) can be solved using the given data, `false` otherwise

This function does almost the same as `HeLP_SP` (2.3.1). It checks whether the Spectrum Problem can be verified for a group, but does not compute the partial augmentations of elements of order k, if `HeLP_sol[k]` already exists. Thus some precalculations using e.g. `HeLP_WithGivenOrder` (3.1.1) are respected. In contrast to `HeLP_SP` (2.3.1) this function also does not check whether the group is solvable to use the Hertweck-result to have an immediate positive solution for (SP).

This function is interesting if one wants to save time or possesses some information, which was not obtained using this package and was entered manually into `HeLP_sol`.

3.3.4 HeLP_AllOrdersKP

▷ `HeLP_AllOrdersKP(CharacterTable|Group)` (function)

Returns: `true` if (KP) can be solved using the given data, `false` otherwise

This function does almost the same as `HeLP_KP` (2.4.1). It checks whether the Kimmerle Problem can be verified for a group, but does not compute the partial augmentations of elements of order k, if `HeLP_sol[k]` already exists. Thus some precalculations using e.g. `HeLP_WithGivenOrder` (3.1.1) are respected. In contrast to `HeLP_KP` (2.4.1) this function also does not check whether the group is nilpotent to use the Weiss-result to have an immediate positive solution for (KP).

This function is interesting if one wants to save time or possesses some information, which was not obtained using this package and was entered manually into `HeLP_sol`.

3.4 Changing the used Character Table

3.4.1 HeLP_ChangeCharKeepSols

▷ `HeLP_ChangeCharKeepSols(CT)` (function)

Returns: nothing

This function changes the used character table to the character table *CT* and keeps all the solutions calculated so far. It is in this case the responsibility of the user that the tables belong to the same group and the ordering of the conjugacy classes in *CT* is consistent with the one in the previously used table. This function can be used to change from one table of the group to another, e.g. from a Brauer table to the ordinary table if the calculations will involve p-singular elements. (In case the involved character tables come from the ATLAS and their InfoText begins with "origin: ATLAS of finite groups", this is done automatically by the program.) A user may also use characters, which are normally not accessible in GAP.

To keep track of the change of the character tables one can set HeLP_Info to level 5. In this first example it is not realized that the character tables belong to the same group, so the solutions for elements of order 2 are recalculated (they have been reset, as another character table is used).

Example

```
gap> SetInfoLevel(HeLP_Info, 5);
gap> C := CharacterTable(SymmetricGroup(4));
CharacterTable( Sym( [ 1 .. 4 ] ) )
gap> HeLP_WithGivenOrder(C mod 2, 3);
#I  USED CHARACTER TABLE CHANGED TO BrauerTable( SymmetricGroup( [ 1 .. 4 ] ), 2
 ), ALL GLOBAL VARIABLES RESET.
#I  Number of solutions for elements of order 3: 1; stored in HeLP_sol[3].
[ [ [ 1 ] ] ]
gap> HeLP_WithGivenOrder(C, 2*3);
#I  USED CHARACTER TABLE CHANGED TO CharacterTable( SymmetricGroup( [ 1 .. 4 ] )
 ), ALL GLOBAL VARIABLES RESET.
#I      Solutions for order 2 not yet calculated.  Restart for this order.
#I      Solutions for order 3 not yet calculated.  Restart for this order.
#I  Number of solutions for elements of order 6: 0; stored in HeLP_sol[6].
[  ]
```

The recalculations of the solutions can be avoided by calling `HeLP_ChangeCharKeepSols` before using another character table.

Example
```
gap> D := CharacterTable(SymmetricGroup(4));
CharacterTable( Sym( [ 1 .. 4 ] ) )
gap> HeLP_WithGivenOrder(D mod 2, 3);
#I  USED CHARACTER TABLE CHANGED TO BrauerTable( SymmetricGroup( [ 1 .. 4 ] ), 2
 ), ALL GLOBAL VARIABLES RESET.
#I  Number of solutions for elements of order 3: 1; stored in HeLP_sol[3].
[ [ [ 1 ] ] ]
gap> HeLP_ChangeCharKeepSols(D);
#I  WARNING: Change used character table without checking if the character table
s have the same underlying groups and the ordering of the conjugacy classes are
the same!
gap> HeLP_WithGivenOrder(D, 2*3);
#I  Using same character table as until now; all known solutions kept.
#I      Solutions for order 2 not yet calculated.  Restart for this order.
#I  Number of solutions for elements of order 6: 0; stored in HeLP_sol[6].
[  ]
```

When using tables from the ATLAS this is done automatically:

Example
```
gap> CA := CharacterTable("A5");
CharacterTable( "A5" )
gap> HeLP_WithGivenOrder(CA mod 2, 5);
#I  USED CHARACTER TABLE CHANGED TO BrauerTable( "A5", 2 ), ALL GLOBAL VARIABLES
 RESET.
#I  Testing possibility 1 out of 1.
#I  Number of solutions for elements of order 5: 2; stored in HeLP_sol[5].
[ [ [ 0, 1 ] ], [ [ 1, 0 ] ] ]
gap> HeLP_WithGivenOrder(CA, 2*5);
#I  Using character table of the same group; all known solutions kept.
#I      Solutions for order 2 not yet calculated.  Restart for this order.
#I  Number of solutions for elements of order 10: 0; stored in HeLP_sol[10].
[  ]
gap> SetInfoLevel(HeLP_Info, 1);
```

3.4.2 HeLP_Reset

▷ `HeLP_Reset()` (function)

Returns: nothing

This function delets all the values calculated so far and resets the global variables `HeLP_CT` and `HeLP_CT` to their initial value `[ [ [1] ] ]` and `CharacterTable(SmallGroup(1,1))` respectively.

3.5 Influencing how the Systems of Inequalities are solved

HeLP uses currently three external programs (i.e. programs that are not part of the GAP-system): zsolve from 4ti2 and/or normaliz to solve the systems of linear inequalities and redund from lrslib to simplify the inequlities before handing them over to the solver (HeLP can also be used without lrslib installed. In general it is recommanded to have lrslib installed, if 4ti2 is used as the solver). The following functions can be used to influence the behaviour of these external programms.

3.5.1 HeLP_Solver

▷ `HeLP_Solver([string])` (function)

Returns: nothing

This function can be used to change the solver used for the HeLP-system between 4ti2 and normaliz. If the function is called without an argument it prints which solver is currently used. If the argument it is called with is one of the stings "4ti2" or "normaliz", then the solver used for future calculations is changed to the one given as argument in case this solver is found by the HeLP-package. If both solvers are found when the package is loaded normaliz is taken as default.

3.5.2 HeLP_UseRedund

▷ `HeLP_UseRedund(bool)` (function)

Returns: nothing

This function determines whether HeLP uses 'redund' from the lrslib-package to remove redundant equations from the HeLP system. If *bool* is `true` 'redund' will be used in all calculation that follow, if it is `false`, 'redund' will not be used (which might take significantly longer). If 'redund' was not found by GAP a warning will be printed and the calculations will be performed without 'redund'. As default 'redund' will be used in all calculations, if 4ti2 is the chosen solver, and 'redund' will not be used, if normaliz is used.

3.5.3 HeLP_Change4ti2Precision

▷ `HeLP_Change4ti2Precision(string)` (function)

Returns: nothing

This function changes the maximum precision of the calculations of 4ti2 to solve the occurring systems of linear inequalities. The possible arguments are `"32"`, `"64"` and `"gmp"`. After calling the function the new precision will be used until this function is used again. The default value is `"32"`. A higher precision causes slower calculations. But this function might be used to increase the precision of 4ti2, when one gets an error message like "Error, 4ti2 Error: Results were near maximum precision (32bit). Please restart with higher precision!" stating that the results were close to the maximum 4ti2-precision. normaliz does automatically change its precision, when it reaches an overflow.

Sometimes it is desirable to perform calculations without redund (even if it is installed and in many cases improves the performance of the package) or with a higher precision. For example, determining the partial augmentations for units of order 14 for `SmallGroup(392, 30)` involves very long calculations (when called with redund and precision 32) or cause errors (when called without redund and precision 32). However, the following works in a reasonable time.

Example
```
gap> C := CharacterTable(SmallGroup(392,30));
CharacterTable( <pc group of size 392 with 5 generators> )
gap> HeLP_Solver("4ti2");
'4ti2' will be used from now on.
gap> HeLP_UseRedund(false);
The calculations will be performed without using 'redund' from now on.
gap> HeLP_ZC(C);
Error, 4ti2 Error:
Results were near maximum precision (32bit).
Please restart with higher precision!
If you continue, your results might be wrong called from
```

```
4ti2Interface_zsolve_equalities_and_inequalities(
 [ ListWithIdenticalEntries( Size( T[1] ), 1 ) ], [ 1 ], temp[1], - temp[2]
 ) called from
HeLP_TestSystemINTERNAL( W[1], W[2], k, arg[3] ) called from
HeLP_WithGivenOrderAndPAINTERNAL( C, k, pa ) called from
HeLP_WithGivenOrderINTERNAL( Irr( T ), k ) called from
<function "HeLP_ZC">( <arguments> )
 called from read-eval loop at line 19 of *stdin*
you can 'quit;' to quit to outer loop, or
you can 'return;' to continue
gap> brk> quit;
#I  Options stack has been reset
gap> HeLP_Change4ti2Precision("64");
The calculations of 4ti2 will be performed with precision 64 from now on.
gap> HeLP_ZC(C);
true
```

The reproducibility of the above example depends on the versions of the progrmas involved and probably also your machine.

3.5.4 HeLP_Vertices

▷ `HeLP_Vertices(`*string*`)` (function)

Returns: nothing

If normlaiz is used as the solver of the HeLP-system this function influences, whether the "VerticesOfPolyhedron" are computed by normaliz. By default these are only computed, if the system has a trivial solution. The function takes "vertices", "novertices" and "default" as arguments. If you do not understand what this means, don't worry.

3.6 Checking solutions, calculating and checking solutions

3.6.1 HeLP_VerifySolution

▷ `HeLP_VerifySolution(`*CharacterTable|ListOfClassFunctions, k[, list_paraugs]*`)` (function)

Returns: List of admissible partial augmentations

This function checks which of the partial augmentations for elements of order `k` given in `HeLP_sol[k]` or the optional third argument `list_paraugs` fulfill the HeLP equations obtained from the characters in the first argument. This function does not solve any inequalities, but only checks, if the given partial augmentations fulfill them. It is for this reason often faster then e.g. `HeLP_WithGivenOrder` (3.1.1).

If there is no third argument given, i.e. the augmentations from `HeLP_sol[k]` are used, the result overwrites `HeLP_sol[k]`.

Example

```
gap> C := CharacterTable("A6");;
gap> HeLP_WithGivenOrder(C, 4);
#I  Number of solutions for elements of order 4: 4; stored in HeLP_sol[4].
[ [ [ 1 ], [ -1, 2 ] ], [ [ 1 ], [ 2, -1 ] ], [ [ 1 ], [ 1, 0 ] ],
  [ [ 1 ], [ 0, 1 ] ] ]
```

```
gap> HeLP_VerifySolution(C mod 3, 4);
[ [ [ 1 ], [ 1, 0 ] ], [ [ 1 ], [ 0, 1 ] ] ]
gap> HeLP_sol[4];
[ [ [ 1 ], [ 1, 0 ] ], [ [ 1 ], [ 0, 1 ] ] ]
```

Example

```
gap> C := CharacterTable("S12");;
gap> HeLP_WithGivenOrder(Irr(C mod 5){[2..6]}, 2);;
#I  Number of solutions for elements of order 2: 563; stored in HeLP_sol[2].
gap> HeLP_VerifySolution(C mod 5, 2);;
gap> Size(HeLP_sol[2]);
387
gap> HeLP_VerifySolution(C mod 3, 2);;
gap> Size(HeLP_sol[2]);
324
```

Using `HeLP_WithGivenOrder(C mod 5, 2)` or `HeLP_WithGivenOrder(C mod 3, 2)` takes much longer since in that case a bigger system of inequalities must be solved.

3.6.2 HeLP_FindAndVerifySolution

▷ `HeLP_FindAndVerifySolution(CharacterTable|ListOfClassFunctions, k)` (function)

Returns: List of admissible partial augmentations or "infinite"

This function provides the same functionality as `HeLP_WithGivenOrder` (3.1.1) but instead of constructiong the corresponding system with all characters from the first argument *CharacterTable|ListOfClassFunctions* it does it consecutively with larger sets of characters from the argument until a finite list of solutions is found and then applies `HeLP_VerifySolution` (3.6.1) to these solutions with the entirety of the class functions in the first argument.

This function is sometimes faster than `HeLP_WithGivenOrder` (3.1.1), but the output is the same, thus the examples from `HeLP_WithGivenOrder` (3.1.1) also apply here.

3.6.3 HeLP_PossiblePartialAugmentationsOfPowers

▷ `HeLP_PossiblePartialAugmentationsOfPowers(n)` (function)

Returns: List of partial augmentations of powers.

This function provides the possible partial augmentations of the powers of units of a given order n, if the partial augmentations if units of order n/p have been already computed for all primes p dividing n. The possibilities are sorted in the same way as, if the order n is checked with any other function like e.g. `HeLP_WithGivenOrder` (3.1.1) or `HeLP_ZC` (2.1.1). Thus, if the InfoLevel is high enough and one obtains that the computation of some possibility is taking too long, one can check it using `HeLP_WithGivenOrderAndPA` (3.1.2).

Example

```
gap> SetInfoLevel(HeLP_Info,4);
gap> C := CharacterTable(SmallGroup(160,91));
CharacterTable( <pc group of size 160 with 6 generators> )
gap> HeLP_WithGivenOrder(C,4);;
#I      Solutions for order 2 not yet calculated.  Restart for this order.
#I  Number of solutions for elements of order 4: 22; stored in HeLP_sol[4].
gap> HeLP_WithGivenOrder(C,10);;
#I      Solutions for order 5 not yet calculated.  Restart for this order.
```

```
#I  Number of solutions for elements of order 10: 6; stored in HeLP_sol[10].
gap> LP := HeLP_PossiblePartialAugmentationsOfPowers(20);;
gap> HeLP_WithGivenOrderAndPA(Irr(C){[2..20]},20,LP[1]);
#I  Number of solutions for elements of order 20 with these partial augmentations
 for the powers: 0.
[  ]
```

3.6.4 HeLP_WriteTrivialSolution

▷ `HeLP_WriteTrivialSolution(C, k)` (function)

Returns: Trivial solutions.

Given a character table `C` and an order `k`, the function calculates the partial augmentations of units of order k that are rationally conjugate to group elements (note that they just coincide with the partial augmentations of group elements) and stores them in `HeLP_sol[k]`. If solutions of order k were already calculated, they are overwritten by this function, so this function can be used in particular if elements of order k are known to be rationally conjugate to group elements by theoretical results.

With the character tables that are currently available in GAP, the Zassenhaus Conjecture for elements of order 4 in $PSL(2,49)$ cannot be solved. However it was proved in [Her07] using the Brauer table modulo 7.

Example

```
gap> C := CharacterTable("L2(49)");
CharacterTable( "L2(49)" )
gap> HeLP_WithGivenOrder(C, 4);
#I  Number of solutions for elements of order 4: 14; stored in HeLP_sol[4].
[ [ [ 1 ], [ -6, 7 ] ], [ [ 1 ], [ -5, 6 ] ], [ [ 1 ], [ -4, 5 ] ],
  [ [ 1 ], [ -3, 4 ] ], [ [ 1 ], [ -2, 3 ] ], [ [ 1 ], [ -1, 2 ] ],
  [ [ 1 ], [ 0, 1 ] ], [ [ 1 ], [ 1, 0 ] ], [ [ 1 ], [ 2, -1 ] ],
  [ [ 1 ], [ 3, -2 ] ], [ [ 1 ], [ 4, -3 ] ], [ [ 1 ], [ 5, -4 ] ],
  [ [ 1 ], [ 6, -5 ] ], [ [ 1 ], [ 7, -6 ] ] ]
gap> C mod 7;
fail
gap> HeLP_WriteTrivialSolution(C, 4);;
gap> HeLP_sol[4];
[ [ [ 1 ], [ 0, 1 ] ] ]
```

3.7 The Wagner test

3.7.1 HeLP_WagnerTest

▷ `HeLP_WagnerTest(k[, list_paraugs, OrdinaryCharacterTable])` (function)

Returns: List of admissible partial augmentations

This function applies the Wagner test (cf. Section 5.4) to the given data. If only the order `k` is given as argument, the Wagner test will be applied to the solutions stored in `HeLP_sol[k]`. If the arguments are the order `k`, a list of possible solutions `list_paraugs` and an ordinary character table `OrdinaryCharacterTable` it applies the test to the solutions given in `list_paraugs` and using the number of conjugacy classes for elements a divisor of `k`, which will be extracted from the head of `OrdinaryCharacterTable`.

Example

```
gap> C := CharacterTable("M11");
CharacterTable( "M11" )
gap> HeLP_WithGivenOrder(C,8);;
#I  Number of solutions for elements of order 8: 36; stored in HeLP_sol[8].
gap> HeLP_sol[8] := HeLP_WagnerTest(8);;
gap> Size(HeLP_sol[8]);
24
```

Thus the Wagner-Test eliminates 12 possible partial augmentations for elements of order 8. Continuing the example:

Example

```
gap> HeLP_WithGivenOrder(C,12);
#I  Number of solutions for elements of order 12: 7; stored in HeLP_sol[12].
[ [ [ 1 ], [ 1 ], [ 2, -1 ], [ 0, 3, -2 ], [ 1, 0, -1, 1 ] ],
  [ [ 1 ], [ 1 ], [ 1, 0 ], [ 0, 3, -2 ], [ 0, 0, 0, 1 ] ],
  [ [ 1 ], [ 1 ], [ -1, 2 ], [ 0, 3, -2 ], [ 0, 0, 2, -1 ] ],
  [ [ 1 ], [ 1 ], [ 0, 1 ], [ 0, 3, -2 ], [ 1, 0, 1, -1 ] ],
  [ [ 1 ], [ 1 ], [ 0, 1 ], [ 0, 3, -2 ], [ -1, 0, 1, 1 ] ],
  [ [ 1 ], [ 1 ], [ 1, 0 ], [ 0, -3, 4 ], [ 0, 0, 0, 1 ] ],
  [ [ 1 ], [ 1 ], [ -1, 2 ], [ 0, -3, 4 ], [ 1, 0, -1, 1 ] ] ]
gap> HeLP_sol[12] := HeLP_WagnerTest(12);
[ [ [ 1 ], [ 1 ], [ 1, 0 ], [ 0, 3, -2 ], [ 0, 0, 0, 1 ] ],
  [ [ 1 ], [ 1 ], [ -1, 2 ], [ 0, 3, -2 ], [ 0, 0, 2, -1 ] ],
  [ [ 1 ], [ 1 ], [ 1, 0 ], [ 0, -3, 4 ], [ 0, 0, 0, 1 ] ] ]
gap> HeLP_sol[4] := HeLP_WagnerTest(4);;
gap> HeLP_WithGivenOrder(C,12);
#I  Number of solutions for elements of order 12: 3; stored in HeLP_sol[12].
[ [ [ 1 ], [ 1 ], [ 2, -1 ], [ 0, 3, -2 ], [ 1, 0, -1, 1 ] ],
  [ [ 1 ], [ 1 ], [ 0, 1 ], [ 0, 3, -2 ], [ 1, 0, 1, -1 ] ],
  [ [ 1 ], [ 1 ], [ 0, 1 ], [ 0, 3, -2 ], [ -1, 0, 1, 1 ] ] ]
gap> HeLP_sol[12] := HeLP_WagnerTest(12);
[  ]
```

Thus there are no normalized units of order 12 in the integral group ring of M_{11}.

Example

```
gap> C := CharacterTable("M22");
CharacterTable( "M22" )
gap> HeLP_WagnerTest(12, [ [ [1], [1], [1,0], [0,0,1], [-3,3,2,3,-4] ] ],C);
[  ]
```

This example is taken from the appendix of [BKL08].

Sometimes the Wagner-Test may even prove the Zassenhaus Conjecture:

Example

```
gap> G := SmallGroup(96,187);
<pc group of size 96 with 6 generators>
gap> C := CharacterTable(G);
CharacterTable( <pc group of size 96 with 6 generators> )
gap> HeLP_WithGivenOrder(C,4);;
#I  Number of solutions for elements of order 4: 34; stored in HeLP_sol[4].
gap> HeLP_WagnerTest(4);
```

```
[ [ [ 0, 0, 1, 0, 0 ], [ 0, 0, 0, 0, 0, 0, 0, 0, 1 ] ],
  [ [ 0, 0, 1, 0, 0 ], [ 0, 0, 0, 0, 0, 0, 1, 0, 0 ] ],
  [ [ 0, 1, 0, 0, 0 ], [ 0, 0, 0, 0, 0, 1, 0, 0, 0 ] ],
  [ [ 0, 1, 0, 0, 0 ], [ 0, 0, 0, 0, 0, 0, 0, 1, 0 ] ] ]
gap> HeLP_ZC(C);
true
```

3.8 Action of the automorphism group

3.8.1 HeLP_AutomorphismOrbits

▷ HeLP_AutomorphismOrbits(*C, k[, list_paraug]*) (function)

Returns: List of admissible partial augmentations

For a list of possible partial augmentations, this function calculates representatives of each orbit of the action of the automorphism group of G on them. The first two mandatory arguments are an ordinary character table *C* (with an underlying group) and the order *k* for which the partial augmentations should be filtered with respect to the action of the automorphism group of G. If as third argument a list of partial augmentations is given, then these will be used, otherwise the partial augmentations that are stored in HeLP_sol[k] are used.

Example

```
gap> C := CharacterTable("A6");;
gap> HeLP_WithGivenOrder(C, 6);
#I  Number of solutions for elements of order 6: 2; stored in HeLP_sol[6].
[ [ [ 1 ], [ 0, 1 ], [ -2, 2, 1 ] ], [ [ 1 ], [ 1, 0 ], [ -2, 1, 2 ] ] ]
gap> HeLP_AutomorphismOrbits(C, 6);
[ [ [ 1 ], [ 0, 1 ], [ -2, 2, 1 ] ] ]
```

3.9 Output

3.9.1 HeLP_PrintSolution

▷ HeLP_PrintSolution(*[k]*) (function)

Returns: nothing

This function prints the possible solutions in a pretty way. If a positive integer *k* as argument is given, then it prints the admissible partial augmentations of units of order *k*, if they are already calculated. If no argument is given, the function prints information on all orders for which there is already some information available.

Example

```
gap> C := CharacterTable("A5");;
gap> HeLP_ZC(C);
true
gap> HeLP_PrintSolution();
Solutions for elements of order 2:
[ [         u ],
  [  [ "2a" ] ],
  [       --- ],
  [     [ 1 ] ] ]
Solutions for elements of order 3:
```

```
[ [          u ],
  [  [ "3a" ] ],
  [        --- ],
  [     [ 1 ] ] ]
Solutions for elements of order 5:
[ [                u ],
  [  [ "5a", "5b" ] ],
  [              --- ],
  [        [ 0, 1 ] ],
  [        [ 1, 0 ] ] ]
There are no admissible partial augmentations for elements of order 6.
There are no admissible partial augmentations for elements of order 10.
There are no admissible partial augmentations for elements of order 15.
There are no admissible partial augmentations for elements of order 30.
gap> C := CharacterTable("A6");;
gap> HeLP_ZC(C);
#I  ZC can't be solved, using the given data, for the orders: [ 6 ].
false
gap> HeLP_PrintSolution(6);
Solutions for elements of order 6:
[ [                   u^3,                  u^2,                        u ],
  [              [ "2a" ],        [ "3a", "3b" ],  [ "2a", "3a", "3b" ] ],
  [                   ---,                  ---,                      --- ],
  [                 [ 1 ],             [ 0, 1 ],            [ -2, 2, 1 ] ],
  [                 [ 1 ],             [ 1, 0 ],            [ -2, 1, 2 ] ] ]
```

3.10 Eigenvalue multiplicities and character values

3.10.1 HeLP_MultiplicitiesOfEigenvalues

▷ HeLP_MultiplicitiesOfEigenvalues(*chi, k, paraugs*) (function)

Returns: a list of multiplicities of eigenvalues

The returned list contains at the l-th spot the multiplicity of `E(k)^(l-1)` as eigenvalue of a unit u of order `k` under the representation corresponding to *chi* having the partial augmentations *paraugs* for the elements u^d for divisors d different from `k`.

3.10.2 HeLP_CharacterValue

▷ HeLP_CharacterValue(*chi, k, paraug*) (function)

Returns: the character value $chi(u)$

The function returns the character value $chi(u)$ of an element u of order k having the partial augmentations *paraug*.

Example

```
gap> C := CharacterTable("A6");;
gap> HeLP_WithGivenOrder(C, 6);
#I  Number of solutions for elements of order 6: 2; stored in HeLP_sol[6].
[ [ [ 1 ], [ 0, 1 ], [ -2, 2, 1 ] ], [ [ 1 ], [ 1, 0 ], [ -2, 1, 2 ] ] ]
gap> chi := Irr(C)[2];;   # a character of degree 5
gap> HeLP_MultiplicitiesOfEigenvalues(chi, 6, HeLP_sol[6][2]);
[ 1, 0, 1, 2, 1, 0 ]
```

```
gap> HeLP_CharacterValue(chi, 6, HeLP_sol[6][2][3]);
-2
gap> HeLP_CharacterValue(chi, 6, [-2,1,2]);
-2
gap> HeLP_CharacterValue(chi, 6, [-2,2,1]);
1
```

These eigenvalues were computed manually by M. Hertweck and may be found in [Her08c].

3.11 Check for triviality modulo normal subgroup

3.11.1 HeLP_IsOneModuloN

▷ `HeLP_IsOneModuloN(UCT, k, pa, G, N)` (function)

Returns: `true` or `false`

This function checks, if the image of a unit in $\mathrm{V}(\mathbb{Z}G)$ given by the partial augmentations of itself (not its powers) is trivial modulo a normal subgroup N, i.e. if it maps to the identity under the natural homomorphism $\mathbb{Z}G \to \mathbb{Z}(G/N)$. The input is a character table, the order of the unit, its partial augmentations, the group and the normal subgroup.

Example

```
gap> G := SmallGroup(144,117);
<pc group of size 144 with 6 generators>
gap> C := CharacterTable(G);
CharacterTable( <pc group of size 144 with 6 generators> )
gap> N := PCore(G, 3);
Group([ f5, f6 ])
gap> Size(N);
9
gap> HeLP_IsOneModuloN(C, 3, [1,0], G, N);
true
gap> HeLP_ZC(C);
true
```

The fact that the unit of order 3 lies in the kernel of the map modulo N explains why `HeLP_ZC` produces a result different from the one recorded in [BHK+18]. Namely, the unit of order 6 described there has non-vanishing partial augmentations at classes of order 6 with non-conjugate 3-parts, which contradicts the p-adic criterion of Hertweck [Mar17], as the 3-part of the unit is trivial modulo the 3-core of the group.

3.11.2 HeLP_ForgetUnderlyingGroup

▷ `HeLP_ForgetUnderlyingGroup(C)` (function)

Returns: Same character table as `C` but without underlying group

Continuing the previous example we see that the normal subgroup structure of the group is needed here to get a positive solution for (ZC).

Example

```
gap> CCop := HeLP_ForgetUnderlyingGroup(C);
CharacterTable( "CT4" )
gap> HeLP_ZC(CCop);
```

```
#I  (ZC) can't be solved, using the given data, for the orders: [ 6 ].
false
```

3.12 Check Kimmerle Problem for single units

3.12.1 HeLP_UnitSatisfiesKP

▷ HeLP_UnitSatisfiesKP(*UCT*, *k*, *pa*) (function)

Returns: `true` or `false`

Decides if a unit described by the partial augmentations of its powers satisfies the Kimmerle Problem. Input is Ordinary character table, order of the unit and the partial augmentations.

Example

```
gap> C := CharacterTable("A7");
CharacterTable( "A7" )
gap> HeLP_ZC(C);
#I  (ZC) can't be solved, using the given data, for the orders: [ 4, 6 ].
false
gap> HeLP_sol[4];
[ [ [ 1 ], [ 0, 1 ] ], [ [ 1 ], [ 2, -1 ] ] ]
gap> HeLP_UnitSatisfiesKP(C, 4, HeLP_sol[4][1]);
true
gap> HeLP_UnitSatisfiesKP(C, 4, HeLP_sol[4][2]);
false
```

3.13 Check whether Zassenhaus Conjecture is known from theoretical results

3.13.1 HeLP_IsZCKnown

▷ HeLP_IsZCKnown(*G*) (function)

Returns: `true` if (ZC) can be derived from theoretical results, `false` otherwise

For the given group `G` this function applies five checks, namely it checks

- if G is nilpotent
- if G has a normal Sylow subgroup with abelian complement,
- if G is cyclic-by-abelian
- if it is of the form $X \rtimes A$, where X and A are abelian and A is of prime order p such that p is smaller then any prime divisor of the order of X
- or if the order of G is smaller than 144.

In all these cases the Zassenhaus Conjecture is known. See 5.6 for references. This function is designed for solvable groups.

Chapter 4

Extended examples

We will give some more examples which are intended to give the user an idea of the behavior on different inputs and the variable HeLP_sol. We also give hints how to use the package more efficiently, to use characters not available in libraries and how InfoLevels can be helpful.

4.1 The Character Table Library

Example

```
gap> G := SL(2,7);
SL(2,7)
gap> HeLP_ZC(G);
#I  The Brauer tables for the following primes are not available: [ 2, 3, 7 ].
#I  (ZC) can't be solved, using the given data, for the orders: [ 8 ].
false
gap> C1 := CharacterTable(G);
CharacterTable( SL(2,7) )
gap> HeLP_ZC(C1);
#I  The Brauer tables for the following primes are not available: [ 2, 3, 7 ].
#I  (ZC) can't be solved, using the given data, for the orders: [ 8 ].
false
gap> C2 := CharacterTable("2.L2(7)");
CharacterTable( "2.L3(2)" )
gap> HeLP_ZC(C2);
true
```

Note that the first and the second call of `HeLP_ZC` (2.1.1) are equivalent – the only difference being that in the first version the character table of the group is also calculated by the function, in the second version the calculations are performed with the given character table. For the third call of `HeLP_ZC` (2.1.1) a character table for SL2(7) is used which comes from the character table library. The different result is due to the fact, that in the third version the Brauer tables are available (the Brauer table for the prime $p = 7$ is needed to rule out some non-trivial partial augmentations for elements of order 8), whereas for the first and the second call no Brauer tables are available in GAP.

4.2 The behavior of the variable HeLP sol

This sections demonstrates when the global variable `HeLP_sol` is reset. This is the case if calculations are performed using (the character table of) another group than before:

Example
```
gap> C := CharacterTable("A5");
CharacterTable( "A5" )
gap> HeLP_ZC(C);
true
gap> HeLP_sol;
[ [ [ [ 1 ] ] ], [ [ [ 1 ] ] ], [ [ [ 1 ] ] ],,
  [ [ [ 0, 1 ] ], [ [ 1, 0 ] ] ], [  ],,,, [  ],,,,, [  ],,,,,,,,,,,,,,, [  ]
 ]
gap> C := CharacterTable("L3(7).2");
CharacterTable( "L3(7).2" )
gap> HeLP_WithGivenOrderAndPA(Irr(C){[3,7,9,10]},21,[[1],[3,9,-11]]);
#I  Number of solutions for elements of order 21 with these partial augmentation
s for the powers: 1.
[ [ [ 1 ], [ 3, 9, -11 ], [ -6, 0, 3, 4 ] ] ]
gap> HeLP_sol;
[ [ [ [ 1 ] ] ] ]
```

The function `HeLP_WithGivenOrderAndPA` (3.1.2) does not write a result in `HeLP_sol[k]` (as it does not calculate all possible solutions of order k). However `HeLP_sol` is reset as a different character table is used. We continue the above example.

Example
```
gap> HeLP_WithGivenOrder(C,3);
#I  Number of solutions for elements of order 3: 1; stored in HeLP_sol[3].
[ [ [ 1 ] ] ]
gap> HeLP_sol;
[ [ [ [ 1 ] ] ],, [ [ [ 1 ] ] ] ]
```

If HeLP detects that the table used belongs to the same group, `HeLP_sol` is not reset:

Example
```
gap> HeLP_WithGivenOrder(C mod 7, 19);
#I  Number of solutions for elements of order 19: 3; stored in HeLP_sol[19].
[ [ [ 0, 0, 1 ] ], [ [ 0, 1, 0 ] ], [ [ 1, 0, 0 ] ] ]
gap> HeLP_sol;
[ [ [ [ 1 ] ] ],, [ [ [ 1 ] ] ],,,,,,,,,,,,,,,,
  [ [ [ 0, 0, 1 ] ], [ [ 0, 1, 0 ] ], [ [ 1, 0, 0 ] ] ] ]
# the previously calaculated result for order 3 is still there.
```

HeLP can detect that the character tables belong to the same group, if they are identical objects in GAP or if both are tables of the same group from the ATLAS and their InfoText begins with "origin: ATLAS of finite groups" (which is usually the case for ATLAS tables). If the program can verify that the character table which is used at the current call of a function belongs to the same group as in the previous call of a function, the solutions stored in `HeLP_sol` are kept. If the character table belongs to another group or it can not be made sure that the character tabel belongs to the same group, `HeLP_sol` is reset to the initial value `[ [ [1] ] ]` representing the trivial solution for units of order 1.

Not reseting `HeLP_sol` can also be achieved using `HeLP_ChangeCharKeepSols` (3.4.1). However, caution should be exercised when using this command since it may manipulate `HeLP_sol` into something meaningless.

Example

```
gap> G := PSL(2,7);
Group([ (3,7,5)(4,8,6), (1,2,6)(3,4,8) ])
gap> HeLP_ZC(G);
#I  The Brauer tables for the following primes are not available: [ 2, 3, 7 ].
#I  (ZC) can't be solved, using the given data, for the orders: [ 6 ].
false
gap> HeLP_sol;
[ [ [ [ 1 ] ] ], [ [ [ 1 ] ] ], [ [ [ 1 ] ] ], [ [ [ 1 ], [ 0, 1 ] ] ],,
  [ [ [ 1 ], [ 1 ], [ -2, 3 ] ] ], [ [ [ 0, 1 ] ], [ [ 1, 0 ] ] ],,,,, [  ],,
  [  ],,,,,,, [  ],,,,,,, [  ],,,,,,,,,,,,,, [  ],,,,,,,,,,,,,,,,,,,,,,,,,,,,,
  ,,,,,,,,,,,,, [  ] ]
gap> C := CharacterTable("L2(7)") mod 7;
BrauerTable( "L3(2)", 7 )
gap> HeLP_ChangeCharKeepSols(C); #This table belongs to the same group.
gap> HeLP_WithGivenOrder(C,6);
#I  Number of solutions for elements of order 6: 0; stored in HeLP_sol[6].
[  ]
gap> HeLP_sol;
[ [ [ [ 1 ] ] ], [ [ [ 1 ] ] ], [ [ [ 1 ] ] ], [ [ [ 1 ], [ 0, 1 ] ] ],,
  [  ], [ [ [ 0, 1 ] ], [ [ 1, 0 ] ] ],,,,, [  ],, [  ],,,,,,, [  ],,,,,,,
  [  ],,,,,,,,,,,,,, [  ],,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,, [  ] ]
gap> C := CharacterTable("L3(7).2") mod 7;
BrauerTable( "L3(7).2", 7 )
gap> HeLP_ChangeCharKeepSols(C); #This table is from a different group
gap> HeLP_WithGivenOrder(C,19);
#I  Number of solutions for elements of order 19: 3; stored in HeLP_sol[19].
[ [ [ 0, 0, 1 ] ], [ [ 0, 1, 0 ] ], [ [ 1, 0, 0 ] ] ]
gap> HeLP_sol;
[ [ [ [ 1 ] ] ], [ [ [ 1 ] ] ], [ [ [ 1 ] ] ], [ [ [ 1 ], [ 0, 1 ] ] ],,
  [  ], [ [ [ 0, 1 ] ], [ [ 1, 0 ] ] ],,,,, [  ],, [  ],,,,,
  [ [ [ 0, 0, 1 ] ], [ [ 0, 1, 0 ] ], [ [ 1, 0, 0 ] ] ],, [  ],,,,,,, [  ],,,,
  ,,,,,,,,,, [  ],,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,, [  ] ]
# The content of HeLP_sol does not have a mathematical value anymore.
```

The following functions manipulate the variable `HeLP_sol`: `HeLP_ZC` (2.1.1), `HeLP_PQ` (2.2.1), `HeLP_WithGivenOrder` (3.1.1), `HeLP_WithGivenOrderSConstant` (3.2.1) (for elements of order t and if the existence of elements of order st can be excluded also for this order), `HeLP_AllOrders` (3.3.1), `HeLP_AllOrdersPQ` (3.3.2), `HeLP_VerifySolution` (3.6.1) (if existing solutions were checked), `HeLP_FindAndVerifySolution` (3.6.2). Note that the functions only will write results in `HeLP_sol[k]` if k is a divisor of the exponent of the group as this information is enough to decide whether (ZC) and (PQ) are valid for the group in consideration. In all other cases an empty list will be returned but no value will be written in `HeLP_sol[k]`.

4.3 Saving time

The most time consuming operation when using the functions of this package is solving the system of inequalities given by the HeLP method, see Section 5.3. This package uses the programs 4ti2 and/or normaliz to do this and it is not completely clear to the authors of this package which input is solved faster by these programs. In any case it is helpful to reduce the number of variables, using e.g. p-

constant characters, and in many situations it is useful to reduce the number of inequalities, i.e. of used characters.

To measure the time a function needs we use `IO_gettimeofday` from the IO-package rather than functions like `time` or `Runtime`, since these measure only the GAP time, but do not return the time the functions spend using 4ti2 or normaliz. We used the following function (which is essentially due to Olexandr Konovalov) to meassure the time used for the computation:

Example
```
TimeFunction := function(f, args)
 # input: the function of which the computing time should be measured
 #        and the list of arguments for this function
 # output: time needed for the calculations in seconds
 local start;
 start := IO_gettimeofday();
 CallFuncList(f,args);
 return IO_gettimeofday().tv_sec - start.tv_sec;
 end;
```

All times will be given in seconds. The computations were perfomed on an a machine with four 2,6 GHz kernels.

A lot of time might be saved by testing only a few characters instead of a whole character table:

Example
```
gap> C := CharacterTable("L2(49)");;
gap> HeLP_Solver("normaliz");
'normaliz' will be used from now on.
gap> TimeFunction(HeLP_WithGivenOrder, [C,35]);
#I  Number of solutions for elements of order 35: 0; stored in HeLP_sol[35].
6  # I.e.: The computation took 6 seconds.
gap> TimeFunction(HeLP_WithGivenOrder, [Irr(C){[2]}, 35]);
#I  Number of solutions for elements of order 35: 0; stored in HeLP_sol[35].
0
gap> HeLP_Solver("4ti2");
'4ti2' will be used from now on.
gap> TimeFunction(HeLP_WithGivenOrder, [C,35]);
#I  Number of solutions for elements of order 35: 0; stored in HeLP_sol[35].
6
gap> TimeFunction(HeLP_WithGivenOrder, [Irr(C){[2]}, 35]);
#I  Number of solutions for elements of order 35: 0; stored in HeLP_sol[35].
1
```

I.e.: Using only one character instead of all of them is about six times faster in this situation and this is also quickly found by `HeLP_FindAndVerifySolution`.

Using only a few characters might even be a life saver:

Example
```
gap> C := CharacterTable("L4(3).2^2");;
gap> HeLP_WithGivenOrder(C, 3);;
#I  Number of solutions for elements of order 3: 63; stored in HeLP_sol[3].
gap> HeLP_WithGivenOrder(C, 13);;
#I  Number of solutions for elements of order 13: 198; stored in HeLP_sol[13].
gap> SetInfoLevel(HeLP_Info,4);
gap> HeLP_Solver("4ti2");
'4ti2' will be used from now on.
```

```
gap> HeLP_UseRedund(true);
'redund' will be used from now on.
gap> TimeFunction(HeLP_WithGivenOrder, [Irr(C){[5,11,16]}, 39]);
#I  Number of solutions for elements of order 39: 0; stored in HeLP_sol[39].
438
gap> HeLP_UseRedund(false);
The calculations will be performed without using 'redund' from now on.
gap> TimeFunction(HeLP_WithGivenOrder, [Irr(C){[5,11,16]}, 39]);
#I  Number of solutions for elements of order 39: 0; stored in HeLP_sol[39].
430
gap> HeLP_Solver("normaliz");
'normaliz' will be used from now on.
gap> TimeFunction(HeLP_WithGivenOrder, [Irr(C){[5,11,16]}, 39]);
#I  Number of solutions for elements of order 39: 0; stored in HeLP_sol[39].
340
gap> HeLP_UseRedund(true);
'redund' will be used from now on.
gap> TimeFunction(HeLP_WithGivenOrder, [Irr(C){[5,11,16]}, 39]);
#I  Number of solutions for elements of order 39: 0; stored in HeLP_sol[39].
419
gap> HeLP_UseRedund(false);
The calculations will be performed without using 'redund' from now on.
gap> HeLP_Solver("normaliz");
'normaliz' will be used from now on.
gap> TimeFunction(HeLP_WithGivenOrder, [Irr(C), 39]);
#I  Number of solutions for elements of order 39: 0; stored in HeLP_sol[39].
6234
```

Sometimes it is helpful to look at groups containing the group of interest:

Example

```
gap> C := CharacterTable("2F4(2)'");;
gap> HeLP_WithGivenOrder(C, 13);;
#I  Number of solutions for elements of order 13: 316; stored in HeLP_sol[13].
gap> HeLP_WithGivenOrder(C, 3);;
#I  Number of solutions for elements of order 3: 1; stored in HeLP_sol[3].
gap> TimeFunction(HeLP_WithGivenOrder, [C, 39]);
#I  Number of solutions for elements of order 39: 0; stored in HeLP_sol[39].
80
gap> C:=CharacterTable("2F4(2)'.2");
CharacterTable( "2F4(2)'.2" )
gap> TimeFunction(HeLP_WithGivenOrder, [C, 39]);
#I  Number of solutions for elements of order 39: 0; stored in HeLP_sol[39].
1
```

This is also a good example to use p-constant characters:

Example

```
gap> C:=CharacterTable("2F4(2)'");
CharacterTable( "2F4(2)'" )
gap> TimeFunction(HeLP_WithGivenOrderSConstant, [C, 13, 3]);
#I    Number of non-trivial 13-constant characters in the list: 19.
0
```

If using 4ti2, for some groups switching redund on and off gives major improvements.

Example
```
gap> HeLP_Solver("4ti2");
'4ti2' will be used from now on.
gap> HeLP_UseRedund(true);
'redund' will be used from now on.
gap> C := CharacterTable(SmallGroup(160,91));;
gap> TimeFunction(HeLP_ZC, [C]);
26
gap> HeLP_Solver("normaliz");
'normaliz' will be used from now on.
gap> TimeFunction(HeLP_ZC, [C]);
12
```

Using 4ti2 but not redund `HeLP_ZC(C)` ran for over 400 hours without a result.

Example
```
gap> C := CharacterTable(SmallGroup(96,12));;
gap> HeLP_UseRedund(false);
The calculations will be performed without using 'redund' from now on.
gap> HeLP_Solver("4ti2");;
gap> TimeFunction(HeLP_ZC, [C]);
2
```

Running this example using redund the computations does not proceed for elements of order 12.

4.4 Using InfoLevels

HeLP provides different InfoLevels for different situations. The variable controlling the InfoLevel is `HeLP_Info` and it might be changed using `SetInfoLevel(HeLP_Info, n)` to set the InfoLevel to n. The maximal `HeLP_Info` entry is 5, the default InfoLevel is 1. The examples below give some idea, how one can use `HeLP_Info`, but do not give complete information on all possibilities.

If one is only interested whether (ZC) or (PQ) can be solved using the HeLP method, one can set `HeLP_Info` to 0:

Example
```
gap> C := CharacterTable("M11");
CharacterTable( "M11" )
gap> HeLP_ZC(C);
#I  ZC can't be solved, using the given data, for the orders: [ 4, 6, 8 ].
false
gap> SetInfoLevel(HeLP_Info, 0);
gap> HeLP_ZC(C);
false
```

If the InfoLevel is set to 2, the functions `HeLP_ZC` (2.1.1) and `HeLP_PQ` (2.2.1) print information which order of torsion units is currently considered, so that the user can keep track of the progress. This may be used for bigger groups to see, if the calculations might finish at some point. Continuing the above example:

Example
```
gap> SetInfoLevel(HeLP_Info, 2);
gap> HeLP_PQ(C);
```

```
#I  Checking order 2.
#I  Checking order 3.
#I  Checking order 5.
#I  Checking order 10.
#I  Checking order 11.
#I  Checking order 15.
#I  Checking order 22.
#I  Checking order 33.
#I  Checking order 55.
true
```

HeLP_Info at InfoLevel 3 provides also some information about the used ordinary character table or Brauer tables:

Example
```
gap> SetInfoLevel(HeLP_Info, 3);
gap> HeLP_PQ(C);
#I  Checking order 2.
#I    Using table BrauerTable( "M11", 3 ).
#I  Checking order 3.
#I    Using table BrauerTable( "M11", 3 ).
#I    Using table BrauerTable( "M11", 11 ).
#I  Checking order 5.
#I    Using table BrauerTable( "M11", 3 ).
#I  Checking order 10.
#I    Using table BrauerTable( "M11", 3 ).
#I  Checking order 11.
#I    Using table BrauerTable( "M11", 3 ).
#I  Checking order 15.
#I    Using table BrauerTable( "M11", 3 ).
#I    Using table BrauerTable( "M11", 11 ).
#I  Checking order 22.
#I    Using table BrauerTable( "M11", 3 ).
#I  Checking order 33.
#I    Using table BrauerTable( "M11", 3 ).
#I    Using table BrauerTable( "M11", 11 ).
#I    Using table BrauerTable( "M11", 2 ).
#I  Checking order 55.
#I    Using table BrauerTable( "M11", 3 ).
true
```

Setting HeLP_Info to 4 is useful when there are many possibilities for the partial augmentations of the powers of some unit. A good example is the example on "L4(3).2^2" in the section on Time Saving 4.3, see above: If you see quickly that almost nothing is happening, you might want to change your strategy.

HeLP_Info at level 5 informs the user on all changes of the used character table. Using it makes sense, if you work with the command HeLP_ChangeCharKeepSols (3.4.1).

4.5 Non-standard characters

The package also allows using characters even if the whole character table is not available. E.g. induced characters:

Example
```
gap> C := CharacterTable("U3(8)");
CharacterTable( "U3(8)" )
gap> G := PSU(3,8);
<permutation group of size 5515776 with 2 generators>
gap> A := AutomorphismGroup(G);
<group of size 99283968 with 4 generators>
gap> AllCharacterTableNames(Size,Size(A));
[ "3.U3(8).6", "3.U3(8).S3" ]
```

This means: The character table of the automorphism group A of PSU(3,8) is not available in GAP. However one can use induced characters:

Example
```
gap> NN := NormalSubgroups(A);
[ <trivial group>, <group of size 5515776 with 2 generators>,
  <group with 3 generators>, <group of size 16547328 with 3 generators>,
  <group of size 49641984 with 4 generators>,
  <group of size 33094656 with 4 generators>,
  <group of size 99283968 with 4 generators> ]
gap> H := NN[2];      #Subgroup of A isomorphic to G
<group of size 5515776 with 2 generators>
gap> CharacterTableWithStoredGroup(H,C);
CharacterTable( <group of size 5515776 with 2 generators> )
gap> D := CharacterTable(H);
CharacterTable( <group of size 5515776 with 2 generators> )
gap> chi := InducedClassFunction(Irr(D)[2],A);
Character( CharacterTable( <group of size 99283968 with 4 generators> ),
 [ 1008, -144, -126, 18, 0, 0, 0, 0, 36, 36, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -18, 0, 0,
  0, 0 ] )
gap> HeLP_WithGivenOrder([chi],7*19);
#I  Number of solutions for elements of order 133: 0; stored in HeLP_sol[133].
[  ]
```

One can also use characters, which are not available in GAP, but are entered manually:

Example
```
gap> C := CharacterTable("L2(49)");
CharacterTable( "L2(49)" )
gap> HeLP_WithGivenOrder(C,15);;
#I  Number of solutions for elements of order 15: 56; stored in HeLP_sol[15].
gap> C7 := C mod 7;
fail
```

The Brauer characters for the prime 7 are well known, see e.g. [Sri64] , but are not yet available in GAP.

Example
```
gap> OrdersClassRepresentatives(C);
[ 1, 2, 3, 4, 5, 5, 6, 7, 7, 8, 8, 12, 12, 24, 24, 24, 24, 25, 25, 25, 25,
  25, 25, 25, 25, 25, 25 ]
gap> chi := ClassFunction(C, [ 3, 0, -1, 0, -E(5)^2-E(5)^3, -E(5)-E(5)^4, 0,
>  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]);
ClassFunction( CharacterTable( "L2(49)" ),
```

```
 [ 3, 0, -1, 0, -E(5)^2-E(5)^3, -E(5)-E(5)^4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ] )
gap> HeLP_WithGivenOrder([chi],15);
#I  Number of solutions for elements of order 15: 0; stored in HeLP_sol[15].
[ ]
```

The class function chi above is of course not a proper character of the group, but the values coincide with the values of a 7-Brauer character of the group on the conjugacy classes of order 1, 3 and 5, i.e. the one needed to use HeLP for order 15. All functions of the HeLP-package only access values of class functions on conjugacy classes of elements with an order dividing the order of the unit in question. That is why this class function chi can be used in this setting.

4.6 A complete example: (PQ) for the MacLaughlin simple group

This section gives a demonstration of many functions of the package. The goal is to verify the Prime Graph Question for the McLaughlin simple group, which was proved in [BK07b]

Example
```
gap> C := CharacterTable("McL");
CharacterTable( "McL" )
gap> SetInfoLevel(HeLP_Info,4);
```

The function HeLP_PQ(C) would take really long. Instead one can use HeLP_AllOrdersPQ(C) several times on a high InfoLevel. Any time you see the function needs long, just try some manual calculations. Compute first the partial augmentations of elements of prime order:

Example
```
gap> HeLP_WithGivenOrder(C,2);;
#I  Number of solutions for elements of order 2: 1; stored in HeLP_sol[2].
gap> HeLP_WithGivenOrder(C mod 2,3);;
#I  Number of solutions for elements of order 3: 4; stored in HeLP_sol[3].
gap> HeLP_WithGivenOrder(C mod 3,5);;
#I  Number of solutions for elements of order 5: 6; stored in HeLP_sol[5].
gap> HeLP_WithGivenOrder(C mod 3,7);;
#I  Number of solutions for elements of order 7: 174; stored in HeLP_sol[7].
gap> HeLP_WithGivenOrder(C mod 3,11);;
#I  Number of solutions for elements of order 11: 20; stored in HeLP_sol[11].
```

For mixed order in most situations p-constant characters are interesting. Check the tables for such characters of small degree.

Example
```
gap> HeLP_WithGivenOrderSConstant(Irr(C){[2,3,4,5]},7,3);
#I   Number of non-trivial 7-constant characters in the list: 4.
[ ]
gap> HeLP_WithGivenOrderSConstant(Irr(C){[2,3,4,5]},11,2);
#I   Number of non-trivial 11-constant characters in the list: 4.
[ ]
gap> HeLP_WithGivenOrderSConstant(Irr(C){[2,3,4,5]},11,3);
#I   Number of non-trivial 11-constant characters in the list: 4.
[ ]
gap> HeLP_WithGivenOrderSConstant(Irr(C mod 3){[2,3,4,5]},7,5);
```

```
#I    Number of non-trivial 7-constant characters in the list: 4.
[  ]
gap> HeLP_WithGivenOrderSConstant(Irr(C mod 3){[2,3,4,5]},7,11);
#I    Number of non-trivial 7-constant characters in the list: 4.
[  ]
gap> HeLP_WithGivenOrderSConstant(Irr(C mod 3){[2,3,4,5]},11,5);
#I    Number of non-trivial 11-constant characters in the list: 2.
[  ]
```

These calculations are enough to obtain an affirmative answer to the Prime Graph Question:

Example

```
gap> HeLP_AllOrdersPQ(C);
#I  Checking order 2.
#I    Using the known solutions for elements of order 2.
#I  Checking order 3.
#I    Using the known solutions for elements of order 3.
#I  Checking order 5.
#I    Using the known solutions for elements of order 5.
#I  Checking order 7.
#I    Using the known solutions for elements of order 7.
#I  Checking order 11.
#I    Using the known solutions for elements of order 11.
#I  Checking order 21.
#I    Using the known solutions for elements of order 21.
#I  Checking order 22.
#I    Using the known solutions for elements of order 22.
#I  Checking order 33.
#I    Using the known solutions for elements of order 33.
#I  Checking order 35.
#I    Using the known solutions for elements of order 35.
#I  Checking order 55.
#I    Using the known solutions for elements of order 55.
#I  Checking order 77.
#I    Using the known solutions for elements of order 77.
true
```

Checking these computations takes a few minutes.

Chapter 5

Background

In this chapter we give a brief overview of the Zassenhaus Conjecture and the Prime Graph Questions and the techniques used in this package. For a more detailed exposition see [BM18].

5.1 The Zassenhaus Conjecture and related questions

Let G be a finite group and let $\mathbb{Z}G$ denote its integral group ring. Let $\mathrm{V}(\mathbb{Z}G)$ be the group of units of augmentation one, aka. normalized units. An element of the unit group of $\mathbb{Z}G$ is called a torsion element, if it has finite order.

A conjecture of H.J. Zassenhaus asserted that every normalized torsion unit of $\mathbb{Z}G$ is conjugate within $\mathbb{Q}G$ ("rationally conjugate") to an element of G, see [Zas74] or [Seh93], Section 37. This is the first of his three famous conjectures about integral group rings and the only one which is was open when the first versions of this package appeared, hence it is referred to as the Zassenhaus Conjecture (ZC). This conjecture asserts that the torsion part of the units of $\mathbb{Z}G$ is as far determined by G as possible.

Negative solutions to the conjecture were finally found in [EM18].

Considering the difficulty of the Zassenhaus Conjecture W. Kimmerle raised the question, whether the Prime Graph of the normalized units of $\mathbb{Z}G$ coincides with that one of G (cf. [Kim07] Problem 21). This is the so called Prime Graph Question (PQ). The prime graph of a group is the loop-free, undirected graph having as vertices those primes p, for which there is an element of order p in the group. Two vertices p and q are joined by an edge, provided there is an element of order pq in the group. In the light of this description, the Prime Graph Question asks, whether there exists an element of order pq in G provided there exists an element of order pq in $\mathrm{V}(\mathbb{Z}G)$ for every pair of primes (p,q).

A question which lies between the Zassenhaus Conjecture and the Prime Graph Question is the Spectrum Problem. It asks, if the orders of elements in G and $\mathrm{V}(\mathbb{Z}G)$ coincide. In general, by a result of J. A. Cohn and D. Livingstone [CL65], Corollary 4.1, and a result of M. Hertweck [Her08a], the following is known about the possible orders of torsion units in integral group rings:

Theorem: The exponents of $\mathrm{V}(\mathbb{Z}G)$ and G coincide. Moreover, if G is solvable, any torsion unit in $\mathrm{V}(\mathbb{Z}G)$ has the same order as some element in G.

Finally, a question raised by W. Kimmerle in [Kim07] asks if any unit of finite order in $\mathrm{V}(\mathbb{Z}G)$ is conjugate in the rational group algebra $\mathbb{Q}H$ to a trivial unit, where H is a finite group containing G. We call this the Kimmerle Problem. This question did not receive much attention while the Zassenhaus Conjecture was still open. It can be shown however that the methods used in [EM18] to construct counterexamples to the Zassenhaus Conjecture can not yield negative solutions to the

Kimmerle Problem. In this sense it remains the strongest statement about torsion units in integral group rings of finite group which could still be true.

5.2 Partial augmentations and the structure of HeLP sol

For a finite group G and an element $x \in G$ let x^G denote the conjugacy class of x in G. The partial augmentation with respect to x or rather the conjugacy class of x is the map ε_x sending an element u to the sum of the coefficients at elements of the conjugacy class of x, i.e.

$$\varepsilon_x\colon \mathbb{Z}G \to \mathbb{Z}, \quad \sum_{g \in G} z_g g \mapsto \sum_{g \in x^G} z_g.$$

Let u be a torsion element in $\mathrm{V}(\mathbb{Z}G)$. By results of G. Higman, S.D. Berman and M. Hertweck the following is known for the partial augmentations of u:

Theorem: ([Seh93], Proposition (1.4); [Her07], Theorem 2.3) $\varepsilon_1(u) = 0$ if $u \neq 1$ and $\varepsilon_x(u) = 0$ if the order of x does not divides the order of u.

Partial augmentations are connected to (ZC) and (PQ) via the following result, which is due to Z. Marciniak, J. Ritter, S. Sehgal and A. Weiss [MRSW87], Theorem 2.5:

Theorem: A torsion unit $u \in \mathrm{V}(\mathbb{Z}G)$ of order k is rationally conjugate to an element of G if and only if all partial augmentations of u^d vanish, except one (which then is necessarily 1) for all divisors d of k.

The last statement also explains the structure of the variable `HeLP_sol`. In `HeLP_sol[k]` the possible partial augmentations for an element of order k and all powers u^d for d dividing k (except for $d = k$) are stored, sorted ascending w.r.t. order of the element u^d. For instance, for $k = 12$ an entry of `HeLP_sol[12]` might be of the following form:

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

The first sublist `[ 1 ]` indicates that the element u^6 of order 2 has the partial augmentation 1 at the only class of elements of order 2, the second sublist `[ 0, 1 ]` indicates that u^4 of order 3 has partial augmentation 0 at the first class of elements of order 3 and 1 at the second class. The third sublist `[ -2, 2, 1 ]` states that the element u^3 of order 4 has partial augmentation -2 at the class of elements of order 2 while 2 and 1 are the partial augmentations at the two classes of elements of order 4 respectively, and so on. Note that this format provides all necessary information on the partial augmentations of u and its powers by the above restrictions on the partial augmentations.

From version 4 onwards this package incorporates more theoretical restrictions on partial augmentations. More precisely, it uses more results about vanishing partial augmentations of normalized torsion units. One is the more general form of the Berman-Higman theorem, namely that if z is a central element in G and $u \in \mathrm{V}(\mathbb{Z}G)$ is a torsion unit different from z, then $\varepsilon_z(u) = 0$. Moreover, two more elaborate criteria derived from the work of Hertweck are used:

Theorem:([Her08a], Proposition 2; [Her08b], Lemma 2.2; [Mar17]) Let $u \in \mathrm{V}(\mathbb{Z}G)$ be of finite order and $\varepsilon_g(u) \neq 0$ for some $g \in G$. Suppose that u has smaller order modulo some normal p-subgroup N of G. Then the p-part of g has the same order as the p-part of u. Furthermore, if the p-part of u is p-adically conjugate to an element in G, then the p-part of g is even conjugate in G to the p-part of u. Such a p-adic conjugation holds, if the order of u modulo a normal p-subgroup of G is not divisible by p, i.e. the p-part of u is trivial modulo a normal p-subgroup.

To apply this theorem, some knowledge on the normal subgroups of G is necessary. Hence it is only applied in the package when the character table one works with possesses an underlying group.

It is clear the Prime Graph Question or Spectrum Problem can be studied using the HeLP-method (if no possible partial augmentations exist for a given order neither does a unit of that order) and the possibility to do this for the Zassenhaus Conjecture is given via the above theorem of Marciniak-Ritter-Sehgal-Weiss. For the Kimmerle Problem a somehow similar result states that a unit $u \in \mathrm{V}(\mathbb{Z}G)$ of order k is conjugate in $\mathbb{Q}H$, for H some group containing G, to a trivial unit if and only if the sum of the coefficients of u at elements of order k equals 1 and the sum of coefficients of elements of order m equals 0 for any $m \neq k$ [MdR19], Proposition 2.1. This shows that the Kimmerle Problem is in fact equvivalent to an earlier question of A. Bovdi and hence results on Bovdi's Problem can also be applied.

For more details on when the variable `HeLP_sol` is modified or reset and how to influence this behavior see Section 4.2 and `HeLP_ChangeCharKeepSols` (3.4.1).

5.3 The HeLP equations

Denote by x^G the conjugacy class of an element x in G. Let u be a torsion unit in $\mathrm{V}(\mathbb{Z}G)$ of order k and D an ordinary representation of G over a field contained in $\mathbb{C}$ with character χ. Then $D(u)$ is a matrix of finite order and thus diagonalizable over $\mathbb{C}$. Let ζ be a primitive k-th root of unity, then the multiplicity $\mu_l(u,\chi)$ of ζ^l as an eigenvalue of $D(u)$ can be computed via Fourier inversion and equals

$$\mu_l(u,\chi) = \frac{1}{k}\sum_{1\neq d|k} \mathrm{Tr}_{\mathbb{Q}(\zeta^d)/\mathbb{Q}}(\chi(u^d)\zeta^{-dl}) + \frac{1}{k}\sum_{x^G} \varepsilon_x(u)\mathrm{Tr}_{\mathbb{Q}(\zeta)/\mathbb{Q}}(\chi(x)\zeta^{-l}).$$

As this multiplicity is a non-negative integer, we have the constraints

$$\mu_l(u,\chi) \in \mathbb{Z}_{\geq 0}$$

for all ordinary characters χ and all l. This formula was given by I.S. Luthar and I.B.S. Passi [LP89].

Later M. Hertweck showed that it may also be used for a representation over a field of characteristic $p > 0$ with Brauer character φ, if p is coprime to k [Her07], § 4. In that case one has to ignore the p-singular conjugacy classes (i.e. the classes of elements with an order divisible by p) and the above formula becomes

$$\mu_l(u,\varphi) = \frac{1}{k}\sum_{1\neq d|k} \mathrm{Tr}_{\mathbb{Q}(\zeta^d)/\mathbb{Q}}(\varphi(u^d)\zeta^{-dl}) + \frac{1}{k}\sum_{x^G,\ p\nmid o(x)} \varepsilon_x(u)\mathrm{Tr}_{\mathbb{Q}(\zeta)/\mathbb{Q}}(\varphi(x)\zeta^{-l}).$$

Again, as this multiplicity is a non-negative integer, we have the constraints

$$\mu_l(u,\varphi) \in \mathbb{Z}_{\geq 0}$$

for all Brauer characters φ and all l.

These equations allow to build a system of integral inequalities for the partial augmentations of u. Solving these inequalities is exactly what the HeLP method does to obtain restrictions on the possible values of the partial augmentations of u. Note that some of the $\varepsilon_x(u)$ are a priori zero by the results in the above sections.

For p-solvable groups representations over fields of characteristic p can not give any new information compared to ordinary representations by the Fong-Swan-Rukolaine Theorem [CR90], Theorem 22.1.

5.4 The Wagner test

We also included a result motivated by a theorem R. Wagner proved 1995 in his Diplomarbeit [Wag95]. This result gives a further restriction on the partial augmentations of torsion units. Though the results was actually available before Wagner's work, cf. [BH08] Remark 6, we named the test after him, since he was the first to use the HeLP-method on a computer. We included it into the functions `HeLP_ZC` (2.1.1), `HeLP_PQ` (2.2.1), `HeLP_SP` (2.3.1), `HeLP_KP` (2.4.1) `HeLP_AllOrders` (3.3.1), `HeLP_AllOrdersPQ` (3.3.2) and `HeLP_WagnerTest` (3.7.1) and call it "Wagner test".

Theorem: For a torsion unit $u \in \mathrm{V}(\mathbb{Z}G)$, a group element s, a prime p and a natural number j we have

$$\sum_{x^{p^j} \sim s} \varepsilon_x(u) \equiv \varepsilon_s(u^{p^j}) \mod p.$$

Combining the Theorem with the HeLP-method may only give new insight, if p^j is a proper divisor of the order of u. Wagner did obtain this result for $s = 1$, when $\varepsilon_s(u) = 0$ by the Berman-Higman Theorem. In the case that u is of prime power order this is a result of J.A. Cohn and D. Livingstone [CL65].

5.5 s-constant characters

If one is interested in units of mixed order $s * t$ for two primes s and t (e.g. if one studies the Prime Graph Question) an idea to make the HeLP method more efficient was introduced by V. Bovdi and O. Konovalov in [BK10], page 4. Assume one has several conjugacy classes of elements of order s, and a character taking the same value on all of these classes. Then the coefficient of every of these conjugacy classes in the system of inequalities of this character, which is obtained via the HeLP method, is the same. Also the constant terms of the inequalities do not depend on the partial augmentations of elements of order s. Thus for such characters one can reduce the number of variables in the inequalities by replacing all the partial augmentations on classes of elements of order s by their sum. To obtain the formulas for the multiplicities of the HeLP method one does not need the partial augmentations of elements of order s. Characters having the above property are called s-constant. In this way the existence of elements of order $s * t$ can be excluded in a quite efficient way without doing calculations for elements of order s.

There is also the concept of (s,t)-constant characters, being constant on both, the conjugacy classes of elements of order s and on the conjugacy classes of elements of order t. The implementation of this is however not yet part of this package.

5.6 Known results about the Zassenhaus Conjecture and the Prime Graph Question

At the moment as this documentation was written, to the best of our knowledge, the following results were available for the Zassenhaus Conjecture and the Prime Graph Question:

For the Zassenhaus Conjecture only the following reduction is available:

Theorem: Assume the Zassenhaus Conjecture holds for a group G. Then (ZC) holds for $G \times C_2$ [HK06], Corollary 3.3, and $G \times \Pi$, where Π denotes a nilpotent group of order prime to the order of G [Her08b], Proposition 8.1.

It is also known to go over to other types of direct products under certain conditions [BKS20]. With this reductions in mind the Zassenhaus Conjecture is known for:

- Nilpotent groups [Wei91],
- Cyclic-By-Abelian groups [CMdR13] and some other special cyclic-by-nilpotent groups [CdR20],
- Groups containing a normal Sylow subgroup with abelian complement [Her06],
- Frobenius groups whose order is divisible by at most two different primes [JPM00],
- Groups $X \rtimes A$, where X and A are abelian and A is of prime order p such that p is smaller then any prime divisor of the order of X [MRSW87],
- All groups of order up to 143 [BHK+18],
- The non-abelian simple groups A_5 [LP89], $A_6 \simeq PSL(2,9)$ [Her08c], $PSL(2,7)$, $PSL(2,11)$, $PSL(2,13)$ [Her07], $PSL(2,8)$, $PSL(2,17)$ [KK15] [Gil13], $PSL(2,19)$, $PSL(2,23)$ [BM17b], $PSL(2,25)$, $PSL(2,31)$, $PSL(2,32)$ [BM19b] and some extensions of these groups. Also for all $PSL(2,p)$ where p is a fermat or a Mersenne prime [MdRS19], and $PSL(2,p)$ and $PSL(2,p^2)$ if $p\pm 1$ or $p^2 \pm 1$ is 4 multiplied by a prime [EM22],
- For special linear groups $SL(2,p)$ and $SL(2,p^2)$ for p a prime [dRS19].

The only known counterexamples to the conjecture are exhibited in [EM18].

For the Prime Graph Question the following strong reduction was obtained in [KK15]:

Theorem: Assume the Prime Graph Question holds for all almost simple images of a group G. Then (PQ) also holds for G.

Here a group G is called almost simple, if it is sandwiched between the inner automorphism group and the whole automorphism group of a non-abelian simple group S. I.e. $Inn(S) \leq G \leq Aut(S)$. Keeping this reduction in mind (PQ) is known for:

- Solvable groups [Kim06],
- All but two of the sporadic simple groups and their automorphism groups [CM21], the exceptions being the Monster and the O'Nan group; for an overview of early HeLP-results see [KK15],
- Groups whose socle is isomorphic to a group $PSL(2,p)$ or $PSL(2,p^2)$, where p denotes a prime, [Her07], [BM17a].
- Groups whose socle is isomorphic to an alternating group, [Sal11] [Sal13][BC17][BM19a],
- Almost simple groups whose order is divisible by at most three different primes [KK15] and [BM17b]. (This implies that it holds for all groups with an order divisible by at most three primes, using the reduction result above.)
- Many almost simple groups whose order is divisible by four different primes [BM17a][BM19b],
- Certain infinite series of simple groups of Lie type of small rank and other groups from the character table library [CM21]

Chapter 6

Remarks on technical problems and the implementation

6.1 Making the HeLP-package run

For all basic functionalities of the HeLP-package (using only the solver normaliz) the standard GAP-installation should suffice to make everything work: Get the most recent GAP from the GAP-webpage by following the instructions on the Download-page. Make sure to also run `InstPackages.sh` as explained there. This should install all packages needed to run HeLP. Just start GAP and type `LoadPackage("HeLP");`. In GAP 4.8.2 the NormalizInterface has to be updated to version 0.9.6 or newer which can be obtained from the website of the package. For GAP 4.8.3 or newer this should not be necessary anymore.

Here is a checklist what to do, if the package does not work or you also want to use the solver 4ti2:

- Make sure you have sufficiently new versions of the following software:
  - GAP (at least 4.8.2)
  - the GAP-package CTblLib (at least 1.2.2)
  - the GAP-package IO (at least 4.2; see also the next bullet point if this package can not be loaded)
  - the GAP-package 4ti2Interface (at least 2015.04.29; this package needs the IO-package)
  - the GAP-package NormalizInterface (at least 0.9.6)

Usually all these packages should come with a sufficiently recent GAP-installation (4.8.2 or newer) and should be contained in the pkg-folder of the GAP-installation. To see if they are working you can load them by typing `LoadPackage("[name]");` after starting GAP, where `[name]` is the name of the package.

- The IO-package needs a C-part to be compiled. To see if this has already been done on your system, you can enter `LoadPackage("IO");` after starting GAP. If the result is `fail` and the package is contained in the pkg-folder, than most likely the C-part is not yet compiled. For information on installation and in particular on how to compile the C-part, see the manual (in particular Chapter 2) or the README-file of that package.

- The installation of normaliz is possible via the GAP-package NormalizInterface (at least 0.9.6). Just access the folder in a terminal and do `./build-normaliz.sh; ./configure; make`.

- If you want to use 4ti2, please make sure that www.4ti2.de (Version 1.6.5 or newer) is properly installed. In case of an error-message "The executable 'zsolve' provided by the software 4ti2 was not found." after typing `LoadPackage("HeLP");` either the software is not properly installed or installed in a directory where GAP can not find it, i.e. a directory not contained in the path-variable. The content of this variable can typically be displayed by typing `echo $PATH` (Linux, Mac) `echo %PATH%` (Windows) in a terminal or a command prompt. The manual of 4ti2 contains several pages of information on how to install the program. Note that the installation of 4ti2 requires gcc (g++) and gmp installed (which come with many Linux installations or can be installed using a package manager). Make sure to execute all four commands indicated in the 4ti2 manual (possibly without the `-prefix=`-part):

  ```
  ./configure -prefix=INSTALLATION-DIRECTORY
  make
  make check
  make install-exec
  ```

  Depending on the settings of your system you might need root privileges (type `sudo` in front of every command) to unpack the files and install them. To check whether the installation worked, you can enter `zsolve` in a terminal. In case one of the required programs (g++ or gmp) was not installed when running `make` for the first time, you might need to run `make clean` and the above commands afterwards again (several times) to compile 4ti2 successfully. If you already have 4ti2 installed in a directory not contained in the path-variable and want to avoid a re-installation, in many cases the following helps:

  – Start a terminal and access a path written in your bash or system_bash. Typically `usr/local/bin` should work.
  – Run `ln -s /[PathToZsolve] zsolve`, where `[PathToZsolve]` is the path to the executable zsolve. This sets a symlink to the right place. E.g. `ln -s /opt/4ti2/bin/zsolve zsolve` was used on the (Linux) computers in Stuttgart.

- In case you use 4ti2, we also recommend to install lrslib, at least version 4.3 (note that version 4.2 or older sometimes produces unwanted behavior). This software provides the 'redund' command, which can be switched on and off within HeLP, but which often leads to better performances (cf. `HeLP_UseRedund` (3.5.2)). For installation see the User's Guide or the Readme-file on the above mentioned homepage. Usually, after unpacking in a directory contained in the path-variable it should be enough to call

  ```
  make all
  ```

  (possibly as root) inside the lrslib-directory.

- If this does not help to get HeLP running, please feel more than welcome to contact one of the maintainers of the package.

6.2 How much 4ti2 and normaliz is really there?

The reason, why the programs 4ti2 and normaliz are used in this package, is basically that they can solve systems of linear inequalities efficiently and there exist good GAP-Interfaces for them. However there is only one line of code where a function is called which accesses 4ti2 and a few more for normaliz. Thus the effort of using another solver of inequalities would be not so big, if there is a GAP-Interface for it. If you are aware of such a solver and would like to use it in this package, please contact the authors of this package. We will be happy to help.

References

[Avi] David Avis. lrslib – reverse search vertex enumeration program. Available at http://cgm.cs.mcgill.ca/~avis/C/lrs.html. 5

[BC17] A. Bächle and M. Caicedo. On the prime graph question for almost simple groups with an alternating socle. *Internat. J. Algebra Comput.*, 27(3):333–347, 2017. 47

[BH08] V. A. Bovdi and M. Hertweck. Zassenhaus conjecture for central extensions of S_5. *J. Group Theory*, 11(1):63–74, 2008. 46

[BHK$^+$18] Andreas Bächle, Allen Herman, Alexander Konovalov, Leo Margolis, and Gurmail Singh. The status of the Zassenhaus conjecture for small groups. *Exp. Math.*, 27(4):431–436, 2018. 9, 31, 47

[BIR$^+$] W. Bruns, B. Ichim, T. Römer, R. Sieg, and C. Söger. Normaliz. algorithms for rational cones and affine monoids. Available at http://normaliz.uos.de. 5

[BJK11] V. A. Bovdi, E. Jespers, and A. B. Konovalov. Torsion units in integral group rings of Janko simple groups. *Math. Comp.*, 80(273):593–615, 2011. 12, 16

[BK07a] V. A. Bovdi and A. B. Konovalov. Integral group ring of the first Mathieu simple group. In *Groups St. Andrews 2005. Vol. 1*, volume 339 of *London Math. Soc. Lecture Note Ser.*, pages 237–245. Cambridge Univ. Press, Cambridge, 2007. 9

[BK07b] V. A. Bovdi and A. B. Konovalov. Integral group ring of the McLaughlin simple group. *Algebra Discrete Math.*, (2):43–53, 2007. 41

[BK10] V. A. Bovdi and A. B. Konovalov. Torsion units in integral group ring of Higman-Sims simple group. *Studia Sci. Math. Hungar.*, 47(1):1–11, 2010. 46

[BKL08] V. A. Bovdi, A. B. Konovalov, and S. Linton. Torsion units in integral group ring of the Mathieu simple group M_{22}. *LMS J. Comput. Math.*, 11:28–39, 2008. 28

[BKS20] Andreas Bächle, Wolfgang Kimmerle, and Mariano Serrano. On the first Zassenhaus conjecture and direct products. *Canad. J. Math.*, 72(3):602–624, 2020. 47

[BM17a] A. Bächle and L. Margolis. On the prime graph question for integral group rings of 4-primary groups I. *Internat. J. Algebra Comput.*, 27(6):731–767, 2017. 11, 47

[BM17b] A. Bächle and L. Margolis. Rational conjugacy of torsion units in integral group rings of non-solvable groups. *Proc. Edinb. Math. Soc. (2)*, 60(4):813–830, 2017. 10, 47

[BM18] A. Bächle and L. Margolis. HeLP: a GAP package for torsion units in integral group rings. *J. Softw. Algebra Geom.*, 8:1–9, 2018. 5, 43

[BM19a] Andreas Bächle and Leo Margolis. An application of blocks to torsion units in group rings. *Proc. Amer. Math. Soc.*, 147(10):4221–4231, 2019. 47

[BM19b] Andreas Bächle and Leo Margolis. On the prime graph question for integral group rings of 4-primary groups II. *Algebr. Represent. Theory*, 22(2):437–457, 2019. 47

[BM21] Andreas Bächle and Leo Margolis. From examples to methods: two cases from the study of units in integral group rings. *Indian J. Pure Appl. Math.*, 52(3):669–686, 2021. 12, 13, 16

[CdR20] Mauricio Caicedo and Ángel del Río. On the Zassenhaus conjecture for certain cyclic-by-nilpotent groups. *Mediterr. J. Math.*, 17(2):Paper No. 62, 17, 2020. 47

[CL65] J. A. Cohn and D. Livingstone. On the structure of group algebras. I. *Canad. J. Math.*, 17:583–593, 1965. 43, 46

[CM21] Mauricio Caicedo and Leo Margolis. Orders of units in integral group rings and blocks of defect 1. *J. Lond. Math. Soc. (2)*, 103(4):1515–1546, 2021. 47

[CMdR13] M. Caicedo, L. Margolis, and Á. del Río. Zassenhaus conjecture for cyclic-by-abelian groups. *J. Lond. Math. Soc. (2)*, 88(1):65–78, 2013. 47

[CR90] C. W. Curtis and I. Reiner. *Methods of representation theory. Vol. I.* Wiley Classics Library. John Wiley & Sons, Inc., New York, 1990. With applications to finite groups and orders, Reprint of the 1981 original, A Wiley-Interscience Publication. 45

[dRS19] Ángel del Río and Mariano Serrano. Zassenhaus conjecture on torsion units holds for $\mathrm{SL}(2,p)$ and $\mathrm{SL}(2,p^2)$. *J. Group Theory*, 22(5):953–974, 2019. 47

[EM18] Florian Eisele and Leo Margolis. A counterexample to the first Zassenhaus conjecture. *Adv. Math.*, 339:599–641, 2018. 43, 47

[EM22] F. Eisele and L. Margolis. Units in blocks of defect 1 and the zassenhaus conjecture. *preprint, arxiv.org/abs/arXiv:2212.06634*, page 29 pages, 2022. 47

[Gil13] J. Gildea. Zassenhaus conjecture for integral group ring of simple linear groups. *J. Algebra Appl.*, 12(6):1350016, 10, 2013. 47

[Her06] M. Hertweck. On the torsion units of some integral group rings. *Algebra Colloq.*, 13(2):329–348, 2006. 47

[Her07] M. Hertweck. Partial augmentations and Brauer character values of torion units in group rings. *Preprint*, 2007. e-print `arXiv:math.RA/0612429v2[math.RA]`. 5, 8, 11, 27, 44, 45, 47

[Her08a] M. Hertweck. The orders of torsion units in integral group rings of finite solvable groups. *Comm. Algebra*, 36(10):3585–3588, 2008. 43, 44

[Her08b] M. Hertweck. Torsion units in integral group rings of certain metabelian groups. *Proc. Edinb. Math. Soc. (2)*, 51(2):363–385, 2008. 44, 46

[Her08c] M. Hertweck. Zassenhaus conjecture for A_6. *Proc. Indian Acad. Sci. Math. Sci.*, 118(2):189–195, 2008. 8, 11, 31, 47

[HK06] C. Höfert and W. Kimmerle. On torsion units of integral group rings of groups of small order. In *Groups, rings and group rings*, volume 248 of *Lect. Notes Pure Appl. Math.*, pages 243–252. Chapman & Hall/CRC, Boca Raton, FL, 2006. 8, 46

[JPM00] S. O. Juriaans and C. Polcino Milies. Units of integral group rings of Frobenius groups. *J. Group Theory*, 3(3):277–284, 2000. 47

[Kim06] W. Kimmerle. On the prime graph of the unit group of integral group rings of finite groups. In *Groups, rings and algebras*, volume 420 of *Contemp. Math.*, pages 215–228. Amer. Math. Soc., Providence, RI, 2006. 47

[Kim07] W. Kimmerle. Mini-Workshop: Arithmetik von Gruppenringen. *Oberwolfach Reports*, 4(4):3209–3239, 2007. 43

[KK15] W. Kimmerle and A.B. Konovalov. Recent advances on torsion subgroups of Integral Group Rings. *Proc. of Groups St Andrews 2013*, pages 331–347, 2015. 47

[LP89] I.S. Luthar and I.B.S. Passi. Zassenhaus conjecture for A_5. *Proc. Indian Acad. Sci. Math. Sci.*, 99(1):1–5, 1989. 5, 7, 45, 47

[Mar17] L. Margolis. A Theorem of Hertweck on p-adic conjugacy. *arxiv.org/abs/1706.02117*, page 11 pages, 2017. 31, 44

[MdR19] Leo Margolis and Ángel del Río. Partial augmentations power property: a Zassenhaus conjecture related problem. *J. Pure Appl. Algebra*, 223(9):4089–4101, 2019. 45

[MdRS19] Leo Margolis, Ángel del Río, and Mariano Serrano. Zassenhaus conjecture on torsion units holds for $\mathrm{PSL}(2, p)$ with p a Fermat or Mersenne prime. *J. Algebra*, 531:320–335, 2019. 47

[MRSW87] Z. Marciniak, J. Ritter, S. Sehgal, and A. Weiss. Torsion units in integral group rings of some metabelian groups. II. *Journal of Number Theory*, 25(3):340–352, 1987. 44, 47

[Sal11] M. Salim. Kimmerle's conjecture for integral group rings of some alternating groups. *Acta Math. Acad. Paedagog. Nyházi. (N.S.)*, 27(1):9–22, 2011. 10, 47

[Sal13] M. Salim. The prime graph conjecture for integral group rings of some alternating groups. *Int. J. Group Theory*, 2(1):175–185, 2013. 47

[Seh93] S.K. Sehgal. *Units in integral group rings*, volume 69 of *Pitman Monographs and Surveys in Pure and Applied Mathematics*. Longman Scientific & Technical, Harlow, 1993. 43, 44

[Sri64] B. Srinivasan. On the modular characters of the special linear group $SL(2, p^n)$. *Proc. London Math. Soc. (3)*, 14:101–114, 1964. 40

[tt] 4 ti 2 team. 4ti2—a software package for algebraic, geometric and combinatorial problems on linear spaces. Available at `www.4ti2.de`. 5

[Wag95] R. Wagner. Zassenhausvermutung über die Gruppen $\mathrm{PSL}(2, p)$. Diplomarbeit Universität Stuttgart, Mai 1995. 46

[Wei91] A. Weiss. Torsion units in integral group rings. *J. Reine Angew. Math.*, 415:175–187, 1991. 47

[Zas74] H. Zassenhaus. On the torsion units of group rings. *Estudos de Mathemátics em homenagem ao Prof. A. Almeida Costa, Instituto de Alta Cultura (Portugese)*, pages 119–126, 1974. 43

Index