SymbCompCC

A GAP4 Package

Version 1.3.1

by

Dörte Feichtenschlager

Institut Computational Mathematics, TU Brausnchweig

Pockelsstr. 14, 38106 Braunschweig, Germany

email: d.feichtenschlager@tu-braunschweig.de

September 2019

Contents

1 Installing and Loading the SymbCompCC Package **3**

1.1 Installing the SymbCompCC Package . . . 3

1.2 Loading the SymbCompCC Package . . . 3

2 Introduction **4**

2.1 Overview . . . 4

2.2 Background on (polycyclic) parametrised presentations . . . 4

2.3 Computation of Schur multiplicators . . . 5

2.4 Computation of low-dimensional cohomology . . . 5

2.5 Example . . . 5

3 p-power-poly-pcp-groups **6**

3.1 Example . . . 6

3.2 Obtaining p-power-poly-pcp-groups . . . 7

3.3 Operations and functions for p-power-poly-pcp-group elements . . . 8

3.4 Operations and functions for p-power-poly-pcp-groups . . . 8

3.5 Info classes for the p-power-poly-pcp-groups . . . 9

3.6 Global variables for the p-power-poly-pcp-groups . . . 10

4 Parametrised Presentations **11**

4.1 Provided pp-presentations . . . 11

5 Schur extensions for p-power-poly-pcp-groups **12**

5.1 Computing Schur extensions . . . 13

5.2 Computing other invariants from Schur extensions . . . 13

5.3 Info classes for the computation of the Schur extension . . . 14

Bibliography **15**

1 Installing and Loading the SymbCompCC Package

1.1 Installing the SymbCompCC Package

The following installation instruction is for unix although the package should work as well with any other operating system.

To install the SymbCompCC package, unpack the archive file, which should have a name of the form `SymbCompCC-`*XXX*`.tar.gz` for some version number *XXX*, by typing

```
bunzip2 SymbCompCC-XXX.tar.gz
tar -xvf SymbCompCC-XXX.tar
```

in the `pkg` directory of your version of GAP 4, or in a directory named `pkg` (e.g. in your home directory). (The only essential difference with installing SymbCompCC in a `pkg` directory different to the GAP 4 home directory is that one must start GAP with the `-l` switch, e.g. if your private `pkg` directory is a subdirectory of `mygap` in your home directory you might type:

```
gap -l ";myhomedir/mygap"
```

where *myhomedir* is the path to your home directory, which may be replaced by a tilde. The empty path before the semicolon is filled in by the default path of the GAP 4 home directory.)

1.2 Loading the SymbCompCC Package

To use the SymbCompCC Package you have to request it explicitly. This is done by calling

```
gap> LoadPackage("SymbCompCC");
true
```

The `LoadPackage` command is described in Section 76.2.1 in the GAP Reference Manual.

If you want to load the SymbCompCC package by default, you can put the `LoadPackage` command into your `.gaprc` file (see Section “ref:the .gaprc file” in the GAP Reference Manual).

2 Introduction

2.1 Overview

The coclass of a finite p-group of order p^n and nilpotency class c is defined as $n-c$. This invariant of finite p-groups has been introduced by Leedham-Green and Newman in [LGN80] and it became of major importance in p-group theory.

A first tool in the classification of all p-groups of coclass r is the coclass graph $G(p,r)$. Its vertices are the isomorphism types of finite p-groups of coclass r. Two vertices G and H are joined by an edge if G is isomorphic to the quotient $H/\gamma(H)$ where $\gamma(H)$ is the last non-trivial term of the lower series of H.

Du Sautoy [dS00] and Eick and Leedham-Green [ELG08] proved that $G(p,r)$ contains certain periodic patterns. Eick and Leedham-Green [ELG08] define infinite coclass sequences of finite p-groups of coclass r which underpin this periodic pattern. In $G(2,r)$ and $G(3,1)$ almost all groups are contained in an infinite coclass sequence.

Eick and Leedham-Green [ELG08] also proved that the infinitely many p-groups in an infinite coclass sequence can be defined by a single parametrised presentation.

The first aim of this package is the definition of polycyclic parametrised presentations; these are parametrised presentations as defined by Eick and Leedham-Green [ELG08] and additionally they have various features of polycyclic presentations. Each such presentation defines all the infinitely many finite p-groups in an infinite coclass sequence.

We then provide some algorithms to compute with polycyclic parametrised presentations. In particular, we introduce a generalisation of the collection algorithm for polycyclic parametrised presentations. Based on this, we describe algorithms to compute polycyclic parametrised presentations for Schur extensions, for the Schur multiplicator and for some low-dimensional cohomology groups. We refer to [EF11] for details on the underlying algorithms and further references.

Finally, we exhibit a database of polycyclic parametrised presentations for the infinite coclass families of the finite 2-groups of coclass at most 2 and the finite 3-groups of coclass 1.

2.2 Background on (polycyclic) parametrised presentations

In this section we describe the polycyclic parametrised presentations (pp-presentations) for infinite coclass sequences.

Let $(G_x | x \in \mathbb{N})$, where $\mathbb{N}$ denotes the natural numbers, be an infinite coclass sequence; x is the parameter of this infinite coclass sequence. Then every group G_x is an extension of a finite p-group P of order p^n by an abelian p-group T_x of rank d. Furthermore, every G_x has a polycyclic presentation (short pp-presentation) on generators $g_1,\ldots,g_n,t_1,\ldots,t_d$ with relations of the form

$$\begin{aligned}
g_i^p &= g_{i+1}^{a_{i,i,i+1}} \cdots g_n^{a_{i,i,n}} t_1^{\alpha_{i,i,1}(x)} \cdots t_d^{\alpha_{i,i,d}(x)},\\
g_i^{g_j} &= g_{j+1}^{a_{i,j,j+1}} \cdots g_n^{a_{i,j,n}} t_1^{\alpha_{i,j,1}(x)} \cdots t_d^{\alpha_{i,j,d}(x)},\\
t_k^{g_i} &= t_1^{b_{k,i,1}(x)} \cdots t_d^{b_{k,i,d}(x)},\\
t_k^{t_l} &= t_k,\\
t_k^{p^{x+e}} &= 1,
\end{aligned}$$

where $1 \le j < i \le n$ and $1 \le k < l \le d$; certain $a_{i,j,m} \in \{0,\ldots,p-1\}$, a non-negative integer e, $\alpha_{k,l,m}(x)$ of the form $c_{k,l,m} + p^x d_{k,l,m}$ and $b_{k,l,m}$ with $b_{k,l,m}, c_{k,l,m}, d_{k,l,m}$ certain p-adic integers. The p-adic exponents arising in the

relations can be reduced modulo the relative orders of the involved elements and thus can be reduced to integers for every specific x.

We call such a pp-presentation *integral* if all the p-adic numbers $b_{k,l,m}, c_{k,l,m}, d_{k,l,m}$ are integers. Our algorithms introduced in this package compute with integral pp-presentations only.

We call such an pp-presentation *consistent* if for every $x \in \mathbb{N}$ the presentation is consistent as a polycyclic presentation; where we possibly reduce the exponents in the presentation modulo the relative orders of the generators.

2.3 Computation of Schur multiplicators

In this section we recall briefly the method of [EF11] to determine the Schur multiplicators of almost all groups G_x in an infinite coclass sequence.

Suppose we are given a consistent integral pp-presentation F/R_x for the groups G_x in an infinite coclass sequence, where F is a free group and R_x is generated by parametrised relations as above. Note that the exponents in these relations depend on x, while the number of generators and the number of relations does not depend on the parameter.

Using this presentation we can define a parametrised presentation for the Schur extensions $G_x^* = F/[F,R_x]$, corresponding to the parametrised presentation F/R_x. The next step is to find the isomorphism types of $Y_x = R_x/[F,R_x]$ since $M(G_x) \cong (F' \cap R_x)/[F,R_x]$ are the torsion subgroups of Y_x as all G_x are finite p-groups.

Then $Y_x = R_x/[F,R_x]$ are generated by certain so-called consistency relations. Using this we can compute the isomorphism types of Y_x and thus the isomorphism types of $M(G_x)$ for almost all G_x in the chosen infinite coclass sequence.

2.4 Computation of low-dimensional cohomology

From the parametrised presentation F/R_x we can see that the Abelian invariants are the same for all groups G_x in an infinite coclass sequence, and we can compute them. Using this and the computation of the Schur multiplicators one obtains $H^n(G_x,\mathbb{Z})$ and $H^n(G_x,GF(p))$ for $0 \le n \le 2$, where the G_x act trivially on $\mathbb{Z}$ and $GF(p)$, respectively.

2.5 Example

In this section we present the well-known example of quaternion groups $Q_{2^{x+3}}$. It is well known that they have a parametrised presentation of the following form:

$$\begin{aligned}\{g_1,g_2,t_1 |\, & g_1^2 = t_1^{2^x},\ g_2^{g_1} = g_2 t_1^{-1+2^{x+1}},\\ & g_2^2 = t_1,\ t_1^{g_1} = t_1^{-1+2^{x+1}},\\ & t_1^{2^{x+1}} = 1\}.\end{aligned}$$

Using this we can define the Schur extensions $Q^*_{2^{x+3}}$

$$\begin{aligned}\{g_1,g_2,t_1,c_1,c_2,c_3 | & g_1^2 = t_1^{2^x} c_3,\ g_2^{g_1} = g_2 t_1^{-1+2^{x+1}} c_2^{1-2^{x+1}},\\ & g_2^2 = t_1 c_1,\ t_1^{g_1} = t_1^{-1+2^{x+1}} c_2^{2-2^{x+1}},\\ & t_1^{2^{x+1}} = c_2^{2^{x+1}},\\ & c_1,c_2,c_3 central\}.\end{aligned}$$

This yields $M(Q_{2^{x+3}}) = 1$.

3 p-power-poly-pcp-groups

Eick and Leedham-Green [ELG08] defined for a prime p and a fixed coclass r infinite coclass sequences. These sequences consist of finite p-groups of coclass r. For each infinite coclass sequence there exists a consistent pp-presentation (see Section 2.2) such that if we choose a natural number for the parameter and possibly reduce the exponents modulo the relative orders, we obtain a consistent polycyclic presentation for a group in the sequence; and for each group in the sequence there exists a natural number such that using this as a value for the parameter, we obtain a polycyclic presentation for the group.

We use these consistent pp-presentations to compute parametrised groups, which we call p-power-poly-pcp-groups. Furthermore, methods for these are presented. Without specifying the parameter we compute certain properties and using the p-power-poly-pcp-groups we do this for all groups they represent at once.

The p-power-poly-pcp-groups have a consistent pp-presentation with generators $g_1, \ldots, g_n, t_1, \ldots t_d$ and $c_1, \ldots, c_m$, for some non-negative integers n, d and m, and relations of the form, where $rel[i,j]$ stores the right hand sides of the relations (see Section 2.2 for more information on pp-presentations),

$$\begin{aligned}
g_i^p &= rel[i,i],\\
t_i^{expo} &= rel[n+i, n+i],\\
c_i^{expo_vec[i]} &= rel[n+d+i, n+d+i],\\
g_i^{g_j} &= rel[j,i],\\
t_i^{g_j} &= rel[j, n+i],\\
t_i^{t_j} &= rel[n+j, n+i],
\end{aligned}$$

where the t_i's commute modulo $\langle c_1, \ldots, c_m \rangle$ and the c_i's are central. So rel (see Section 3.2) are the right hand sides of the relations, where some depend on the parameter. The relative orders $expo$ and $expo\_vec[i]$ of the generators t_j and c_i depend on the parameter.

3.1 Example

In this section we present the well-known example of quaternion groups $Q_{2^{x+3}}$. They have a pp-presentation of the following form:

$$\begin{aligned}
\{g_1, g_2, t_1 \mid & g_1^2 = t_1^{2^x},\ g_2^{g_1} = g_2 t_1^{-1+2^{x+1}},\\
& g_2^2 = t_1,\ t_1^{g_1} = t_1^{-1+2^{x+1}},\\
& t_1^{2^{x+1}} = 1\}.
\end{aligned}$$

3.2 Obtaining p-power-poly-pcp-groups

To obtain p-power-poly-pcp-groups:

1 ▶ `PPPPcpGroups(` *rel*, *n*, *d*, *m*, *expo*, *expo_vec*, *prime*, *cc*, *name* `)` F
▶ `PPPPcpGroups(` *rec* `)` F

returns the p-power-poly-pcp-groups described by the consistent pp-presentation with generators $g_1, \ldots, g_n, t_1, \ldots t_d$, $c_1, \ldots, c_m$, for some non-negative integers n, d and m, and relations of the form

$$\begin{aligned}
g_i^p &= rel[i,i],\\
t_i^{expo} &= rel[n+i, n+i],\\
c_i^{expo_vec[i]} &= rel[n+d+i, n+d+i],\\
g_i^{g_j} &= rel[j,i],\\
t_i^{g_j} &= rel[j, n+i],\\
t_i^{t_j} &= rel[n+j, n+i]\cdot
\end{aligned}$$

The input consists of the following:

rel
: is the list of the right hand sides of the relations, where each relation is presented by a list consisting of tuples; the first entry i of a tuple is the index of the generator (if $i \leq n$, then it represents generator g_i, if $n < i \leq d$, then it represents generator t_{i-n} and otherwise it represents generator c_{i-n-d}) and the second entry of the tuple is the corresponding exponent. Note that the exponents of the g_i's are saved as integers and all other exponents as lists, representing elements depending on the parameter.

n
: is the number of generators g_i,

d
: is the number of generators t_i,

m
: is the number of generators c_i,

expo
: is the relative order of all generators t_i; note that *expo* is a list that represents an element depending on the parameter,

expo_vec
: is the list of relative orders, where the ith entry of the list gives the relative order of the generator c_i; note that each relative order is a list that represents an element depending on the parameter,

prime
: is the underlying prime p,

cc
: if the p-power-poly-pcp-groups represent an infinite coclass sequence of p-groups of coclass r, then $cc = r$. If they represent Schur extensions of groups in an infinite coclass sequence, then *cc* is the coclass of the groups in this infinite coclass sequence.

name
: a string to name the p-power-poly-pcp-groups.

rec
: is a record of the form *rec(rel, expo, n, d, m, prime, cc, expo_vec, name)*.

The pp-presentation is described at the beginning of Chapter “symbcompcc:p-power-poly-pcp-group”. Note that the consistency of the presentation is checked and that the presentation has to be consistent.

```
gap> ParPresGlobalVar_2_1[1];
rec(
  rel := [ [ [ [ 1, 0 ] ] ], [ [ [ 2, 1 ], [ 3, -1+2*2^x ] ], [ [ 3, 1 ] ] ],
      [ [ [ 3, -1+2*2^x ] ], [ [ 3, 1 ] ], [ [ 3, 0 ] ] ] ], expo := 2*2^x,
  n := 2, d := 1, m := 0, prime := 2, cc := 1, expo_vec := [  ], name := "D" )
gap> G := PPPPcpGroups( ParPresGlobalVar_2_1[1] );
< P-Power-Poly-pcp-groups with 3 generators of relative orders [ 2,2,2*2^x ] >
```

2 ▶ `PPPPcpGroupsElement( G, word )` F

constructs an element in p-power-poly-pcp-groups, where G is a p-power-poly-pcp-group (thus representing an infinite coclass sequence through a pp-presentation) with generators $g_1,\ldots,g_n,t_1,\ldots,t_d,c_1,\ldots,c_m$ and *word* is a list of tuples, where the first entry i in the tuple gives the index of the generator (if $i \leq n$, then it represents generator g_i, if $n < i \leq d$, then it represents generator t_{i-n} and otherwise it represents generator c_{i-n-d}) and the second entry of the tuple is the corresponding exponent. Note that the exponents of the g_i's must be integers, while all other exponents can be integers or lists, representing an element depending on the parameter.

```
gap> G := PPPPcpGroups( ParPresGlobalVar_2_1[3] );
< P-Power-Poly-pcp-groups with 3 generators of relative orders [ 2,2,2*2^x ] >
gap> g1 := PPPPcpGroupsElement( G , [[1,1]] );
g1
gap> g := PPPPcpGroupsElement( G , [[1,1],[2,1],[3,1]] );
g1*g2*t1
gap> h := PPPPcpGroupsElement( G , [[1,1],[2,1],[3,G!.expo-1]] );
g1*g2*t1^(-1+2*2^x)
```

3.3 Operations and functions for p-power-poly-pcp-group elements

The typical operations for group elements can be carried out for p-power-poly-pcp-group elements, like `*`, `/`, Inverse, One, equality and ShallowCopy.

1 ▶ `CollectPPPPcp( a )` F

collects the p-power-poly-pcp-group element a so that after reducing to integers for every specific value for the parameter x, the element is collected in the polycyclic group, represented by x in the underlying pp-presentation.

Note that the global variable `COLLECT_PPOWERPOLY_PCP` determines whether every element will be collected immediately, when created, or not, see `COLLECT_PPOWERPOLY_PCP`, 3.6.1.

3.4 Operations and functions for p-power-poly-pcp-groups

For p-power-poly-pcp-groups:

1 ▶ `GeneratorsOfGroup( G )`

returns a set of generators for the p-power-poly-pcp-groups G.

2 ▶ `One( G )`

obtains the identity element of the p-power-poly-pcp-groups G.

3 ▶ `IsConsistentPPPPcp( G )` F
▶ `IsConsistentPPPPcp( ParPres )` F

checks if the underlying pp-presentation of the p-power-poly-pcp-groups G is consistent or if the pp-presenta-tion *ParPres* is consistent.

4 ► `GetPcGroupPPowerPoly( ` *ParPres*`, ` *n* ` )` F
► `GetPcGroupPPowerPoly( ` *G*`, ` *n* ` )` F

takes the pp-presentation given by the record *ParPres* as in `PPPPcpGroups`, 3.2.1 or the *p*-power-poly-pcp-groups *G* and takes *n*, a non-negative integer, as a value for the parameter to obtain a pc-presentation for the corresponding finite *p*-group.

5 ► `GetPcpGroupPPowerPoly( ` *ParPres*`, ` *n* ` )` F
► `GetPcpGroupPPowerPoly( ` *G*`, ` *n* ` )` F

takes pp-presentation given by the record *ParPres* as in `PPPPcpGroups`, 3.2.1 or the *p*-power-poly-pcp-groups *G* and takes *n*, a non-negative integer, as the parameter to obtain a pcp-presentation for the corresponding finite *p*-group, for further information we refer to the polycyclic package.

6 ► `GAPInputPPPPcpGroups( ` *file*`, ` *G* ` )` F
► `GAPInputPPPPcpGroups( ` *file*`, ` *ParPres* ` )` F

prints the *p*-power-poly-pcp-groups *G* defined by *ParPres* in the file *file* as a record that could be used as input to `PPPPcpGroups`, 3.2.1 to create *p*-power-poly-pcp-groups.

7 ► `GAPInputPPPPcpGroupsAppend( ` *file*`, ` *G* ` )` F
► `GAPInputPPPPcpGroupsAppend( ` *file*`, ` *ParPres* ` )` F

appends the pp-presentation of the *p*-power-poly-pcp-groups *G* defined by *ParPres* to the file *file* as a record that could be used as input to `PPPPcpGroups`, 3.2.1 to create *p*-power-poly-pcp-groups.

8 ► `LatexInputPPPPcpGroups( ` *file*`, ` *G* ` )` F
► `LatexInputPPPPcpGroups( ` *file*`, ` *ParPres* ` )` F

prints the pp-presentation of *G* as given by *ParPres* in latex-code to the file *file*. Note that only non-trivial relations are printed.

9 ► `LatexInputPPPPcpGroupsAppend( ` *file*`, ` *G* ` )` F
► `LatexInputPPPPcpGroupsAppend( ` *file*`, ` *ParPres* ` )` F

appends the pp-presentation of *G* as given by *ParPres* in latex-code to the file *file*. Note that only non-trivial relations are appended.

10 ► `LatexInputPPPPcpGroupsAllAppend( ` *file*`, ` *G* ` )` F
► `LatexInputPPPPcpGroupsAllAppend( ` *file*`, ` *ParPres* ` )` F

appends the pp-presentation of *G* as given by *ParPres* in latex-code to the file *file*. Note that all relations are appended.

3.5 Info classes for the p-power-poly-pcp-groups

The following info classes are available:

1 ► `InfoConsistencyPPPPcp` V

is an InfoClass with the following levels.

`level 1`
: displays the first consistency relation that fails during the consistency check;

`level 2`
: displays which family of consistency relations have been checked during a consistency check.

the default value is 1.

2 ► `InfoCollectingPPPPcp` V

is an InfoClass with the following levels.

`level 1`
: displays some information during collecting;

the default value is 0.

3.6 Global variables for the p-power-poly-pcp-groups

The following global variables are available with default value:

1 ► COLLECT_PPOWERPOLY_PCP V

is a global variable determining if every *p*-power-poly-pcp-group element is collected, when created, the default value is true.

4 Parametrised Presentations

In this chapter we describe which pp-presentations for infinite coclass sequences (see [ELG08]) are provided.

4.1 Provided pp-presentations

1 ► `ParPresGlobalVar_2_1` V
► `ParPresGlobalVar_2_2` V
► `ParPresGlobalVar_3_1` V

are lists consisting of the pp-presentations of the infinite coclass sequences of finite p-groups of coclass r, where the first number in the name gives the underlying prime and the second the underlying coclass. Each entry in the list is a record *rec*(*rel*, *expo*, *n*, *d*, *m*, *prime*, *cc*, *expo_vec*, *name*) with $m = 0$ and *expo_vec* = [ ]. The record entries are of a form such that each record can be used as input for `PPPPcpGroups`, 3.2.1. See 3.2.1 for more information.

2 ► `ParPresGlobalVar_p_r_Names` V

gives the names of the infinite coclass sequences of finite p-groups of coclass r.

5 Schur extensions for p-power-poly-pcp-groups

In this chapter we describe how the consistent pp-presentations of infinite coclass sequences can be used to compute a pp-presentation for the corresponding Schur extensions (see [EF11]).

For a group $G = F/R$ the Schur extension H is defined as $H = F/[F,R]$ (see [EN08]).

So for a parameter x that can take values in the positive integers, let $(G_x = F/R_x | x \in \mathbb{N})$, for $\mathbb{N}$ the positive integers, describe an infinite coclass sequence of finite p-groups G_X of coclass r. Then for each value for the parameter x, the group G_x has a consistent polycyclic presentation with generators $g_1, \cdots, g_n, t_1, \cdots, t_d$ and relations

$$
\begin{aligned}
g_i^p &= rel[i][i], \\
t_i^{expo} &= rel[n+i][n+i], \\
g_i^{g_j} &= rel[j][i], \\
t_i^{g_j} &= rel[j][n+i], \\
t_i^{t_j} &= 1\cdot
\end{aligned}
$$

Then we compute a consistent pp-presentation of the corresponding Schur extensions of with generators $g_1, \cdots, g_n, t_1, \cdots, t_d, c_1, \cdots c_m$ and relations

$$
\begin{aligned}
g_i^p &= rel[i][i], \\
t_i^{expo} &= rel[n+i][n+i], \\
c_i^{expo\_vec[i]} &= rel[n+d+i, n+d+i], \\
g_i^{g_j} &= rel[j][i], \\
t_i^{g_j} &= rel[j][n+i], \\
t_i^{t_j} &= rel[n+j][n+i], \\
c_i^{g_j} &= 1, \\
c_i^{t_j} &= 1, \\
c_i^{c_j} &= 1\cdot
\end{aligned}
$$

where the t_i's commute modulo $c\_1, \cdots, c\_m$ and the c_i's are central.

5.1 Computing Schur extensions

1 ▶ `SchurExtParPres( G )`

computes the Schur extensions corresponding to the p-power-poly-pcp-groups G and returns them as p-power-poly-pcp-groups.

2 ▶ `SchurExtParPres( ParPres )` F

computes a consistent pp-presentation of Schur extensions of the groups defined by the record *ParPres* which describes p-power-poly-pcp-groups. The output is a record *rec*(*rel*, *expo*, *n*, *d*, *m*, *prime*, *cc*, *expo_vec*, *name*), which describes the Schur extensions as p-power-poly-pcp-groups; it is encoded in a form that it can be used as input for `PPPPcpGroups`, 3.2.1.

```
gap> SchurExtParPres( ParPresGlobalVar_2_1[1] );
rec( prime := 2,
  rel := [ [ [ [ 7, 1 ] ] ], [ [ [ 2, 1 ], [ 3, -1+2*2^x ], [ 6, 1-2*2^x ] ],
          [ [ 3, 1 ], [ 5, 1 ] ] ],
      [ [ [ 3, -1+2*2^x ], [ 4, 1 ], [ 6, 2-2*2^x ] ], [ [ 3, 1 ] ],
          [ [ 4, 1 ], [ 6, 2*2^x ] ] ],
      [ [ [ 4, 1 ] ], [ [ 4, 1 ] ], [ [ 4, 1 ] ], [ [ 4, 0 ] ] ],
      [ [ [ 5, 1 ] ], [ [ 5, 1 ] ], [ [ 5, 1 ] ], [ [ 5, 1 ] ], [ [ 5, 0 ] ] ]
        ,
      [ [ [ 6, 1 ] ], [ [ 6, 1 ] ], [ [ 6, 1 ] ], [ [ 6, 1 ] ], [ [ 6, 1 ] ],
          [ [ 6, 0 ] ] ],
      [ [ [ 7, 1 ] ], [ [ 7, 1 ] ], [ [ 7, 1 ] ], [ [ 7, 1 ] ], [ [ 7, 1 ] ],
          [ [ 7, 1 ] ], [ [ 7, 0 ] ] ] ], n := 2, d := 1, m := 4,
  expo := 2*2^x, expo_vec := [ 2, 0, 0, 0 ], cc := fail, name := "SchurExt_D"
 )
```

5.2 Computing other invariants from Schur extensions

1 ▶ `AbelianInvariantsMultiplier( G )` F

computes the abelian invariants of the Schur multiplicators *M(G)* of the p-power-poly-pcp-groups G. The output is a list $[d_1, \cdots, d_k]$ consisting elements d_i, depending on the underlying parameter, such that $M(G) \cong C_{d_1} \times \ldots \times C_{d_k}$.

```
gap> G := PPPPcpGroups( ParPresGlobalVar_2_1[1] );
< P-Power-Poly-pcp-groups with 3 generators of relative orders [ 2,2,2*2^x ] >
gap> AbelianInvariantsMultiplier( G );
[ 2 ]
```

2 ▶ `SchurMultiplicatorPPPPcps( G )` F

computes the Schur multiplicators of the p-power-poly-pcp-groups G and then returns the corresponding `PPPPcpGroups`, 3.2.1.

```
gap> G := PPPPcpGroups( ParPresGlobalVar_3_1[1] );
< P-Power-Poly pcp-group with 5 generators of relative orders [ 3,3,3,3*3^x,3*3^x ] >
gap> SchurMultiplicatorPPPPcps( G );
< P-Power-Poly-pcp-groups with 2 generators of relative orders [ 3,9*3^x ] >
```

3 ▶ `AbelianInvariants( G )` F

computes the abelian invariants of the p-power-poly-pcp-groups G and returns them as a list of list describing the parametrised elements.

```
gap> G := PPPPcpGroups( ParPresGlobalVar_2_1[1] );
< P-Power-Poly-pcp-groups with 3 generators of relative orders [ 2,2,2*2^x ] >
gap> AbelianInvariants( G );
[ 2, 2 ]
```

4 ► `ZeroCohomologyPPPPcps( G[, p] )` F

computes the zero-th-cohomology groups $H^0(G,R)$ of the p-power-poly-pcp-groups G with coefficients in R, where $R \cong GF(p)$ if the prime p is given or $R \cong \mathbb{Z}$ otherwise. The action of G on R is taken to be trivial. The function returns a list of integers $[a_1,\ldots,a_k]$ where the cohomology group is isomorphic to $C_{a_1} \times \ldots \times C_{a_k}$ with C_i a cyclic group of order i (for $i > 0$) and C_0 is interpreted as $\mathbb{Z}$.

```
gap> G := PPPPcpGroups( ParPresGlobalVar_2_1[1] );
< P-Power-Poly-pcp-groups with 3 generators of relative orders [ 2,2,2*2^x ] >
gap> ZeroCohomologyPPPPcp( G, 2 );
[ 2 ]
```

5 ► `FirstCohomologyPPPPcps( G[, p] )` F

computes the first-cohomology groups $H^1(G,R)$ of the p-power-poly-pcp-groups G with coefficients in R, where $R \cong GF(p)$ if the prime p is given or $R \cong \mathbb{Z}$ otherwise. The action of G on R is taken to be trivial. The function returns a list of integers $[a_1,\ldots,a_k]$ where the cohomology group is isomorphic to $C_{a_1} \times \ldots \times C_{a_k}$ with C_i a cyclic group of order i (for $i > 0$) and C_0 is interpreted as $\mathbb{Z}$.

```
gap> G := PPPPcpGroups( ParPresGlobalVar_2_1[1] );
< P-Power-Poly-pcp-groups with 3 generators of relative orders [ 2,2,2*2^x ] >
gap> FirstCohomologyPPPPcps( G );
[  ]
```

6 ► `SecondCohomologyPPPPcps( G[, p] )` F

computes the second-cohomology groups $H^2(G,R)$ of the p-power-poly-pcp-groups G with coefficients in R, where $R \cong GF(p)$ if the prime p is given or $R \cong \mathbb{Z}$ otherwise. The action of G on R is taken to be trivial. The function returns a list of integers $[a_1,\ldots,a_k]$ where the cohomology group is isomorphic to $C_{a_1} \times \ldots \times C_{a_k}$ with C_i a cyclic group of order i (for $i > 0$) and C_0 is interpreted as $\mathbb{Z}$.

```
gap> G := PPPPcpGroups( ParPresGlobalVar_2_1[1] );
< P-Power-Poly-pcp-groups with 3 generators of relative orders [ 2,2,2*2^x ] >
gap> SecondCohomologyPPPPcps( G, 2 );
[ 2, 2, 2 ]
```

5.3 Info classes for the computation of the Schur extension

The following info classes are available

1 ► `InfoConsistencyRelPPowerPoly` V

`level 1`
: shows which consistency relations are computed and gives the result;

the default value is 0.

2 ► `InfoCollectingPPowerPoly` V

`level 1`
: shows what is done during collecting;

the default value is 0.

Bibliography

[dS00] M. du Sautoy. Counting p-groups and nilpotent groups. *Inst. Hautes Etudes Sci. Publ. Math.*, 92:63–112, 2000.

[EF11] B. Eick and D. Feichtenschlager. Computation of low-dimensional (co)homology groups for infinite sequences of p-groups with fixed coclass. *Internat. J. Algebra Comput.*, 21(4):635–649, 2011.

[ELG08] B. Eick and C. R. Leedham-Green. On the classification of prime-power groups by coclass. *Bulletin London Math. Soc.*, 40(2), 2008.

[EN08] B. Eick and W. Nickel. Computing the Schur multiplicator and the nonabelian tensor square of a polycyclic group. *J. Algebra*, 320(2):927–944, 2008.

[LGN80] C. R. Leedham-Green and M. F. Newman. Space groups and groups of prime-power order I. *Arch. Math.*, 35:193–202, 1980.