NQL

Computing nilpotent quotients of L-presented groups

A GAP 4 package

by

René Hartung
Institute for Computational Mathematics
TU Braunschweig

Contents

1 Preface

In 1980, Grigorchuk [Gri80] gave an example of an infinite, finitely generated torsion group, which provided a counter-example for the general Burnside problem. It is nowadays called the Grigorchuk group and was originally defined as a group of transformations of the interval $[0, 1]$ which preserve the Lebesgue measure. Grigorchuk [Gri80] also showed that this group is not finitely presented. In 1985, Lysenok [Lys85] determined the following presentation for the Grigorchuk group

$$\langle a, b, c, d \mid a^2, b^2, c^2, d^2, bcd, [d, d^a]^{\sigma^n}, [d, d^{acaca}]^{\sigma^n}, (n \in \mathbb{N})\rangle,$$

where σ is the homomorphism of the free group on a, b, c, d induced by $a \mapsto c^a, b \mapsto d, c \mapsto b$, and $d \mapsto c$. Thus the infinitely many relators of this presentation can be described in finite terms using powers of the endomorphism σ.

In 2003, Bartholdi [Bar03] introduced the notion of an L-presented group for groups of this type (see Chapter 2 for a precise definition of L-presented groups). He proved that each finitely generated, contracting, semi-fractal, regular branch group is finitely L-presented but not finitely presented.

The NQL-package defines new GAP objects to work with L-presented groups. The main part of the package is a nilpotent quotient algorithm for L-presented groups. That is an algorithm which takes as input an L-presented group G and a positive integer c. It computes a polycyclic presentation for the lower central series quotient $G/\gamma_{c+1}(G)$.

The nilpotent quotient algorithm defined in this package generalizes the method by Nickel [Nic96] as implemented in the NQ-package of GAP: see [Nic03]. In difference to NQ, the NQL-package is implemented in GAP.

Our method can be readily modified to determine p-quotients of a finitely L-presented group. An implementation of a PQ is planned for future expansions of the NQL-package.

As finite presentations can be considered as a special type of finite L-presentations, our algorithm also applies to finitely presented groups. It coincides with Nickel's NQ in this special case.

2 Introduction to L-presented groups

Let S be an arbitrary alphabet, Q and R be finite subsets of the free group F_S on S, and Φ be a finite set of homomorphisms $\varphi : F_S \to F_S$. An **L-presentation** is an expression of the form $\langle S \mid Q \mid \Phi \mid R\rangle$. It defines the **L-presented group** $G = F_S/K$ where

$$K = \left\langle Q \cup \bigcup_{\varphi\in\Phi^*} R^{\varphi} \right\rangle^{F_S}$$

and Φ^* is the monoid generated by Φ, i.e. the closure of $\Phi \cup \{\mathrm{id}\}$ under composition.

The elements in Q will be called **fixed relators** and the elements in R will be called **iterated relators**. Furthermore an L-presentation will be called

- **ascending** if Q is empty.
- **invariant** if the normal subgroup K is φ-invariant for each φ in Φ; that is, $K^{\varphi} \subseteq K$ for each φ in Φ.

Note that every ascending L-presentation is invariant. In general it is a non trivial task to decide whether a given L-presentation is invariant.

2.1 Creating an L-presented group

The construction of an L-presented group is similar to the construction of finitely presented groups (see Chapter 45.1 of the GAP Reference manual for further details).

1 ▶ `LPresentedGroup( F, frels, endos, irels )` F

returns the GAP object of an L-presented group with the underlying free group F, the fixed relators *frels*, the set of endomorphisms *endos*, and the iterated relators *irels*. The input variables *frels* and *irels* are finite subsets of F and *endos* is a finite list of homomorphisms $F \to F$.

The Grigorchuk group,

$$\langle a, b, c, d \mid a^2, b^2, c^2, d^2, bcd \mid \sigma \mid [d, d^a], [d, d^{acaca}]\rangle,$$

can be constructed as follows.

```
gap> F:=FreeGroup("a","b","c","d");
<free group on the generators [ a, b, c, d ]>
gap> AssignGeneratorVariables(F);
#I  Assigned the global variables [ a, b, c, d ]
gap> frels:=[a^2,b^2,c^2,d^2,b*c*d];;
gap> endos:=[GroupHomomorphismByImagesNC(F,F,[a,b,c,d],[c^a,d,b,c])];;
gap> irels:=[Comm(d,d^a),Comm(d,d^(a*c*a*c*a))];;
gap> G:=LPresentedGroup(F,frels,endos,irels);
<L-presented group on the generators [ a, b, c, d ]>
```

2 ▶ `ExamplesOfLPresentations( n )` F

returns some of the examples discussed in [Bar03] for $1 \leq n \leq 9$.

$n = 1$ First Grigorchuk group on 4 generators (see [Gri80], [Lys85]; and [Bar03], Thm. 4.6)

$n = 2$ First Grigorchuk group on 3 generators (see [Gri80], [Lys85]; and [Bar03], Thm. 4.6)

$n = 3$ Lamplighter group ([Bar03], Thm. 4.1)

$n = 4$ Brunner-Sidki-Vieira group (see [BSV99]; and [Bar03], Thm. 4.4)

$n = 5$ Grigorchuk supergroup (see [BG02]; and [Bar03], Thm. 4.6)

$n = 6$ Fabrykowski-Gupta-3 group (see [FG85]; cf. [Bar03], Thm. 4.7)

$n = 7$ Gupta-Sidki-3 group (see [Sid87]); cf. [Bar03], Thm. 4.9)

$n = 8$ an index-3 subgroup of the Gupta-Sidki group

$n = 9$ Basilica group (see [GZ02], [BV05])

3 ► `EngelGroup( n, c )` F

returns an L-presentation for an Engel group on n generators that satisfy the c-th Engel identity.

4 ► `FreeNilpotentGroup( n, c )` F

returns an L-presentation for a free nilpotent group of class c on n generators.

5 ► `GeneralizedFabrykowskiGuptaLpGroup( n )`

returns an L-presentation for the n-th generalized Fabrykowski-Gupta group. For n=3 it coincides with the Fabrykowski-Gupta group above.

2.2 The underlying free group

An L-presented group is defined as an image of its underlying free group. Note that these are two different GAP objects, but the elements of the L-presented group are represented by words in the underlying free group.

1 ► `FreeGroupOfLpGroup( LpGroup )` A

returns the underlying free group of the L-presented group *LpGroup*.

2 ► `FreeGeneratorsOfLpGroup( LpGroup )` A

returns the generators of the free group underlying the L-presented group *LpGroup*.

3 ► `GeneratorsOfGroup( LpGroup )` O

returns the generators of the L-presented group *LpGroup*. These are the images of the generators of the underlying free group under the natural homomorphism.

4 ► `UnderlyingElement( elm )` O

returns the preimage of an L-presented group element *elm* in the underlying free group. More precisely, each element of an L-presented group is represented by an element in the free group. This method returns the corresponding element in the free group.

5 ► `ElementOfLpGroup( fam, elm )` O

returns the element in the L-presented group represented by the word *elm* on the generators of the underlying free group if *fam* is the family of L-presented group elements.

```
gap> F:=FreeGroup(2);;
gap> G:=LPresentedGroup(F,[F.1^2],[IdentityMapping(F)],[F.2]);;
gap> FreeGroupOfLpGroup(G)=F;
true
gap> GeneratorsOfGroup(G);
[ f1, f2 ]
gap> FreeGeneratorsOfLpGroup(G);
[ f1, f2 ]
gap> last=last2;
false
gap> UnderlyingElement(G.1);
f1
gap> last in F;
true
gap> ElementOfLpGroup( ElementsFamily( FamilyObj( G ) ), last2 ) in G;
true
```

2.3 Accessing an L-presentation

The fixed relators, the iterated relators, and the endomorphisms of an L-presented group can be accessed by the following functions.

1 ▶ `FixedRelatorsOfLpGroup(` *LpGroup* `)` A

returns the fixed relators of the L-presented group *LpGroup* as words in the underlying free group.

2 ▶ `IteratedRelatorsOfLpGroup(` *LpGroup* `)` A

returns the iterated relators of the L-presented group *LpGroup* as words in the underlying free group.

3 ▶ `EndomorphismsOfLpGroup(` *LpGroup* `)` A

returns the endomorphisms of the L-presented group *LpGroup* as endomorphisms of the underlying free group.

```
gap> F:=FreeGroup(2);;
gap> G:=LPresentedGroup(F,[F.1^2],[IdentityMapping(F)],[F.2]);
<L-presented group on the generators [ f1, f2 ]>
gap> FixedRelatorsOfLpGroup(G);
[ f1^2 ]
gap> IteratedRelatorsOfLpGroup(G);
[ f2 ]
gap> EndomorphismsOfLpGroup(G);
[ IdentityMapping( <free group on the generators [ f1, f2 ]> ) ]
```

2.4 Properties and attributes of L-presented groups

To determine the method for the nilpotent quotient algorithm, L-presented groups have the following properties:

1 ▶ `IsAscendingLPresentation(` *LpGroup* `)` P

tests whether the L-presentation of *LpGroup* is ascending; that is if the set of fixed relators is empty.

2 ▶ `IsInvariantLPresentation(` *LpGroup* `)` P

tests whether the L-presentation of *LpGroup* is invariant. Note that no method is implemented.

3 ► `UnderlyingInvariantLPresentation( LpGroup )` A

returns an underlying invariant L-presentation for the L-presented group *LpGroup*.

An **underlying invariant L-presentation** for the L-presentation $\langle S \mid Q \mid \Phi \mid R\rangle$ is an invariant L-presentation $\langle S \mid Q' \mid \Phi \mid R\rangle$ with $Q' \subseteq Q$. Note that such invariant L-presentation always exists since $Q' = \emptyset$ yields an ascending and hence invariant L-presentation.

The underlying invariant L-presentation is used for computation purposes in the nilpotent quotient algorithm. For this, it is useful to have such a presentation with Q' as large as possible. The method implemented in `UnderlyingInvariantLPresentation` returns the ascending L-presentation in general.

4 ► `EmbeddingOfAscendingSubgroup( LpGroup )` A

returns an embedding of an ascending subgroup of the L-presented group *LpGroup*. This attribute is set for ascending L-presentations only. In this case it is the identity map of *LpGroup*.

2.5 Methods for L-presented groups

1 ► `MappedWord( x, gens, imgs )` O

returns the new group element that is obtained by replacing each occurrence of a generator *gen* in the list of generators *gens* by the corresponding group element *img* in the list of group elements *imgs*. The lists *gens* and *imgs* must of course have the same length.

2 ► `EpimorphismFromFpGroup( LpGroup, n )` O

returns an epimorphism from a finitely presented group on *LpGroup*. The finitely presented group is achieved from *LpGroup* by applying only words of length at most n of the monoid generated by the endomorphisms of *LpGroup*.

3 ► `SplitExtensionByAutomorphismsLpGroup( LpGroup, H, auts)` O

returns the split extension of the *LpGroup* by an L-presented or a finitely presented group H where the action of each generator of H on *LpGroup* is described by an automorphism in *auts*. Thus for each generator of H an automorphisms in *auts* must exist.

```
gap> F:=FreeGroup("a");
<free group on the generators [ a ]>
gap> H:=F/[F.1^3];
<fp group on the generators [ a ]>
gap> U:=ExamplesOfLPresentations(8);
<L-presented group on the generators [ t, u, v ]>
gap> aut:=GroupHomomorphismByImagesNC(U,U,[U.1,U.2,U.3],[U.2,U.3,U.1]);
[ t, u, v ] -> [ u, v, t ]
gap> SplitExtensionByAutomorphismsLpGroup(U,H,[aut]);
<L-presented group on the generators [ t, u, v, a ]>
```

3 Nilpotent Quotients of L-presented groups

Our method to determine polycyclic presentations for nilpotent quotients of L-presented groups generalizes Nickel's nilpotent quotient algorithm for finitely presented groups. We refer to the documentation of the NQ-package for further details on the algorithm.

3.1 New methods for L-presented groups

1 ▶ `NilpotentQuotientLpGroup( LpGroup[, c] )` O

returns a polycyclic presentation for $LpGroup/\gamma_{c+1}(LpGroup)$ if c is specified. If c is not given, this method attempts to compute the largest nilpotent quotient of *LpGroup* and will terminate only if *LpGroup* has a largest nilpotent quotient.

The following example computes the class-5-quotient of the Grigorchuk group on four generators.

```
gap> G:=ExamplesOfLPresentations(1);;
gap> H:=NilpotentQuotientLpGroup(G,5);
Pcp-group with orders [ 2, 2, 2, 2, 2, 2, 2, 2, 2, 2 ]
gap> lcs:=LowerCentralSeries(H);
[ Pcp-group with orders [ 2, 2, 2, 2, 2, 2, 2, 2, 2, 2 ],
  Pcp-group with orders [ 2, 2, 2, 2, 2, 2, 2 ],
  Pcp-group with orders [ 2, 2, 2, 2, 2 ], Pcp-group with orders [ 2, 2, 2 ],
  Pcp-group with orders [ 2, 2 ], Pcp-group with orders [  ] ]
gap> List([1..5],x->lcs[x]/lcs[x+1]);
[ Pcp-group with orders [ 2, 2, 2 ], Pcp-group with orders [ 2, 2 ],
  Pcp-group with orders [ 2, 2 ], Pcp-group with orders [ 2 ],
  Pcp-group with orders [ 2, 2 ] ]
```

2 ▶ `LargestNilpotentQuotient( LpGroup )` A

returns the largest nilpotent quotient of the L-presented group *LpGroup* if such quotient exists. It uses the `NilpotentQuotientLpGroup`-method of the NQL-package. If *LpGroup* does not have a largest nilpotent quotient, this method will not terminate.

3 ▶ `NqEpimorphismNilpotentQuotientLpGroup( LpGroup[, c] )` O
▶ `NqEpimorphismNilpotentQuotientLpGroup( LpGroup[, PcpGroup] )` O

In the first case, this method returns an epimorphism from the L-presented group *LpGroup* into its class-c quotient $LpGroup/\gamma_{c+1}(LpGroup)$ if c is specified. If c is not given, this method attempts to compute an epimorphism into the largest nilpotent quotient of *LpGroup*. If *LpGroup* does not have a largest nilpotent quotient this method will not terminate.

If a pcp-group *PcpGroup* is given as additional parameter, then *PcpGroup* has to be a nilpotent quotient of *LpGroup*. The method computes an epimorphism from the L-presented group *LpGroup* into *PcpGroup*.

The following example computes an epimorphism from the Grigorchuk group on four generators into its class-5-quotient.

```
gap> G:=ExamplesOfLPresentations(1);
<L-presented group on the generators [ a, b, c, d ]>
gap> epi:=NqEpimorphismNilpotentQuotientLpGroup(G,5);
[ a, b, c, d ] -> [ g1, g2*g3, g2, g3 ]
gap> H:=Image(epi);
Pcp-group with orders [ 2, 2, 2, 2, 2, 2, 2, 2, 2, 2 ]
gap> NilpotencyClassOfGroup(H);
5
gap> H:=NilpotentQuotientLpGroup(G,7);
Pcp-group with orders [ 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2 ]
gap> NilpotentQuotientLpGroup(G,10);
Pcp-group with orders [ 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2 ]
gap> NqEpimorphismNilpotentQuotientLpGroup(G,H);
[ a, b, c, d ] -> [ g1, g2*g3, g2, g3 ]
gap> Image(last);
Pcp-group with orders [ 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2 ]
```

4 ► `AbelianInvariants(` *LpGroup* `)` O

returns the Abelian invariants of the Abelian quotient of the L-presented group *LpGroup*. It uses functions of the `NilpotentQuotientLpGroup` method (see 3.1.1).

```
gap> G:=ExamplesOfLPresentations(1);;
gap> AbelianInvariants(G);
[ 2, 2, 2 ]
```

3.2 A short description of the algorithm

In the following we give a short description of the nilpotent quotient algorithm for non-invariant L-presentations.

Let $\langle S \mid Q \mid \Phi \mid R\rangle$ be an L-presentation for the L-presented group G with underlying invariant L-presentation $\langle S \mid Q' \mid \Phi \mid R\rangle$. Denote the L-presented group given by the underlying invariant L-presentation by $\bar{G}$.

The first step in computing a polycyclic presentation for $G/\gamma_c(G)$ of the L-presented group G is to determine a polycyclic presentation for $\bar{G}/\gamma_c(\bar{G})$.

This will be done by induction on c. The induction step is a generalization of Nickel's nilpotent quotient algorithm. It returns a nilpotent presentation H for $\bar{G}/\gamma_c(\bar{G})$ and an epimorphism $\delta : \bar{G} \to H$.

Then a nilpotent presentation for $G/\gamma_c(G)$ can be determined using the nilpotent presentation H and an extension $\delta' : F_S \to H$ of the epimorphism δ. The nilpotent quotient $G/\gamma_c(G)$ is isomorphic to the factor $H/\langle Q^{\delta'}\rangle^H$ of H and the Polycyclic-package can be used to compute a polycyclic presentation for $G/\gamma_c(G)$.

The efficiency of this general approach depends on the underlying invariant L-presentation $\langle S \mid Q' \mid \Phi \mid R\rangle$. The set of fixed relators Q' should be as large as possible. Otherwise, the nilpotent quotient H gets unnecessarily large and slows down the induction step for the underlying invariant L-presentation.

The Grigorchuk group on four generators has the L-presentation

$$\langle a, c, b, d \mid a^2, b^2, c^2, d^2, bcd \mid \sigma \mid [d, d^a], [d, d^{acaca}]\rangle.$$

This L-presentation is invariant and hence the underlying invariant L-presentation is the same as above. There are two ways to compute nilpotent quotients of these groups: First, one can manually set the property `IsInvariantLPresentation` for this group.

```
gap> F:=FreeGroup("a","b","c","d");
<free group on the generators [ a, b, c, d ]>
gap> AssignGeneratorVariables(F);
#I  Assigned the global variables [ a, b, c, d ]
gap> rels:=[a^2,b^2,c^2,d^2,b*d*c];;
gap> endos:=[GroupHomomorphismByImagesNC(F,F,[a,b,c,d],[c^a,d,b,c])];;
gap> itrels:=[Comm(d,d^a),Comm(d,d^(a*c*a*c*a))];;
gap> G:=LPresentedGroup(F,rels,endos,itrels);
<L-presented group on the generators [ a, b, c, d ]>
gap> List(rels,x->x^endos[1]);
[ a^-1*c^2*a, d^2, b^2, c^2, d*c*b ]
gap> SetIsInvariantLPresentation(G,true);
```

Second, one can define the underlying invariant L-presentation as the same L-presentation as the group itself.

```
gap> F:=FreeGroup("a","b","c","d");
<free group on the generators [ a, b, c, d ]>
gap> AssignGeneratorVariables(F);
#I  Assigned the global variables [ a, b, c, d ]
gap> rels:=[a^2,b^2,c^2,d^2,b*d*c];;
gap> endos:=[GroupHomomorphismByImagesNC(F,F,[a,b,c,d],[c^a,d,b,c])];;
gap> itrels:=[Comm(d,d^a),Comm(d,d^(a*c*a*c*a))];;
gap> G:=LPresentedGroup(F,rels,endos,itrels);
<L-presented group on the generators [ a, b, c, d ]>
gap> U:=LPresentedGroup(F,rels,endos,itrels);;
gap> SetUnderlyingInvariantLPresentation(G,U);
```

For saving memory the first method should be preferred. In general the L-presentation is not invariant and thus the second method is the method of choice.

4 The underlying functions

4.1 Nilpotent Quotient Systems for invariant L-presentations

For an L-presented group G which is given by an invariant L-presentation a polycyclic presentation for $G/\gamma_{c+1}(G)$ is computed by determine a weighted nilpotent quotient system for G/G' and extending it inductively to a quotient system for $G/\gamma_{c+1}(G)$.

A quotient system in the NQL package is a record containing the following entries:

Lpres
: the (invariant) L-presentation of the quotient system.

Pccol
: `FromTheLeftCollector` of the nilpotent quotient represented by this quotient system.

Imgs
: images of the generators of the L-presented group under the epimorphism onto the nilpotent quotient *Pccol*. For each generator of the L-presented group *Lpres* there is an integer or a generator exponent list. If the image is an integer *int* the image is a definition of the *int*-th generator of the nilpotent presentation *Pccol*.

Epimorphism
: epimorphism from the L-presented group *Lpres* onto its nilpotent quotient *Pccol* with the images of the generators given by *Imgs*.

Weights
: weight of each generator of the nilpotent presentation *Pccol*.

Definitions
: the definition of each generator of *Pccol*. Each generator in the quotient system has a definition as an image or as a commutator of the form $[a_j, a_i]$ where a_j and a_i are generators of a certain weight. If the i-th entry is an integer, the i-th generator of *Pccol* has a definition as an image. Otherwise the i-th entry is a 2-tuple $[k, l]$ and the i-th generator has a definition as commutator $[a_k, a_l]$.

A quotient system of an L-presented group given by an invariant L-presentation can be computed by the following functions. Both are implemented in the `NilpotentQuotientLpGroup-` and `NqEpimorphismNilpotentQuotientLpGroup` method.

1 ► `InitQuotientSystem(` *LpGroup* `)`

returns a weighted nilpotent quotient system for the largest Abelian quotient of the L-presented group *LpGroup*.

2 ► `ExtendQuotientSystem(` *QS* `)`

extends the weighted nilpotent quotient system *QS* of an L-presented group given by an invariant L-presentation.

```
gap> G:=ExamplesOfLPresentations(1);
<L-presented group on the generators [ a, b, c, d ]>
gap> Q:=InitQuotientSystem(G);
rec( Lpres := <L-presented group on the generators [ a, b, c, d ]>,
  Pccol := <<from the left collector with 3 generators>>,
  Imgs := [ 1, [ 2, 1, 3, 1 ], 2, 3 ], Epimorphism := [ a, b, c, d ] ->
    [ g1, g2*g3, g2, g3 ], Weights := [ 1, 1, 1 ], Definitions := [ 1, 3, 4 ]
 )
gap> ExtendQuotientSystem(Q);
rec( Lpres := <L-presented group on the generators [ a, b, c, d ]>,
  Pccol := <<from the left collector with 5 generators>>,
  Imgs := [ 1, [ 2, 1, 3, 1 ], 2, 3 ],
  Definitions := [ 1, 3, 4, [ 2, 1 ], [ 3, 1 ] ],
  Weights := [ 1, 1, 1, 2, 2 ], Epimorphism := [ a, b, c, d ] ->
    [ g1, g2*g3, g2, g3 ] )
```

4.2 Attributes of L-presented groups

To avoid repeated extensions of quotient systems the largest known quotient system is stored as an attribute of the invariant L-presentation. For non-invariant L-presentations the known nilpotent quotients and its epimorphisms are stored as an attribute.

1 ▶ `NilpotentQuotientSystem(` *LpGroup* `)` A

returns the largest known nilpotent quotient system of an L-presented group that is given by an invariant L-presentation.

```
gap> G:=ExamplesOfLPresentations(1);;
gap> NilpotentQuotientLpGroup(G,5);
Pcp-group with orders [ 2, 2, 2, 2, 2, 2, 2, 2, 2, 2 ]
gap> NilpotentQuotientSystem(G);
rec( Lpres := <L-presented group on the generators [ a, b, c, d ]>,
  Pccol := <<from the left collector with 10 generators>>,
  Imgs := [ 1, [ 2, 1, 3, 1 ], 2, 3 ],
  Definitions := [ 1, 3, 4, [ 2, 1 ], [ 3, 1 ], [ 4, 2 ], [ 4, 3 ], [ 7, 1 ],
      [ 8, 2 ], [ 8, 3 ] ], Weights := [ 1, 1, 1, 2, 2, 3, 3, 4, 5, 5 ],
  Epimorphism := [ a, b, c, d ] -> [ g1, g2*g3, g2, g3 ] )
gap> NilpotencyClassOfGroup(PcpGroupByCollectorNC(last.Pccol));
5
```

2 ▶ `NilpotentQuotients(` *LpGroup* `)` A

returns all known nilpotent quotients of the non-invariant L-presentation *LpGroup*.

```
gap> G:=ExamplesOfLPresentations(3);;
gap> HasIsInvariantLPresentation(G);
false
gap> NilpotentQuotientLpGroup(G,3);
Pcp-group with orders [ 2, 0, 2, 2 ]
gap> NilpotentQuotients(G);
[ rec( Quotient := Pcp-group with orders [ 2, 0 ],
      Epimorphism := [ a, b, t ] -> [ g1, g1, g2 ] ),
  rec( Quotient := Pcp-group with orders [ 2, 0, 2 ],
      Epimorphism := [ a, b, t ] -> [ g1, g1, g2 ] ),
```

```
    rec( Quotient := Pcp-group with orders [ 2, 0, 2, 2 ],
        Epimorphism := [ a, b, t ] -> [ g1, g1, g2 ] ) ]
gap> NilpotentQuotientSystem(UnderlyingInvariantLPresentation(G));
rec( Lpres := <L-presented group on the generators [ a, b, t ]>,
  Pccol := <<from the left collector with 9 generators>>, Imgs := [ 1, 2, 3 ],
  Definitions := [ 1, 2, 3, [ 3, 1 ], [ 3, 2 ], [ 4, 1 ], [ 5, 2 ], [ 4, 3 ],
      [ 5, 3 ] ], Weights := [ 1, 1, 1, 2, 2, 3, 3, 3, 3 ],
  Epimorphism := [ a, b, t ] -> [ g1, g2, g3 ] )
```

4.3 The Info-Class InfoNQL

To get some information about the progress of the algorithm, one can use the info class `InfoNQL`.

1 ► `InfoNQL`

is the info class of the NQL-package. It gives further information on the progress of the nilpotent quotient algorithm for L-presented groups.

```
gap> SetInfoLevel(InfoNQL,1);;
gap> G:=ExamplesOfLPresentations(1);
#I  The first Grigorchuk group on 4 generators
<L-presented group on the generators [ a, b, c, d ]>
gap> NilpotentQuotientLpGroup(G,5);
#I  Class 1: 3 generators with relative orders: [ 2, 2, 2 ]
#I  Class 2: 2 generators with relative orders: [ 2, 2 ]
#I  Class 3: 2 generators with relative orders: [ 2, 2 ]
#I  Class 4: 1 generators with relative orders: [ 2 ]
#I  Class 5: 2 generators with relative orders: [ 2, 2 ]
Pcp-group with orders [ 2, 2, 2, 2, 2, 2, 2, 2, 2, 2 ]
gap> NilpotentQuotientLpGroup(G,10);
#I  Class 6: 2 generators with relative orders: [ 2, 2 ]
#I  Class 7: 1 generators with relative orders: [ 2 ]
#I  Class 8: 1 generators with relative orders: [ 2 ]
#I  Class 9: 2 generators with relative orders: [ 2, 2 ]
#I  Class 10: 2 generators with relative orders: [ 2, 2 ]
Pcp-group with orders [ 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2 ]
```

2 ► `InfoNQL_MAX_GENS`

this global variable sets the limit of generators whose relative order will be shown on each step of the nilpotent quotient algorithm if the 'InfoLevel' of `InfoNQL` is positive.

Bibliography

[Bar03] Laurent Bartholdi. Endomorphic presentations of branch groups. *J. Algebra*, 268:419–443, 2003.

[BG02] Laurent Bartholdi and Rostislav I. Grigorchuk. On parabolic subgroups and Hecke algebras of some fractal groups. *Serdica Math. J.*, 28(1):47–90, 2002.

[BSV99] A. M. Brunner, Said Sidki, and Ana Cristina Vieira. A just-nonsolvable torsion-free group defined on the binary tree. 211(1):99–114, 1999.

[BV05] Laurent Bartholdi and Bálint Virág. Amenability via random walks. *Duke Math. J.*, 130(1):39–56, 2005.

[FG85] Jacek Fabrykowski and Narain Gupta. On groups with sub-exponential growth functions. *J. Indian Math. Soc. (N.S.)*, 49(3-4):249–256 (1987), 1985.

[Gri80] R.I. Grigorchuk. Burnside's problem on periodic groups. *Functional Analysis and its Applications*, 14:41–43, 1980.

[GZ02] Rostislav Grigorchuk and Andrzej Zuk. On a torsion-free weakly branch group defined by a three state automaton. *Internat. J. Algebra Comput.*, 12(1–2):223–246, 2002.

[Lys85] I.G. Lysenok. A system of defining relations for a Grigorchuk group. *Mathematical Notes*, 38:784–792, 1985.

[Nic96] Werner Nickel. Computing nilpotent quotients of finitely presented groups. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 25:175–191, 1996.

[Nic03] W. Nickel. *NQ*, 2003. A GAP4 package, see [GAP4].

[Sid87] Said Sidki. On a 2-generated infinite 3-group: The presentation problem. *Journal of Algebra*, 110:13–23, 1987.

Index

This index covers only this manual. A page number in *italics* refers to a whole section which is devoted to the indexed subject. Keywords are sorted with case and spaces ignored, e.g., "`PermutationCharacter`" comes before "permutation group".