

# Computational homology of $n$ -types

PHD THESIS

by

Le Van Luyen

*Supervisor:* Professor Graham Ellis

SCHOOL OF MATHEMATICS, STATISTICS AND APPLIED MATHEMATICS

NATIONAL UNIVERSITY OF IRELAND, GALWAY



January 2014

# Contents

<b>Declaration</b>	<b>v</b>
<b>Acknowledgement</b>	<b>vi</b>
<b>List of symbols</b>	<b>vii</b>
<b>Summary</b>	<b>viii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Main goal and outline of thesis . . . . .	2
1.2 Review of necessary background material . . . . .	3
1.2.1 Homological algebra . . . . .	3
1.2.2 Simplicial groups . . . . .	5
<b>2 Homology of 1-types</b>	<b>8</b>
2.1 Homology of groups . . . . .	9
2.2 Persistent group homology of dihedral 2-groups . . . . .	12
2.3 Bar resolution, small resolutions and chain homotopy equivalences .	18
<b>3 Homology of <math>n</math>-types</b>	<b>25</b>
3.1 Perturbation Lemma . . . . .	26

---

3.2	Chain complex for homology of simplicial groups . . . . .	31
<b>4</b>	<b>Eilenberg-Mac Lane spaces</b>	<b>39</b>
4.1	Construction of Eilenberg-Mac Lane simplicial groups . . . . .	40
4.2	Small chain complex for homology of $K(\mathbb{Z}_m, 2)$ . . . . .	45
<b>5</b>	<b>Homology of 2-types</b>	<b>47</b>
5.1	Group theoretic examples of crossed modules . . . . .	48
5.2	Nerve of $\text{cat}^1$ -groups . . . . .	49
5.3	$\text{Cat}^1$ -groups and crossed modules of low order . . . . .	54
5.4	Quasi-isomorphisms of crossed modules . . . . .	61
5.5	Homology of crossed modules . . . . .	67
5.6	2-types of low order . . . . .	70
<b>6</b>	<b>Homology of maps of <math>n</math>-types</b>	<b>77</b>
6.1	Algorithm for homology of maps of $n$ -types . . . . .	78
<b>7</b>	<b>Persistent homology of 2-types</b>	<b>82</b>
7.1	Definition of persistent homology of crossed modules . . . . .	83
7.2	Algorithm for persistent homology of crossed modules . . . . .	92
	<b>Further Works</b>	<b>95</b>
	<b>Bibliography</b>	<b>97</b>
	<b>Index</b>	<b>101</b>
<b>8</b>	<b>Appendix: GAP code</b>	<b>103</b>

---

8.1	Data types . . . . .	104
8.2	List of functions . . . . .	106
8.3	GAP Code . . . . .	113
8.3.1	BarResolutionEquivalence( $R$ ) . . . . .	113
8.3.2	BarComplexEquivalence( $R$ ) . . . . .	118
8.3.3	ChainComplexOfSimplicialGroup( $X$ ) . . . . .	120
8.3.4	EilenbergMacLaneSimplicialGroup( $X,n,l$ ) . . . . .	133
8.3.5	CrossedModuleByAutomorphismGroup( $G$ ) . . . . .	141
8.3.6	CrossedModuleByNormalSubgroup( $G,N$ ) . . . . .	142
8.3.7	Order( $X$ ) . . . . .	142
8.3.8	HomotopyGroup( $X,n$ ) . . . . .	142
8.3.9	CatOneGroupByCrossedModule( $X$ ) . . . . .	143
8.3.10	CrossedModuleByCatOneGroup( $X$ ) . . . . .	146
8.3.11	NerveOfCatOneGroup( $X,n$ ) . . . . .	148
8.3.12	CatOneGroupsByGroup( $G$ ) . . . . .	156
8.3.13	NumberSmallCatOneGroups( $arg$ ) . . . . .	164
8.3.14	SmallCatOneGroup( $m,k,i$ ) . . . . .	165
8.3.15	IsomorphismCatOneGroups( $C,D$ ) . . . . .	166
8.3.16	IdCatOneGroup( $C$ ) . . . . .	169
8.3.17	NumberSmallCrossedModules( $m$ ) . . . . .	172
8.3.18	SmallCrossedModule( $m,k$ ) . . . . .	172
8.3.19	IsomorphismCrossedModules( $XC, XD$ ) . . . . .	173
8.3.20	IdCrossedModule( $X$ ) . . . . .	174

---

8.3.21	SubQuasiIsomorph(C)	175
8.3.22	QuotientQuasiIsomorph(C)	176
8.3.23	QuasiIsomorph(X)	178
8.3.24	Homology(X,n)	179
8.3.25	HomotopyCrossedModule(X)	179
8.3.26	NumberSmallQuasiCrossedModules(m)	180
8.3.27	SmallQuasiCrossedModule(m,k)	180
8.3.28	IdQuasiCrossedModule(X)	180
8.3.29	HomotopyLowerCentralSeriesOfCrossedModule(X)	181
8.3.30	PersistentHomologyOfCrossedModule(X,n)	183

# Declaration

I, Le Van Luyen, certify that the thesis is all my own work and that I have not obtained a degree in this University or elsewhere on the basis of any of this work.

# Acknowledgement

I would like to thank my supervisor Professor Graham Ellis for suggesting the studies from which this thesis arose and for his constructive guidance and warm encouragement during the course of this work. Without his guidances, this thesis would never be done.

Thanks also to all the Riverside Terrapin postgrads; we have had great laughs, good times and a bit of maths. Many thanks to all members of staff at the School of Maths, NUI Galway and especially to Mary Kelly who has always been so attentive and helpful.

I am grateful to the NUI Galway for offering me a PhD fellowship, without which I would not have been able to complete this research.

Finally, it is my pleasure to acknowledge the support and encouragement of my wife at all stages of the preparation of this thesis.

Galway, January 8, 2014

Le Van Luyen

# List of symbols

$[x]$	greatest integer less than or equal to $x$
$\mathbb{Z}$	integer numbers $\{\dots, -2, -1, 0, 1, 2, \dots\}$
$\mathbb{Z}G$	integral group ring
$\mathbb{Z}_n$	$\{\overline{0}, \overline{1}, \dots, \overline{n-1}\}$
$C_n$	cyclic group of order $n$
$D_{2n}$	dihedral group of order $2n$
$\oplus$	direct sum
$\otimes$	tensor product
$\rtimes$	semidirect product
$\mathbb{K}$	field
$\mathbb{F}_p$	field of $p$ elements
$1_A$	identity map from $A$ to $A$
$\text{rank}(f)$	dimension of the image of $f$
$\{\gamma_i G\}_{i \geq 1}$	lower central series of group $G$
$\langle g \rangle$	group generated by element $g$
$\mathbf{1}$	trivial group
$\text{Aut}(G)$	automorphism group of group $G$
$Z(M)$	center of group $M$
$[N, M]$	commutator of two subgroups $N$ and $M$

# Summary

The thesis makes the following new contributions to the area of combinatorial homotopy theory:

1. We determine the persistent homology of dihedral groups of order  $2^n$  and confirm a conjecture in [22]. (See Corollary 2.2.4.)
2. We implement an algorithm for computing a  $\mathbb{Z}G$ -equivariant chain homotopy equivalence

$$R_*^G \rightleftarrows B_*^G$$

between the bar resolution  $B_*^G$  of group  $G$  and the smaller resolution  $R_*^G$  of group  $G$  given in the HAP package [20]. (See Algorithm 2.3.4.)

3. We implement the following functors on computer:
  - The isomorphism between the category of  $\text{cat}^1$ -groups and the category of crossed modules

$$\lambda : (\text{Crossed modules}) \rightarrow (\text{Cat}^1\text{-groups}) \text{ (see Algorithm 5.2.1),}$$

$$\gamma : (\text{Cat}^1\text{-groups}) \rightarrow (\text{Crossed modules}) \text{ (see Algorithm 5.2.2).}$$

- The nerve of  $\text{cat}^1$ -groups

$$\mathcal{N} : (\text{Cat}^1\text{-groups}) \rightarrow (\text{Simplicial groups}) \text{ (see Algorithm 5.2.3).}$$

- The Eilenberg-Mac Lane simplicial group  $K(A, n)$  with  $A$  an abelian group

$$K(-, n) : (\text{Abelian groups}) \rightarrow (\text{Simplicial abelian groups})$$

(see Algorithms 4.1.1, 4.1.2).

4. We devise and implement an algorithm which inputs a finite crossed module  $\partial : M \rightarrow P$  and outputs a quasi-isomorphic crossed module  $\partial' : M' \rightarrow P'$  where  $\partial'$  has order less than or equal to the order of  $\partial$ . (See Algorithm 5.4.2.)

5. We devise and implement an algorithm which inputs a finite group  $G$  and outputs all non-isomorphic  $\text{cat}^1$ -group structures on group  $G$  (see Algorithm 5.3.1). By using this algorithm, we construct data of  $\text{cat}^1$ -groups and crossed modules of order  $m \leq 255$ .
6. We devise and implement an algorithm for computing a chain complex for homology of a simplicial group (see Algorithm 3.2.1). By using this chain complex, we compute the integral homology of simplicial groups.
7. We devise and implement an algorithm for computing a homology map induced by a morphism of simplicial groups. (See Algorithm 6.1.1.)
8. We classify most of the 2-types of “order”  $m \leq 255$ . (See Section 5.6.)
9. We introduce a notion of persistent homology of crossed modules and prove that it is a quasi-isomorphism invariant (see Theorem 7.1.10). We devise and implement an algorithm for computing this notion (see Algorithm 7.2.2).

# Chapter 1

## Introduction

## 1.1 Main goal and outline of thesis

The main goal of this thesis is the development of computational tools for helping with the classification of 2-types. Our primary computational tool is the homology, and persistent homology, of 2-types. We provide a classification of most of the 2-types of “order”  $m \leq 255$ .

The thesis has eight chapters.

Chapter 1 recalls preliminary notions and results that will be used in the thesis. These include chain complexes and simplicial groups.

Chapter 2 is devoted to 1-types. We present a new result in the persistent group homology of dihedral 2-groups. Moreover, we construct an algorithm to compute a chain homotopy equivalence between the bar resolution and the HAP resolution of a group. The algorithm is an important step in developing an algorithm in Chapter 3.

Chapter 3 recalls an important “Perturbation Lemma”. Using this lemma, we construct an algorithm to compute a chain complex for homology of a simplicial group.

Chapters 4, 5 develop special cases of Chapter 3. Using the computation of the chain complex for homology of simplicial groups, we can calculate the homology of crossed modules and Eilenberg-Mac Lane spaces.

Chapter 6 presents an algorithm to compute a homology map induced by a morphism of simplicial groups.

Chapter 7 introduces the definition of persistent homology of a crossed module. The persistent homology is a quasi-isomorphism invariant of crossed modules. We also give an algorithm to compute the persistent homology of crossed modules.

Chapter 8 concerns computation in GAP. We present the data types of objects that are mentioned in this thesis such as:  $\text{cat}^1$ -groups, crossed modules, simplicial groups. We also give the description of all functions relating the computation and their GAP code.

## 1.2 Review of necessary background material

### 1.2.1 Homological algebra

**Definition 1.2.1.** [47] Let  $R$  be a ring. A *chain complex*  $C = (C_n, d_n)_{n \in \mathbb{Z}}$  of  $R$ -modules is a sequence of homomorphisms of  $R$ -modules

$$\cdots \xrightarrow{d_{n+2}} C_{n+1} \xrightarrow{d_{n+1}} C_n \xrightarrow{d_n} C_{n-1} \xrightarrow{d_{n-1}} \cdots$$

such that  $d_n d_{n+1} = 0$  for all  $n$ . The chain complex  $C$  is called a *sequence* if  $\text{Im } d_{n+1} = \text{Ker } d_n$  for all  $n$ .

**Definition 1.2.2.** [47] Let  $C = (C_n, d_n)_{n \in \mathbb{Z}}$  be a chain complex of  $R$ -modules. For each  $n \in \mathbb{Z}$ , the  *$n$ th homology module* of  $C$  is defined to be the quotient module

$$H_n(C) = \frac{\text{Ker } d_n}{\text{Im } d_{n+1}}.$$

**Definition 1.2.3.** [47] Let  $C = (C_n, d_n)$  and  $C' = (C'_n, d'_n)$  be chain complexes of  $R$ -modules. A *chain map*  $f: C \rightarrow C'$  is a sequence of homomorphisms of  $R$ -modules  $f_n: C_n \rightarrow C'_n$  such that the following diagram commutes

$$\begin{array}{ccccccc} \cdots & \longrightarrow & C_{n+1} & \xrightarrow{d_{n+1}} & C_n & \xrightarrow{d_n} & C_{n-1} & \longrightarrow & \cdots \\ & & \downarrow f_{n+1} & & \downarrow f_n & & \downarrow f_{n-1} & & \\ \cdots & \longrightarrow & C'_{n+1} & \xrightarrow{d'_{n+1}} & C'_n & \xrightarrow{d'_n} & C'_{n-1} & \longrightarrow & \cdots \end{array}$$

It is not difficult to prove that a chain map  $f: C \rightarrow C'$  induces  $R$ -module homomorphisms

$$H_n(f): H_n(C) \rightarrow H_n(C') \text{ for all } n.$$

**Definition 1.2.4.** [47] Let  $f, g: C \rightarrow C'$  be chain maps. A *chain homotopy*  $h$  between  $f$  and  $g$ , denoted by  $h: f \simeq g$ , is a sequence of homomorphisms  $h_n: C_n \rightarrow C'_{n+1}$  such that

$$f_n - g_n = d'_{n+1} h_n + h_{n-1} d_n \text{ for all } n.$$

If there exists a chain homotopy between  $f$  and  $g$ , then  $f$  and  $g$  are said to be *chain homotopic*.

**Lemma 1.2.1.** [47] *If  $f, g: C \rightarrow C'$  are chain homotopic then they induce the same*

homomorphisms

$$H_n(C) \rightarrow H_n(C') \text{ for all } n.$$

**Definition 1.2.5.** [47] Let  $f: C \rightarrow C'$  be a chain map. The map  $f$  is said to be a *chain homotopy equivalence* if there is a chain map  $g: C' \rightarrow C$  such that  $gf$  and  $fg$  are chain homotopic to the respective identity chain maps of  $C$  and  $C'$ .

**Definition 1.2.6.** [47] A chain map  $f: C \rightarrow C'$  is said to be a *quasi-isomorphism* if  $f$  induces isomorphisms

$$H_n(f): H_n(C) \xrightarrow{\cong} H_n(C') \text{ for all } n.$$

**Definition 1.2.7.** [47] Two chain complexes  $C$  and  $C'$  are said to be *quasi-isomorphic* if there is a sequence

$$C \xrightarrow{f_1} C_1 \xleftarrow{f_2} C_2 \xrightarrow{f_3} \cdots \xleftarrow{f_k} C'$$

of chain maps such that each  $f_i$  is a quasi-isomorphism for  $1 \leq i \leq k$ .

**Definition 1.2.8.** [47] A *bicomplex*  $M$  of  $R$ -modules is a family  $\{M_{p,q}\}_{p,q \in \mathbb{Z}}$  of  $R$ -modules, together with homomorphisms  $d^h: M_{p,q} \rightarrow M_{p-1,q}$  and  $d^v: M_{p,q} \rightarrow M_{p,q-1}$  such that  $d^h d^h = d^v d^v = d^v d^h + d^h d^v = 0$ . It is pictured as the following diagram

$$\begin{array}{ccccccc} & & \vdots & & \vdots & & \vdots \\ & & \downarrow & & \downarrow & & \downarrow \\ \cdots & \longrightarrow & M_{p+1,q+1} & \xrightarrow{d^h} & M_{p,q+1} & \xrightarrow{d^h} & M_{p-1,q+1} \longrightarrow \cdots \\ & & \downarrow d^v & & \downarrow d^v & & \downarrow d^v \\ \cdots & \longrightarrow & M_{p+1,q} & \xrightarrow{d^h} & M_{p,q} & \xrightarrow{d^h} & M_{p-1,q} \longrightarrow \cdots \\ & & \downarrow d^v & & \downarrow d^v & & \downarrow d^v \\ \cdots & \longrightarrow & M_{p+1,q-1} & \xrightarrow{d^h} & M_{p,q-1} & \xrightarrow{d^h} & M_{p-1,q-1} \longrightarrow \cdots \\ & & \downarrow & & \downarrow & & \downarrow \\ & & \vdots & & \vdots & & \vdots \end{array}$$

**Definition 1.2.9.** [47] Let  $M = \{M_{p,q}\}_{p,q \in \mathbb{Z}}$  be a bicomplex of  $R$ -modules. The *total complex* of  $M$ , denoted by  $\text{Tot}(M)$ , is a chain complex defined by

$$\text{Tot}(M)_n = \bigoplus_{p+q=n} M_{p,q}$$

with boundary map given by  $d_n(x) = d^v(x) + d^h(x)$  for  $x \in M_{p,q}$ .

## 1.2.2 Simplicial groups

**Definition 1.2.10.** [35] Let  $\mathcal{C}$  be a category. The category  $s\mathcal{C}$  of simplicial objects over  $\mathcal{C}$  is defined as follows:

- An object  $K \in s\mathcal{C}$  consists of
  - A set of objects  $\{K_n\}_{n \geq 0}$  in  $\mathcal{C}$ ;
  - For every pair of integers  $(i, n)$  such that  $0 \leq i \leq n$ , *face* and *degeneracy* maps  $d_i: K_n \rightarrow K_{n-1}$  and  $s_i: K_n \rightarrow K_{n+1}$  satisfying the *simplicial identities*:

$$d_i d_j = d_{j-1} d_i \quad \text{if } i < j$$

$$d_i s_j = s_{j-1} d_i \quad \text{if } i < j$$

$$d_j s_j = 1 = d_{j+1} s_j$$

$$d_i s_j = s_j d_{i-1} \quad \text{if } i > j + 1$$

$$s_i s_j = s_{j+1} s_i \quad \text{if } i \leq j.$$

- Let  $K$  and  $L$  be simplicial objects. A *morphism*  $f: K \rightarrow L$  consists of maps  $f_n: K_n \rightarrow L_n$  which commute with face maps and degeneracy maps, that is,  $f_{n-1} d_i = d_i f_n$  and  $f_{n+1} s_i = s_i f_n$  for all  $0 \leq i \leq n$ .

If  $\mathcal{C}$  is a category of set, the elements of  $K_n$  are called the *n-simplices* of  $K$ .

**Definition 1.2.11.** A *simplicial set* is a simplicial object over the category of sets and a *simplicial group* is a simplicial object over the category of groups.

**Definition 1.2.12.** A *bisimplicial set* is a simplicial object over the category of simplicial sets.

**Definition 1.2.13.** Let  $\mathcal{C}$  be a category. The *nerve* of  $\mathcal{C}$ , denoted by  $\mathcal{N}\mathcal{C}$ , is a simplicial set constructed as follows:

- For each integer  $n \geq 0$ , the set  $\mathcal{N}_n \mathcal{C}$  of *n-simplices* is the set of diagrams

$$A_0 \rightarrow A_1 \rightarrow \cdots \rightarrow A_n$$

of objects and morphisms from  $\mathcal{C}$ .

- For each pair of integers  $(i, n)$  such that  $0 \leq i \leq n$ ,
  - the face maps  $d_i: \mathcal{N}_n \mathcal{C} \rightarrow \mathcal{N}_{n-1} \mathcal{C}$  are given by composition of morphisms at the  $i$ th node in the diagram (or dropping the first or last arrow if  $i = 0$  or  $n$  respectively),
  - the degeneracy maps  $s_i: \mathcal{N}_n \mathcal{C} \rightarrow \mathcal{N}_{n+1} \mathcal{C}$  are given by inserting identity morphisms at the  $i$ th node in the diagram.

If  $G$  is a group, then  $G$  can be identified with a category with one object  $*$  and one morphism  $g: * \rightarrow *$  for each element  $g$  of  $G$ . So we can construct the nerve of a group  $G$ . In particular, the *nerve* of group  $G$  is a simplicial set  $\mathcal{N}G$  with

$$\mathcal{N}_0 G = \mathbf{1}; \mathcal{N}_n G = \underbrace{G \times \cdots \times G}_n$$

and a collection of face maps  $d_i: \mathcal{N}_n G \rightarrow \mathcal{N}_{n-1} G$  and degeneracy maps  $s_i: \mathcal{N}_n G \rightarrow \mathcal{N}_{n+1} G$ ,

$$d_i(g_1, \dots, g_n) = \begin{cases} (g_2, \dots, g_n) & \text{if } i = 0, \\ (g_1, \dots, g_i g_{i+1}, \dots, g_n) & \text{if } 0 < i < n, \\ (g_1, \dots, g_{n-1}) & \text{if } i = n, \end{cases}$$

$$s_i(g_1, \dots, g_n) = (g_1, \dots, g_i, 1, g_{i+1}, \dots, g_n) \quad \text{if } 0 \leq i \leq n.$$

**Definition 1.2.14.** Let  $G_*$  be a simplicial group. The *Moore complex* of  $G_*$ , denoted by  $MG_* = (M_n G_*, \partial_n)_{n \geq 0}$ , is defined by

$$M_n G_* = G_n \cap \text{Ker } d_1 \cap \dots \cap \text{Ker } d_n$$

with differential map  $\partial_n: M_n G_* \rightarrow M_{n-1} G_*$  induced from  $d_0$  by restriction.

**Definition 1.2.15.** Let  $G_*$  be a simplicial group. The *homotopy groups* of  $G_*$  are defined to be the homology groups of its Moore complex,

$$\pi_n(G_*) = \text{Ker } (\partial_n: M_n G_* \rightarrow M_{n-1} G_*) / \text{Im } (\partial_{n+1}: M_{n+1} G_* \rightarrow M_n G_*).$$

**Definition 1.2.16.** Let  $G_*$  be a simplicial abelian group. We set  $\mathcal{A}_n G_* = G_n$  ( $n \geq 0$ )

with boundary map given by

$$\partial_n = \sum_{i=0}^n (-1)^i d_i^m : \mathcal{A}_n G_* \rightarrow \mathcal{A}_{n-1} G_*.$$

Then  $\mathcal{A}G_*$  is a chain complex. We call  $\mathcal{A}G_*$  be the *alternating chain complex* of  $G_*$ .

**Definition 1.2.17.** A morphism  $f: G_* \rightarrow G'_*$  of simplicial groups is said to be a *weak equivalence* if  $f$  induces isomorphisms

$$\pi_n(G_*) \xrightarrow{\cong} \pi_n(G'_*) \text{ for all } n \geq 0.$$

**Definition 1.2.18.** Two simplicial groups  $G_*$  and  $G'_*$  are said to be *weakly equivalent* if there is a sequence

$$G_* \xrightarrow{f_1} G_*^1 \xleftarrow{f_2} G_*^2 \xrightarrow{f_3} \dots \xleftarrow{f_k} G'_*$$

of morphisms of simplicial groups such that each  $f_i$  is a weak equivalence for  $1 \leq i \leq k$ .

## Chapter 2

### Homology of 1-types

An *1-type* is a CW-space  $X$  with  $\pi_i X = 0$  for  $i = 0, i \geq 2$ . It can be modelled algebraically by a free  $\mathbb{Z}G$ -resolution where  $G = \pi_1 X$ . (The definition of “ $\mathbb{Z}G$ -resolution” is given below.)

## 2.1 Homology of groups

**Definition 2.1.1.** [42] Let  $G$  be a group and  $\mathbb{Z}$  be the group of integers considered as a trivial  $\mathbb{Z}G$ -module. The map  $\epsilon: \mathbb{Z}G \rightarrow \mathbb{Z}$  from the integral group ring to  $\mathbb{Z}$ , given by  $\sum m_g g \mapsto \sum m_g$ , is called the *augmentation*.

It is easy to see that  $\epsilon$  is a  $\mathbb{Z}G$ -module homomorphism.

**Definition 2.1.2.** [1] Let  $G$  be a group. A *free  $\mathbb{Z}G$ -resolution of  $\mathbb{Z}$*  is an exact sequence of  $\mathbb{Z}G$ -modules

$$R_*^G : \quad \cdots \rightarrow R_{n+1}^G \xrightarrow{\partial_{n+1}} R_n^G \xrightarrow{\partial_n} R_{n-1}^G \rightarrow \cdots \rightarrow R_1^G \xrightarrow{\partial_1} R_0^G \xrightarrow{\epsilon} R_{-1}^G = \mathbb{Z} \rightarrow 0$$

with each  $R_i^G$  a free  $\mathbb{Z}G$ -module for all  $i \geq 0$ .

**Definition 2.1.3.** Let  $R_*^G$  be a free  $\mathbb{Z}G$ -resolution of  $\mathbb{Z}$ . A *contracting homotopy*  $h$  of  $R_*^G$  is a sequence of homomorphisms of  $\mathbb{Z}$ -modules  $h_n: R_n^G \rightarrow R_{n+1}^G$  ( $n \geq -1$ ) such that

$$\begin{aligned} \epsilon h_{-1} &= 1_{\mathbb{Z}}, \\ h_{-1} \epsilon + \partial_1 h_0 &= 1_{R_0^G}, \\ h_{i-1} \partial_i + \partial_{i+1} h_i &= 1_{R_i^G} \text{ for } i > 0. \end{aligned}$$

**Definition 2.1.4.** [1] Let  $R_*^G$  be a free  $\mathbb{Z}G$ -resolution of  $\mathbb{Z}$  and  $A$  be a  $\mathbb{Z}G$ -module. The *homology* of  $G$  with coefficients in  $A$  is defined by

$$H_n(G, A) := H_n(R_*^G \otimes_{\mathbb{Z}G} A) \text{ for all } n \geq 0.$$

**Definition 2.1.5.** Let  $\phi: G \rightarrow G'$  be a group homomorphism. A chain map  $f: R_*^G \rightarrow R_*^{G'}$  of  $\mathbb{Z}$ -chain complexes is said to be  *$\phi$ -equivariant* if  $f_n(gx) = \phi(g)f_n(x)$  for all  $g \in G, x \in R_n^G, n \geq 0$ .

**Proposition 2.1.1.** *Let  $R_*^G$  be a free  $\mathbb{Z}G$ -resolution of  $\mathbb{Z}$  and  $R_*^{G'}$  be a free  $\mathbb{Z}G'$ -resolution of  $\mathbb{Z}$ . Let  $\phi: G \rightarrow G'$  be a group homomorphism. Note that, for  $i \geq 1$ ,  $R_i^{G'}$  is a  $\mathbb{Z}G$ -module with  $G$  acting via  $\phi$ . Then*

(i) *There exists a  $\phi$ -equivariant chain map  $f: R_*^G \rightarrow R_*^{G'}$  for which*

$$\begin{array}{ccc} R_0^G & \xrightarrow{f_0} & R_0^{G'} \\ \downarrow \epsilon & & \downarrow \epsilon' \\ \mathbb{Z} & \xrightarrow{=} & \mathbb{Z} \end{array}$$

*commutes. Here  $\epsilon, \epsilon'$  are surjective homomorphisms with  $\text{Ker } \epsilon = \text{Im } \partial_0, \text{Ker } \epsilon' = \text{Im } \partial'_0$ .*

(ii) *Any two  $\phi$ -equivariant chain maps  $f: R_*^G \rightarrow R_*^{G'}$  and  $g: R_*^G \rightarrow R_*^{G'}$  are chain homotopic via a  $\mathbb{Z}G$ -equivariant homotopy  $h_n: R_n^G \rightarrow R_{n+1}^{G'}$ .*

**Lemma 2.1.2.** [5] *Let  $\phi: G \rightarrow G'$  be a homomorphism of groups and  $A$  be  $\mathbb{Z}G'$ -module. Note that  $A$  is also a  $\mathbb{Z}G$ -module with  $G$  acting via  $\phi$ . Then  $\phi$  induces homomorphisms*

$$H_n(\phi): H_n(G, A) \rightarrow H_n(G', A) \text{ for all } n \geq 0.$$

**Proposition 2.1.3.** [1] *Let  $G$  be a cyclic group of order  $n$  generated by  $x$ . Then there is a free  $\mathbb{Z}G$ -resolution of  $\mathbb{Z}$*

$$R_*^G : \quad \cdots \rightarrow \mathbb{Z}G \xrightarrow{N_x} \mathbb{Z}G \xrightarrow{x-1} \mathbb{Z}G \xrightarrow{N_x} \mathbb{Z}G \xrightarrow{x-1} \mathbb{Z}G \xrightarrow{\epsilon} \mathbb{Z} \rightarrow 0$$

*with  $N_x = 1 + x + \cdots + x^{n-1}$  and*

$$H_m(G, \mathbb{Z}) \cong \begin{cases} \mathbb{Z} & \text{if } m = 0, \\ \mathbb{Z}_n & \text{if } m \text{ odd,} \\ 0 & \text{otherwise.} \end{cases}$$

**Definition 2.1.6.** For  $n \geq 1$  we denote by  $D_{2n}$  the dihedral group generated by  $x, y$  subject to the relations  $x^n = 1, y^2 = 1, yxy^{-1} = x^{-1}$ .

*Remark 2.1.1.* We abuse notation and let  $x, y$  denote generators for  $D_2, D_4, D_6, \dots$  and let the context remove any possible ambiguity.

**Proposition 2.1.4.** [1] *Let  $G$  be the dihedral group of order  $2n$ . Then there is a free  $\mathbb{Z}G$ -resolution of  $\mathbb{Z}$  (denoted by  $R_*^G$ ) arising as the total complex of the following bicomplex*

$$\begin{array}{ccccccc}
 & \vdots & & \vdots & & \vdots & & \vdots \\
 & \downarrow & & \downarrow & & \downarrow & & \downarrow \\
 \cdots & \longrightarrow & \mathbb{Z}G & \xrightarrow{\alpha_{33}} & \mathbb{Z}G & \xrightarrow{\alpha_{23}} & \mathbb{Z}G & \xrightarrow{\alpha_{13}} & \mathbb{Z}G \\
 & & \downarrow \beta_{33} & & \downarrow \beta_{23} & & \downarrow \beta_{13} & & \downarrow \beta_{03} \\
 \cdots & \longrightarrow & \mathbb{Z}G & \xrightarrow{\alpha_{32}} & \mathbb{Z}G & \xrightarrow{\alpha_{22}} & \mathbb{Z}G & \xrightarrow{\alpha_{12}} & \mathbb{Z}G \\
 & & \downarrow \beta_{32} & & \downarrow \beta_{22} & & \downarrow \beta_{12} & & \downarrow \beta_{02} \\
 \cdots & \longrightarrow & \mathbb{Z}G & \xrightarrow{\alpha_{31}} & \mathbb{Z}G & \xrightarrow{\alpha_{21}} & \mathbb{Z}G & \xrightarrow{\alpha_{11}} & \mathbb{Z}G \\
 & & \downarrow \beta_{31} & & \downarrow \beta_{21} & & \downarrow \beta_{11} & & \downarrow \beta_{01} \\
 \cdots & \longrightarrow & \mathbb{Z}G & \xrightarrow{\alpha_{30}} & \mathbb{Z}G & \xrightarrow{\alpha_{20}} & \mathbb{Z}G & \xrightarrow{\alpha_{10}} & \mathbb{Z}G \\
 & & & & & & & & \\
 & & & & \longleftarrow p & & & & \\
 & & & & & & & & \uparrow q
 \end{array}$$

Here

$$\alpha_{pq} = \begin{cases} x - 1 & \text{if } p \text{ odd,} \\ N_x & \text{if } p \text{ even,} \end{cases}$$

with  $N_x = 1 + x + x^2 + \cdots + x^{n-1}$  and

$$\beta_{pq} = \begin{cases} y - 1 & p \equiv 0 \pmod{4}, q \text{ odd,} \\ y + 1 & p \equiv 0 \pmod{4}, q \text{ even,} \\ yx + 1 & p \equiv 1 \pmod{4}, q \text{ odd,} \\ yx - 1 & p \equiv 1 \pmod{4}, q \text{ even,} \\ -y - 1 & p \equiv 2 \pmod{4}, q \text{ odd,} \\ -y + 1 & p \equiv 2 \pmod{4}, q \text{ even,} \\ -yx + 1 & p \equiv 3 \pmod{4}, q \text{ odd,} \\ -yx - 1 & p \equiv 3 \pmod{4}, q \text{ even.} \end{cases}$$

In particular, for  $i \geq 1$ , the boundary map  $\partial_i: R_i^G \rightarrow R_{i-1}^G$  is given by

$$\partial_i(e_{p,q}) = \begin{cases} \alpha_{pq}e_{p-1,q} + \beta_{pq}e_{p,q-1} & \text{if } p, q \geq 1, \\ \alpha_{pq}e_{p-1,q} & \text{if } p \geq 1, q = 0, \\ \beta_{pq}e_{p,q-1} & \text{if } p = 0, q \geq 1, \end{cases}$$

with  $\{e_{p,q}\}_{p+q=i}$  the basis of the  $R_i^G$ .

**Proposition 2.1.5.** [1] *Let  $G$  be the dihedral group of order  $2n$  with  $n$  even. Then*

$$H_m(G, \mathbb{Z}_2) = \mathbb{Z}_2^{m+1} \text{ for all } m \geq 0.$$

## 2.2 Persistent group homology of dihedral 2-groups

Recall that the *lower central series* of group  $G$  is defined by

$$\gamma_1 G = G, \gamma_{i+1} G = [G, \gamma_i G] \text{ for } i \geq 1$$

and the *class* of  $G$  is the smallest non-negative integer  $c$  such that  $\gamma_{c+1} G = \mathbf{1}$ .

In this section, we fix  $G$  to be a  $p$ -group of class  $n - 1$ . The lower central series of  $G$  gives rise to a sequence of homomorphisms of groups.

$$G/\gamma_n G \rightarrow G/\gamma_{n-1} G \rightarrow \cdots \rightarrow G/\gamma_3 G \rightarrow G/\gamma_2 G.$$

Let  $\mathbb{F}_p$  be the field of  $p$  elements. By using the functor  $H_m(-, \mathbb{F}_p)$ , we obtain the sequence of induced linear maps of vector spaces on  $\mathbb{F}_p$

$$H_m(G/\gamma_n G, \mathbb{F}_p) \rightarrow H_m(G/\gamma_{n-1} G, \mathbb{F}_p) \rightarrow \cdots \rightarrow H_m(G/\gamma_3 G, \mathbb{F}_p) \rightarrow H_m(G/\gamma_2 G, \mathbb{F}_p).$$

Now we determine the rank of  $H_m(G/\gamma_i G, \mathbb{F}_p) \rightarrow H_m(G/\gamma_{i-1} G, \mathbb{F}_p)$  and phrase our answer in the language of persistent homology.

**Definition 2.2.1.** Let  $p$  be a prime number and  $G$  be a  $p$ -group of class  $n - 1$ . We have a sequence of linear maps of vector spaces on  $\mathbb{F}_p$

$$H_m(G/\gamma_n G, \mathbb{F}_p) \rightarrow H_m(G/\gamma_{n-1} G, \mathbb{F}_p) \rightarrow \cdots \rightarrow H_m(G/\gamma_3 G, \mathbb{F}_p) \rightarrow H_m(G/\gamma_2 G, \mathbb{F}_p).$$

We let  $PH_m(G)$  be the upper triangular matrix  $PH_m(G) = (p_{ij})_{1 \leq i, j \leq n-1}$  with

- $p_{ij} = \text{rank of } H_m(G/\gamma_{n-i+1} G, \mathbb{F}_p) \rightarrow H_m(G/\gamma_{n-j+1} G, \mathbb{F}_p) \text{ for } i < j.$
- $p_{ii} = \text{the dimension of } H_m(G/\gamma_{n-i+1} G, \mathbb{F}_p).$
- $p_{ij} = 0 \text{ for } i > j.$

The entries  $p_{ij}$  of the matrix  $PH_m(G)$  are called the *persistent Betti numbers* of  $G$  at degree  $m$ .

Now we focus on dihedral groups of order  $2^n$  and we give a result on the persistent Betti numbers of these groups. Recall that if  $G$  is a dihedral group of order  $2^n$  then the class of  $G$  is  $n - 1$ . It means  $\gamma_n G = \mathbf{1}$  and  $\gamma_{n-1} G \neq \mathbf{1}$ .

**Lemma 2.2.1.** *Let  $G = D_{2^n}$ . Then*

(i) *for each  $2 \leq k \leq n$ ,  $\gamma_k G = \langle x^{2^{k-1}} \rangle$  and  $\phi_k: D_{2^k} \cong G/\gamma_k G$  with*

$$\phi_k: x \mapsto x\gamma_k G, y \mapsto y\gamma_k G;$$

(ii) *the sequence of homomorphism*

$$G/\gamma_n G \xrightarrow{\alpha_n} G/\gamma_{n-1} G \xrightarrow{\alpha_{n-1}} \cdots \xrightarrow{\alpha_4} G/\gamma_3 G \xrightarrow{\alpha_3} G/\gamma_2 G$$

*is isomorphic to*

$$D_{2^n} \xrightarrow{\beta_n} D_{2^{n-1}} \xrightarrow{\beta_{n-1}} \cdots \xrightarrow{\beta_4} D_{2^3} \xrightarrow{\beta_3} D_{2^2}$$

*with  $\alpha_k$  the quotient map and  $\beta_k: x \mapsto x, y \mapsto y$  for all  $3 \leq k \leq n$ .*

**Proof.** To prove (i) it suffices to show that  $\gamma_k G = \langle x^{2^{k-1}} \rangle$  by induction on  $k$ .

For  $k = 2$ ,  $[x, y] = xyx^{-1}y^{-1} = xx = x^2$ . So  $\langle x^2 \rangle \leq \gamma_2 G$ . Since  $G$  is a dihedral group of order  $2^n$ , every element of  $G$  can be represented in the form  $x^i y^j$  with  $0 \leq i \leq 2^{n-1} - 1, j = 0, 1$ . Let  $a, b \in G$ . We compute the commutator  $[a, b] = aba^{-1}b^{-1}$ .

- If  $a = x^i, b = x^j$  then  $[a, b] = 1$ .
- If  $a = x^i, b = x^j y$  then  $[a, b] = x^{2i}$ .
- If  $a = x^i y, b = x^j$  then  $[a, b] = x^{-2j}$ .
- If  $a = x^i y, b = x^j y$  then  $[a, b] = x^{2(i-j)}$ .

From the above cases, we see that  $[a, b] \in \langle x^2 \rangle$ . This implies  $\gamma_2 G \leq \langle x^2 \rangle$ . Thus  $\gamma_2 G = \langle x^2 \rangle$ .

Suppose that  $\gamma_k G = \langle x^{2^{k-1}} \rangle$  for some  $2 \leq k \leq n-1$ . Then we have

$$[x^{2^{k-1}}, y] = x^{2^{k-1}} y x^{-2^{k-1}} y^{-1} = x^{2^{k-1}} x^{2^{k-1}} = x^{2^k}.$$

This implies  $\langle x^{2^k} \rangle \leq \gamma_{k+1} G$ . Let  $a \in \gamma_k G, b \in G$ ,

- If  $a = x^{i2^{k-1}}, b = x^j$  then  $[a, b] = 1$ .
- If  $a = x^{i2^{k-1}}, b = x^j y$  then  $[a, b] = x^{i2^k}$ .

We see that all generators of  $\gamma_{k+1} G$  are in  $\langle x^{2^k} \rangle$ . This implies  $\gamma_{k+1} G \leq \langle x^{2^k} \rangle$ . Thus  $\gamma_{k+1} G = \langle x^{2^k} \rangle$ .

By the induction hypothesis we have  $\gamma_k G = \langle x^{2^{k-1}} \rangle$  for all  $2 \leq k \leq n$ .

Part (ii) is an easy observation based on the proof of part (i). □

**Lemma 2.2.2.** *Let  $G = D_{2kn}, G' = D_{2n}$  and let  $\phi: G \rightarrow G'$  be the homomorphism defined by  $\phi(x) = x, \phi(y) = y$ . We consider  $R_*^G, R_*^{G'}$  as the free  $\mathbb{Z}G$ -resolution and free  $\mathbb{Z}G'$ -resolution of  $\mathbb{Z}$  obtained from Proposition 2.1.4. Then there exists a  $\phi$ -equivariant chain map  $f: R_*^G \rightarrow R_*^{G'}$  defined by*

$$\begin{aligned} f_{-1} &= 1_{\mathbb{Z}}, \\ f_i(e_{p,q}^G) &= k \binom{\frac{p}{2}}{2} e_{p,q}^{G'}, \end{aligned}$$

for  $i \geq 0, p+q=i$ , where  $\{e_{p,q}^G\}, \{e_{p,q}^{G'}\}$  are the bases of  $R_i^G$  and  $R_i^{G'}$ .

**Proof.** We need to prove that  $f$  is a  $\mathbb{Z}$ -equivariant chain map. It means  $f_{i-1} \partial_i^G = \partial_i^{G'} f_i$  for all  $i \geq 0$ .

Let us call  $N_x^G, \alpha_{pq}^G, \beta_{pq}^G, \partial_i^G$  be the components of  $R_*^G$  and  $N_x^{G'}, \alpha_{pq}^{G'}, \beta_{pq}^{G'}, \partial_i^{G'}$  be the components of  $R_*^{G'}$  as in Proposition 2.1.4. It is easy to see that

$$\begin{aligned} \phi(N_x^G) &= k N_x^{G'}, \\ \phi(\alpha_{pq}^G) &= \begin{cases} \alpha_{pq}^{G'} & \text{if } p \text{ odd,} \\ k \alpha_{pq}^{G'} & \text{if } p \text{ even,} \end{cases} \\ \phi(\beta_{pq}^G) &= \beta_{pq}^{G'}. \end{aligned}$$

We easily see  $f_{-1}\partial_0^G = \partial_0^{G'}f_0$ . For  $i \geq 1$ ,

- With  $p = 2m + 1$  ( $m \geq 0$ )

$$\begin{aligned}
f_{i-1}\partial_i^G(e_{p,q}^G) &= f_{i-1}(\alpha_{pq}^G e_{p-1,q}^G + \beta_{pq}^G e_{p,q-1}^G) \\
&= \phi(\alpha_{pq}^G) f_{i-1}(e_{p-1,q}^G) + \phi(\beta_{pq}^G) f_{i-1}(e_{p,q-1}^G) \\
&= \alpha_{pq}^{G'} k^{\lfloor \frac{p-1}{2} \rfloor} e_{p-1,q}^{G'} + \beta_{pq}^{G'} k^{\lfloor \frac{p}{2} \rfloor} e_{p,q-1}^{G'} \\
&= \alpha_{pq}^{G'} k^m e_{p-1,q}^{G'} + \beta_{pq}^{G'} k^m e_{p,q-1}^{G'} \\
&= k^m (\alpha_{pq}^{G'} e_{p-1,q}^{G'} + \beta_{pq}^{G'} e_{p,q-1}^{G'}); \\
\partial_i^{G'} f_i(e_{p,q}^G) &= \partial_i^{G'} (k^{\lfloor \frac{p}{2} \rfloor} e_{p,q}^{G'}) = \partial_i^{G'} (k^m e_{p,q}^{G'}) \\
&= k^m (\alpha_{pq}^{G'} e_{p-1,q}^{G'} + \beta_{pq}^{G'} e_{p,q-1}^{G'}).
\end{aligned}$$

- With  $p = 2m$  ( $m \geq 1$ )

$$\begin{aligned}
f_{i-1}\partial_i^G(e_{p,q}^G) &= f_{i-1}(\alpha_{pq}^G e_{p-1,q}^G + \beta_{pq}^G e_{p,q-1}^G) \\
&= \phi(\alpha_{pq}^G) f_{i-1}(e_{p-1,q}^G) + \phi(\beta_{pq}^G) f_{i-1}(e_{p,q-1}^G) \\
&= k\alpha_{pq}^{G'} k^{\lfloor \frac{p-1}{2} \rfloor} e_{p-1,q}^{G'} + \beta_{pq}^{G'} k^{\lfloor \frac{p}{2} \rfloor} e_{p,q-1}^{G'} \\
&= k\alpha_{pq}^{G'} k^{m-1} e_{p-1,q}^{G'} + \beta_{pq}^{G'} k^m e_{p,q-1}^{G'} \\
&= k^m (\alpha_{pq}^{G'} e_{p-1,q}^{G'} + \beta_{pq}^{G'} e_{p,q-1}^{G'}); \\
\partial_i^{G'} f_i(e_{p,q}^G) &= \partial_i^{G'} (k^{\lfloor \frac{p}{2} \rfloor} e_{p,q}^{G'}) = \partial_i^{G'} (k^m e_{p,q}^{G'}) \\
&= k^m (\alpha_{pq}^{G'} e_{p-1,q}^{G'} + \beta_{pq}^{G'} e_{p,q-1}^{G'}).
\end{aligned}$$

So we deduce  $f_{i-1}\partial_i^G = \partial_i^{G'}f_i$  for all  $i \geq 0$ . This implies  $f$  is a  $\mathbb{Z}$ -equivariant chain map.  $\square$

**Theorem 2.2.3.** *Let  $G = D_{2kn}$ ,  $G' = D_{2n}$  with  $n, k$  even and let  $\phi: G \rightarrow G'$  be the homomorphism defined by  $\phi(x) = x$ ,  $\phi(y) = y$ . Then  $\phi$  induces homomorphisms*

$$H_m(\phi): H_m(G, \mathbb{Z}_2) \rightarrow H_m(G', \mathbb{Z}_2) \text{ for all } m \geq 0$$

and

$$\text{rank}(H_m(\phi)) = \begin{cases} 1 & \text{if } m = 0, \\ 2 & \text{if } m > 0. \end{cases}$$

**Proof.** From Lemma 2.2.2, the homomorphism  $\phi$  induces an  $\phi$ -equivariant chain map  $f: R_*^G \rightarrow R_*^{G'}$  satisfying

$$f_i(e_{p,q}^G) = k \binom{p}{2} e_{p,q}^{G'} \text{ for all } i \geq 0.$$

By taking the tensor product with  $\mathbb{Z}_2$ , the chain map  $f$  induces a chain map

$$\tilde{f}: R_*^G \otimes_{\mathbb{Z}G} \mathbb{Z}_2 \rightarrow R_*^{G'} \otimes_{\mathbb{Z}G'} \mathbb{Z}_2.$$

As we know  $\{e_{p,q}^G\}_{p+q=i}$  and  $\{e_{p,q}^{G'}\}_{p+q=i}$  are the bases of  $R_i^G$  and  $R_i^{G'}$ . We can consider  $\{e_{p,q}^G\}_{p+q=i}$  and  $\{e_{p,q}^{G'}\}_{p+q=1}$  as bases of  $R_i^G \otimes_{\mathbb{Z}G} \mathbb{Z}_2$  and  $R_i^{G'} \otimes_{\mathbb{Z}G'} \mathbb{Z}_2$ . Since  $k$  is even, we have

$$\tilde{f}_i(e_{p,q}^G) = \begin{cases} e_{p,q}^{G'} & \text{if } p = 0, 1, \\ 0 & \text{if } p \geq 2. \end{cases}$$

We know that  $R_i^G \otimes_{\mathbb{Z}G} \mathbb{Z}_2 \cong \mathbb{Z}_2^{i+1}$  and  $R_i^{G'} \otimes_{\mathbb{Z}G'} \mathbb{Z}_2 \cong \mathbb{Z}_2^{i+1}$ . It follows that  $\tilde{f}$  is represented by the following diagram

$$\begin{array}{ccccccccccc} \cdots & \longrightarrow & \mathbb{Z}_2^4 & \xrightarrow{0} & \mathbb{Z}_2^3 & \xrightarrow{0} & \mathbb{Z}_2^2 & \xrightarrow{0} & \mathbb{Z}_2 & \longrightarrow & 0 \\ & & \downarrow 0 \oplus 0 \oplus 1 \oplus 1 & & \downarrow 0 \oplus 1 \oplus 1 & & \downarrow 1 \oplus 1 & & \downarrow 1 & & \\ \cdots & \longrightarrow & \mathbb{Z}_2^4 & \xrightarrow{0} & \mathbb{Z}_2^3 & \xrightarrow{0} & \mathbb{Z}_2^2 & \xrightarrow{0} & \mathbb{Z}_2 & \longrightarrow & 0. \end{array}$$

Thus the chain map  $\tilde{f}$  induces homomorphisms  $H_m(\phi): H_m(G, \mathbb{Z}_2) \rightarrow H_m(G', \mathbb{Z}_2)$  with

$$\text{rank}(H_m(\phi)) = \begin{cases} 1 & \text{if } m = 0, \\ 2 & \text{if } m > 0. \end{cases} \quad \square$$

From the above we obtain the following result.

**Corollary 2.2.4.** *Let  $G$  be the dihedral group of order  $2^n$ . The persistent Betti numbers of  $G$  at degree  $m$  form an upper triangular matrix  $PH_m(G) = (p_{ij})$  of size  $(n-1)$  where:*

- If  $m = 0$  then  $p_{ij} = 1$  for all  $1 \leq i \leq j \leq n-1$ .

- If  $m \geq 1$  then

$$p_{ij} = \begin{cases} 2 & \text{if } i < j, \\ m + 1 & \text{if } i = j. \end{cases}$$

**Proof.** By using Lemma 2.2.1, we have

$$G/\gamma_n G \rightarrow G/\gamma_{n-1} G \rightarrow \cdots \rightarrow G/\gamma_3 G \rightarrow G/\gamma_2 G$$

is isomorphic to

$$D_{2^n} \rightarrow D_{2^{n-1}} \rightarrow \cdots \rightarrow D_{2^3} \rightarrow D_{2^2}.$$

Now we apply the functor  $H_m(-, \mathbb{Z}_2)$  to the above two sequences. We obtain that

$$H_m(G/\gamma_n G, \mathbb{Z}_2) \rightarrow H_m(G/\gamma_{n-1} G, \mathbb{Z}_2) \rightarrow \cdots \rightarrow H_m(G/\gamma_3 G, \mathbb{Z}_2) \rightarrow H_m(G/\gamma_2 G, \mathbb{Z}_2)$$

is isomorphic to

$$H_m(D_{2^n}, \mathbb{Z}_2) \rightarrow H_m(D_{2^{n-1}}, \mathbb{Z}_2) \rightarrow \cdots \rightarrow H_m(D_{2^3}, \mathbb{Z}_2) \rightarrow H_m(D_{2^2}, \mathbb{Z}_2).$$

For  $2 \leq j < i \leq n$ , we apply Theorem 2.2.3 for the group homomorphism  $D_{2^i} \rightarrow D_{2^j}$ . We obtain

$$\text{rank}(H_m(D_{2^i}, \mathbb{Z}_2) \rightarrow H_m(D_{2^j}, \mathbb{Z}_2)) = \begin{cases} 1 & \text{if } m = 0, \\ 2 & \text{if } m > 0. \end{cases}$$

On the other hand, by applying Proposition 2.1.5 we have the dimension of  $H_m(D_{2^i}, \mathbb{Z}_2)$  is equal to  $m + 1$ . □

Now we recall the coclass theory and use the results of the persistent homology of dihedral groups to confirm a conjecture in the paper [22].

The coclass theory was initiated in 1980 by Leedham-Green and Newman [32]. It suggests to use the coclass as primary invariant to classify and investigate finite  $p$ -groups. A major tool in coclass theory is the coclass graph  $\mathcal{G}(p, r)$  whose vertices are the  $p$ -groups of coclass  $c$  and two groups  $G$  and  $H$  are connected by an edge if  $H/\gamma(H) \cong G$  where  $\gamma(H)$  denotes the last non-trivial term of the lower central series of  $H$ . Ellis and King [22] introduced notion of persistent homology for coclass trees. Let  $\mathbb{T}$  is a coclass tree in the coclass graph  $\mathcal{G}(p, r)$  and let  $G_l$  denote the  $p$ -group

at level  $l$  on the infinite path of  $\mathbb{T}$ . Let  $\text{Im } v_n^{l,k}$  denote the image of the canonical homology homomorphism  $v_n^{l,k} : H_n(G_{l+k}, \mathbb{F}_p) \rightarrow H_n(G_l, \mathbb{F}_p)$ . Then they define the  $l$ -persistent homology of  $\mathbb{T}$  in degree  $n$  is the subgroup  $P_l H_n(\mathbb{T}) = \bigcap_{k=1}^{\infty} \text{Im } v_n^{l,k}$  of the homology group  $H_n(G_l, \mathbb{F}_p)$ . Note that there is a canonical infinite sequence of surjective homomorphisms

$$\cdots \rightarrow P_{l+2} H_n(\mathbb{T}) \rightarrow P_{l+1} H_n(\mathbb{T}) \rightarrow P_l H_n(\mathbb{T}).$$

The persistent homology  $PH_n(\mathbb{T})$  of  $\mathbb{T}$  is defined to be the inverse limit of this sequence. By using calculations on computer, they strongly suggest the following conjecture

**Conjecture 2.2.5.** For  $\mathbb{T}$  the infinite tree in  $\mathcal{G}(2, 1)$  we have

$$PH_n(\mathbb{T}) = \mathbb{F}_2 \oplus \mathbb{F}_2 \quad (n \geq 1).$$

Moreover, the infinite path of  $\mathbb{T}$  in  $\mathcal{G}(2, 1)$  is a sequence of dihedral groups  $D_4, D_8, D_{16}, D_{32}, \dots, D_{2^k}, D_{2^{k+1}}, \dots$ . By applying Theorem 2.2.3, we obtain

$$\text{Im } v_n^{l,k} = \mathbb{F}_2 \oplus \mathbb{F}_2 \text{ for all } n \geq 1.$$

This implies  $P_l H_n(\mathbb{T}) = \mathbb{F}_2 \oplus \mathbb{F}_2$  for all  $l \geq 1, n \geq 1$ . Thus Conjecture 2.2.5 is confirmed.

## 2.3 Bar resolution, small resolutions and chain homotopy equivalences

**Definition 2.3.1.** For any group  $G$ , a free  $\mathbb{Z}G$ -resolution  $R_*^G$  of  $\mathbb{Z}$  is said to be *finitely generated* if the free  $\mathbb{Z}G$ -module  $R_i^G$  has finite rank for each  $i \geq 0$ .

**Definition 2.3.2.** We say that a group  $G$  is *n-constructible* if we have an algorithm with:

- Input: Group  $G$  and integer  $n \geq 0$ .

- Output: The first  $n + 1$  terms of a free  $\mathbb{Z}G$ -resolution of  $\mathbb{Z}$ ,

$$R_*^G : R_n^G \xrightarrow{d_n} R_{n-1}^G \rightarrow \cdots \rightarrow R_2^G \xrightarrow{d_2} R_1^G \xrightarrow{d_1} R_0^G,$$

in which each  $R_i^G$  is a finitely generated free  $\mathbb{Z}G$ -module for all  $0 \leq i \leq n$ .

The HAP package [20] provides implementations of practical algorithms for computing the first  $n + 1$  terms of finitely generated free  $\mathbb{Z}G$ -resolutions for many groups. The algorithm in HAP have the general form of Algorithm 2.3.1.

**Algorithm 2.3.1.**

**Input:** A  $n$ -constructible group  $G$ .

**Output:**

- The  $\mathbb{Z}G$ -rank of the  $i$ th free module  $R_i^G$  ( $0 \leq i \leq n$ ).
- The image of the  $k$ th free  $\mathbb{Z}G$ -generator of  $R_i^G$  under the boundary homomorphism

$$d_i : R_i^G \rightarrow R_{i-1}^G \quad (1 \leq i \leq n).$$

- The image of the  $k$ th free  $\mathbb{Z}$ -generator of  $R_i^G$  under a contracting homotopy

$$h_i : R_i^G \rightarrow R_{i+1}^G \quad (0 \leq i \leq n - 1).$$

The contracting homotopy in Algorithm 2.3.1 can be used to make algorithmic the following frequent element of choice.

For  $x \in \text{Ker } d_i$  choose an element  $\tilde{x} \in R_{i+1}^G$  such that  $d_{i+1}(\tilde{x}) = x$ .

We can choose  $\tilde{x} = h_i(x)$ . In particular, for any group homomorphism  $\phi : G \rightarrow G'$ , the contracting homotopy on a free  $\mathbb{Z}G'$ -resolution  $R_*^{G'}$  provides an explicit induced  $\phi$ -equivariant chain map  $f : R_*^G \rightarrow R_*^{G'}$ .

We call the free  $\mathbb{Z}G$ -resolution obtained from Algorithm 2.3.1 the *HAP resolution* of group  $G$ .

Moreover, the HAP package [20] provides implementations of an algorithm for com-

putting the HAP complex  $\overline{R}_*^G$  of group  $G$  given by

$$\overline{R}_*^G := R_*^G \otimes_{\mathbb{Z}G} \mathbb{Z}.$$

**Algorithm 2.3.2.**

**Input:** A HAP resolution  $R_*^G$  of group  $G$ .

**Output:**

- The  $\mathbb{Z}$ -rank of the  $i$ th free  $\mathbb{Z}$ -module  $\overline{R}_i^G$  ( $i \geq 0$ ).
- The image of the  $k$ th free  $\mathbb{Z}$ -generator of  $\overline{R}_i^G$  under the boundary homomorphism

$$d_i: \overline{R}_i^G \rightarrow \overline{R}_{i-1}^G \quad (i \geq 1).$$

**Definition 2.3.3.** [42] For any group  $G$ , the *bar resolution* of  $\mathbb{Z}$  is the sequence

$$B_*^G: \quad \cdots \rightarrow B_{n+1}^G \xrightarrow{\partial_{n+1}} B_n^G \xrightarrow{\partial_n} B_{n-1}^G \rightarrow \cdots \rightarrow B_1^G \xrightarrow{\partial_1} B_0^G \xrightarrow{\epsilon} \mathbb{Z} \rightarrow 0$$

where  $B_0$  is the free  $\mathbb{Z}G$ -module on the single generator  $[\ ]$ ,  $\epsilon: B_0 \rightarrow \mathbb{Z}$  is the augmentation. For  $n \geq 1$ ,  $B_n$  is the free  $\mathbb{Z}G$ -module generated by  $n$ -tuples  $[g_1 | \cdots | g_n]$  ( $g_i \in G$ ), and the boundary homomorphism is  $\partial_n = \sum_{i=0}^n (-1)^i d_i: B_n^G \rightarrow B_{n-1}^G$  where

$$d_i[g_1 | \cdots | g_n] = \begin{cases} g_1[g_2 | \cdots | g_n] & \text{if } i = 0, \\ [g_1 | \cdots | g_i g_{i+1} | \cdots | g_n] & \text{if } 0 < i < n, \\ [g_1 | \cdots | g_{n-1}] & \text{if } i = n. \end{cases}$$

We let  $B_*^G$  denote the bar resolution of group  $G$ .

Let  $h_n: B_n^G \rightarrow B_{n+1}^G$  be the  $\mathbb{Z}$ -homomorphism given by

$$g[g_1 | \cdots | g_n] \mapsto [g|g_1 | \cdots | g_n].$$

It is not difficult to check that  $h_n$  is a contracting homotopy of the bar resolution  $B_*^G$ .

We define the *bar complex*  $\overline{B}_*^G$  of group  $G$  to be the chain complex of free abelian groups given by

$$\overline{B}_*^G := B_*^G \otimes_{\mathbb{Z}G} \mathbb{Z}.$$

Note that the module  $B_i^G$  is not of finite rank when  $G$  is infinite. Even for finite groups the rank of  $B_i^G$  is large and it is usually not possible to compute the homology of group  $G$  from its bar complex  $\overline{B}_*$ .

Now we use formulas in the definition of the bar resolution and obtain an algorithm for computing the bar resolution  $B_*^G$  of group  $G$  on computer.

**Algorithm 2.3.3.**

**Input:** A group  $G$  and an integer  $n \geq 0$ .

**Output:**

- The image of the free generator  $[g_1 | \cdots | g_i]$  of  $B_i^G$  under the boundary map

$$\partial_i: B_i^G \rightarrow B_{i-1}^G \quad (1 \leq i \leq n).$$

- The image of the element  $g[g_1 | \cdots | g_i]$  of  $B_i^G$  under the contracting homotopy

$$h_i: B_i^G \rightarrow B_{i+1}^G \quad (0 \leq i \leq n-1).$$

**Procedure:** We consider an element  $mg[g_1 | \cdots | g_i]$  of  $B_i^G$  as a list  $[m, g, g_1, \dots, g_i]$  on the computer. By using the formulas in Definition 2.3.3, we easily compute the boundary homomorphism  $\partial_i$  and the contracting homotopy  $h_i$ .

Since the bar resolution  $B_*^G$  and the HAP resolution  $R_*^G$  of group  $G$  are free  $\mathbb{Z}G$ -resolutions of  $\mathbb{Z}$  there exists a diagram

$$\begin{array}{ccccccc} \cdots & B_{n+1}^G & \xleftarrow{H_n} & B_n^G & \xleftarrow{H_{n-1}} & B_{n-1}^G & \cdots & B_1^G & \xleftarrow{H_0} & B_0^G \\ & \uparrow \iota_{n+1} & & \downarrow \psi_{n+1} & \uparrow \iota_n & \downarrow \psi_n & \uparrow \iota_{n-1} & \downarrow \psi_{n-1} & \uparrow \iota_1 & \downarrow \psi_1 & \uparrow \iota_0 & \downarrow \psi_0 \\ \cdots & R_{n+1}^G & \longrightarrow & R_n^G & \longrightarrow & R_{n-1}^G & \cdots & R_1^G & \longrightarrow & R_0^G \end{array}$$

where

- $\psi_*: B_*^G \rightarrow R_*^G$  is a  $\mathbb{Z}G$ -equivariant chain homotopy equivalence.
- $\iota_*: R_*^G \rightarrow B_*^G$  is a  $\mathbb{Z}G$ -equivariant chain homotopy equivalence.

- The  $\mathbb{Z}G$ -equivariant homotopy map  $H_i: B_i^G \rightarrow B_{i+1}^G$  satisfies

$$\iota_i \psi_i = 1 + \partial_{i+1} H_i + H_{i-1} \partial_i.$$

We give an algorithm for computing a  $\mathbb{Z}G$ -equivariant chain homotopy equivalence between the bar resolution  $B_*^G$  and the HAP resolution  $R_*^G$ .

**Algorithm 2.3.4.**

**Input:** A HAP resolution  $R_*^G$  of group  $G$ .

**Output:**

- The image of the free generator  $[g_1 | \cdots | g_i]$  of  $B_i^G$  under the chain map  $\psi_i: B_i^G \rightarrow R_i^G$ .
- The image of the  $k$ th free generator of  $R_i^G$  under the chain map  $\iota_i: R_i^G \rightarrow B_i^G$ .
- The image of the free generator  $[g_1 | \cdots | g_i]$  of  $B_i^G$  under the homotopy map  $H_i: B_i^G \rightarrow B_{i+1}^G$  satisfying  $\iota_i \psi_i = 1 + \partial_{i+1} H_i + H_{i-1} \partial_i$ .

**Procedure:** We construct  $\iota_i, \psi_i, H_i$  by finding the image of the free generators of  $R_i^G$  and  $B_i^G$  and by using induction. We suppose that  $a_i^k$  is a free generator of  $R_i^G$  and  $b_i^k$  is a free generator of  $B_i^G$ .

- Construction of  $\iota_i$ .
  - For  $i = 0$ , set  $\iota_0 := 1$ .
  - For  $i > 0$ , set  $\iota_i(a_i^k) := h_{i-1} \iota_{i-1} d_i(a_i^k)$ .
- Construction of  $\psi_i$ .
  - For  $i = 0$ , set  $\psi_0 := 1$ .
  - For  $i > 0$ , set  $\psi_i(b_i^k) := h_{i-1} \psi_{i-1} \partial_i(b_i^k)$ .
- Construction of  $H_i$ .
  - For  $i = 0$ , set  $H_0 := 0$

- For  $i > 0$ . Since  $h_i$  is the contracting homotopy of the bar resolution, we have

$$\partial_{i+1}h_i + h_{i-1}\partial_i = 1.$$

So

$$\partial_{i+1}h_i(\iota_i\psi_i - 1 - H_{i-1}\partial_i)(b_i^k) + h_{i-1}\partial_i(\iota_i\psi_i - 1 - H_{i-1}\partial_i)(b_i^k) = (\iota_i\psi_i - 1 - H_{i-1}\partial_i)(b_i^k).$$

Moreover, by using induction, we can easily prove  $\partial_i(\iota_i\psi_i - 1 - H_{i-1}\partial_i) =$

0. It follows that

$$\partial_{i+1}h_i(\iota_i\psi_i - 1 - H_{i-1}\partial_i)(b_i^k) = (\iota_i\psi_i - 1 - H_{i-1}\partial_i)(b_i^k).$$

So we choose  $H_i(b_i^k) := h_i(\iota_i\psi_i - 1 - H_{i-1}\partial_i)(b_i^k)$ . Then

$$\partial_{i+1}H_i(b_i^k) = (\iota_i\psi_i - 1 - H_{i-1}\partial_i)(b_i^k)$$

or

$$\iota_i\psi_i(b_i^k) = (b_i^k) + \partial_{i+1}H_i(b_i^k) + H_{i-1}\partial_i(b_i^k).$$

Note that the construction of the formulae of  $\iota_i, \psi_i, H_i$  looks quite similar to the formulae of  $f_n, g_n, k_n$  in the paper [39] on page 306.

By applying the tensor product to the result obtained from Algorithm 2.3.4 with  $\mathbb{Z}$ , we obtain an algorithm for constructing a  $\mathbb{Z}$ -chain homotopy equivalence between the bar complex  $\overline{B}_*^G$  and the HAP complex  $\overline{R}_*^G$ .

$$\begin{array}{ccccccc} \cdots & \overline{B}_{n+1}^G & \xleftarrow{H_n} & \overline{B}_n^G & \xleftarrow{H_{n-1}} & \overline{B}_{n-1}^G & \cdots & \overline{B}_1^G & \xleftarrow{H_0} & \overline{B}_0^G \\ \uparrow \downarrow \iota_{n+1} & \uparrow \downarrow \psi_{n+1} & \uparrow \downarrow \iota_n & \uparrow \downarrow \psi_n & \uparrow \downarrow \iota_{n-1} & \uparrow \downarrow \psi_{n-1} & \uparrow \downarrow \iota_1 & \uparrow \downarrow \psi_1 & \uparrow \downarrow \iota_0 & \uparrow \downarrow \psi_0 \\ \cdots & \overline{R}_{n+1}^G & \longrightarrow & \overline{R}_n^G & \longrightarrow & \overline{R}_{n-1}^G & \cdots & \overline{R}_1^G & \longrightarrow & \overline{R}_0^G \end{array}$$

### Algorithm 2.3.5.

**Input:** A HAP resolution  $R_*^G$  of group  $G$ .

**Output:**

- The image of the free generator  $[g_1 | \cdots | g_i]$  of  $\overline{B}_i^G$  under the chain map  $\psi_i: \overline{B}_i^G \rightarrow$

$\overline{R}_i^G$ .

- The image of the  $k$ th free generator of  $\overline{R}_i^G$  under the chain map  $\iota_i: \overline{R}_i^G \rightarrow \overline{B}_i^G$ .
- The image of the free generator  $[g_1 | \cdots | g_i]$  of  $\overline{B}_i^G$  under the homotopy map  $H_i: \overline{B}_i^G \rightarrow \overline{B}_{i+1}^G$  satisfying  $\iota_i \psi_i = 1 + \partial_{i+1} H_i + H_{i-1} \partial_i$ .

**Procedure:**

- Applying Algorithm 2.3.4 for  $R_*^G$ , we obtain  $\mathbb{Z}G$ -equivariant chain homotopy equivalence between the bar resolution  $B_*^G$  and the HAP resolution  $R_*^G$ .
- Take the tensor product of this chain homotopy equivalence with  $\mathbb{Z}$ . Note that, if  $e_i$  is a free generator of  $B_i^G$  or  $R_i^G$  then  $mge_i \otimes k = mke_i$ .

# Chapter 3

## Homology of $n$ -types

An  $n$ -type  $X$  is a CW-space with homotopy groups  $\pi_i(X) = 0$  for all  $i > n$ . Up to homotopy equivalence such a space can be specified algebraically by means of a simplicial group  $G_*$  whose Moore complex is trivial in degrees greater than or equal to  $n$ . More precisely, by treating each group  $G_i$  as a category with one object and constructing the nerve  $\mathcal{N}G_i$  one obtains a bisimplicial set  $\mathcal{N}G_*$ . The diagonal of this bisimplicial set,  $\Delta\mathcal{N}G_*$ , is a simplicial set whose geometric realization is a CW-space  $B(G_*)$ . The condition on the Moore complex of  $G_*$  is sufficient to ensure that  $B(G_*)$  is an  $n$ -type. The functor  $B$  induces an equivalence of categories

$$Ho(\text{Simplicial groups with Moore complex trivial in degrees } \geq n) \xrightarrow{\cong} Ho(n\text{-types})$$

where  $Ho(\mathcal{C})$  denotes the category obtained from a category  $\mathcal{C}$  by localizing with respect to those maps in  $\mathcal{C}$ , termed *quasi-isomorphisms*, that induce isomorphisms on homotopy groups. (See [33] for more detail on this general theory.)

In this chapter, we describe an algorithm for computing a chain complex for homology of a simplicial group (see Definition 3.2.5 and Algorithm 3.2.1).

## 3.1 Perturbation Lemma

**Definition 3.1.1.** [12] A *homotopy equivalence data*

$$(L, b) \xleftarrow[p_i]{p} (M, b), h \tag{3.1}$$

consists of the following:

- (i) two chain complexes  $(L, b)$ ,  $(M, b)$  and quasi-isomorphisms  $i, p$  between them;
- (ii) a chain homotopy  $h : ip \simeq 1_M$  (so  $i_n p_n - 1_{M_n} = h_{n-1} b_n + b_{n+1} h_n$  for all  $n$ ).

Note that this definition does not necessarily induce a chain homotopy  $h' : pi \simeq 1_L$ .

*Remark 3.1.1.* In applications  $M$  will denote a massive chain complex such as a bar resolution, and  $L$  will denote a little chain complex such as an explicitly computed HAP resolution.

**Definition 3.1.2.** [12] A *perturbation*  $\delta$  of (3.1) is a sequence of homomorphisms  $\delta_n : M_n \rightarrow M_{n-1}$  such that  $(b_n + \delta_n)(b_{n+1} + \delta_{n+1}) = 0$  for all  $n$ . We call it *small* if

$(1_{M_n} - \delta_{n+1}h_n)$  is invertible for all  $n$ . In this case we put:

$$A_n = (1_{M_{n-1}} - \delta_n h_{n-1})^{-1} \delta_n,$$

and we consider

$$(L, b') \xleftarrow[i']{p'} (M, b + \delta), \quad h' \tag{3.2}$$

with:

$$i'_n = i_n + h_{n-1}A_n i_n, \quad p'_n = p_n + p_n A_{n+1} h_n, \quad h'_n = h_n + h_n A_{n+1} h_n, \quad b'_n = b_n + p_{n-1} A_n i_n$$

for all  $n$ .

*Remark 3.1.2.*  $(1 - x)$  is invertible means  $(1 - x)^{-1} = 1 + x + x^2 + x^3 + \dots + x^{k-1}$  for some  $k$  satisfying  $x^k = 0$ .

*Remark 3.1.3.* If  $(\delta_{n+1}h_n)^k = 0$  for some  $k > 0$  then  $(1_{M_n} - \delta_{n+1}h_n)$  is invertible.

**Theorem 3.1.1.** [12] *If  $\delta$  is a small perturbation of the homotopy equivalence data (3.1), then the perturbed data (3.2) is a homotopy equivalence data.*

A proof of this theorem is given in [12]. Because this paper has not been published yet, we include the proof in this thesis with just minor modifications on [12].

To prove Theorem 3.1.1, we need the following.

**Lemma 3.1.2.** *For any homotopy equivalence data (3.1), any small perturbation  $\delta$  and any  $n \in \mathbb{Z}$ , we have*

$$\delta_n h_{n-1} A_n = A_n h_{n-1} \delta_n = A_n - \delta_n, \tag{3.3}$$

$$(1_{M_n} - \delta_{n+1} h_n)^{-1} = 1_{M_n} + A_{n+1} h_n, \tag{3.4}$$

$$(1_{M_n} - h_{n-1} \delta_n)^{-1} = 1_{M_n} + h_{n-1} A_n, \tag{3.5}$$

$$A_n i_n p_n A_{n+1} + A_n b_{n+1} + b_n A_{n+1} = 0. \tag{3.6}$$

**Proof.** From the definition of  $A_n$ ,  $(1_{M_{n-1}} - \delta_n h_{n-1}) A_n = \delta_n$  which proves  $\delta_n h_{n-1} A_n = A_n - \delta_n$ . Multiplying the identity  $\delta_n h_{n-1} \delta_n = \delta_n - (1_{M_{n-1}} - \delta_n h_{n-1}) \delta_n$  by  $(1_{M_{n-1}} - \delta_n h_{n-1})^{-1}$  from the left we also get  $A_n h_{n-1} \delta_n = A_n - \delta_n$ . These prove (3.3). The relations we have to check in order to prove (3.4) and (3.5) follow immediately from

(3.3). For instance:

$$\begin{aligned}
(1_{M_n} - \delta_{n+1}h_n)(1_{M_n} + A_{n+1}h_n) &= 1_{M_n} + A_{n+1}h_n - \delta_{n+1}h_n - \delta_{n+1}h_nA_{n+1}h_n \\
&= 1_{M_n} + (A_{n+1} - \delta_{n+1} - \delta_{n+1}h_nA_{n+1})h_n \\
&\stackrel{(3.3)}{=} 1_{M_n}. \\
(1_{M_n} - h_{n-1}\delta_n)(1_{M_n} + h_{n-1}A_n) &= 1_{M_n} + h_{n-1}A_n - h_{n-1}\delta_n - h_{n-1}\delta_nh_{n-1}A_n \\
&= 1_{M_n} + h_{n-1}(A_n - \delta_n - \delta_nh_{n-1}A_n) \\
&\stackrel{(3.3)}{=} 1_{M_n}
\end{aligned}$$

To prove (3.6) we use (3.3), (3.4) and the relations  $i_n p_n - 1_{M_n} = h_{n-1}b_n + b_{n+1}h_n$ ,  $(b_n + \delta_n)(b_{n+1} + \delta_{n+1}) = 0$

$$\begin{aligned}
&A_n i_n p_n A_{n+1} + A_n b_{n+1} + b_n A_{n+1} = \\
&= A_n(1_{M_n} + h_{n-1}b_n + b_{n+1}h_n)A_{n+1} + A_n b_{n+1} + b_n A_{n+1} \\
&= A_n A_{n+1} + A_n b_{n+1}(h_n A_{n+1} + 1_{M_{n+1}}) + (A_n h_{n-1} + 1_{M_{n-1}})b_n A_{n+1} \\
&\stackrel{(3.4),(3.5)}{=} A_n A_{n+1} + A_n b_{n+1}(1_{M_{n+1}} - h_n \delta_{n+1})^{-1} + (1_{M_{n-1}} - \delta_n h_{n-1})^{-1} b_n A_{n+1} \\
&= (1_{M_{n-1}} - \delta_n h_{n-1})^{-1} [(1_{M_{n-1}} - \delta_n h_{n-1}) A_n A_{n+1} (1_{M_{n+1}} - h_n \delta_{n+1}) + \\
&\quad (1_{M_{n-1}} - \delta_n h_{n-1}) A_n b_{n+1} + b_n A_{n+1} (1_{M_{n+1}} - h_n \delta_{n+1})] (1_{M_{n+1}} - h_n \delta_{n+1})^{-1} \\
&= (1_{M_{n-1}} - \delta_n h_{n-1})^{-1} [(A_n - \delta_n h_{n-1} A_n)(A_{n+1} - A_{n+1} h_n \delta_{n+1}) + \\
&\quad (A_n - \delta_n h_{n-1} A_n) b_{n+1} + b_n (A_{n+1} - A_{n+1} h_n \delta_{n+1})] (1_{M_{n+1}} - h_n \delta_{n+1})^{-1} \\
&\stackrel{(3.3)}{=} (1_{M_{n-1}} - \delta_n h_{n-1})^{-1} [\delta_n \delta_{n+1} + \delta_n b_{n+1} + b_n \delta_{n+1}] (1_{M_{n+1}} - h_n \delta_{n+1})^{-1} \\
&= (1_{M_{n-1}} - \delta_n h_{n-1})^{-1} [(b_n + \delta_n)(b_{n+1} + \delta_{n+1})] (1_{M_{n+1}} - h_n \delta_{n+1})^{-1} = 0.
\end{aligned}$$

**Proof Theorem 3.1.1** We have to prove various relations:

1)  $(L, b')$  is a chain complex ( i.e.  $b'_n b'_{n+1} = 0$  for all  $n$ ).

$$\begin{aligned}
b'_n b'_{n+1} &= (b_n + p_{n+1} A_n i_n)(b_{n+1} + p_n A_{n+1} i_{n+1}) \\
&= b_n b_{n+1} + b_n p_n A_{n+1} i_{n+1} + p_{n+1} A_n i_n b_{n+1} + p_{n+1} (A_n i_n p_n A_{n+1}) i_{n+1} \\
&\stackrel{(3.6)}{=} b_n p_n A_{n+1} i_{n+1} + p_{n+1} A_n i_n b_{n+1} - p_{n+1} (A_n b_{n+1} + b_n A_{n+1}) i_{n+1} \\
&= (b_n p_n - p_{n+1} b_n) A_{n+1} i_{n+1} + p_{n+1} A_n (i_n b_{n+1} - b_{n+1} i_{n+1}) = 0.
\end{aligned}$$

2)  $i'$  is a chain map ( i.e.  $i'_{n-1} b'_n = (b_n + \delta_n) i_n$  for all  $n$ ).

$$\begin{aligned}
& i'_{n-1}b'_n - (b_n + \delta_n)i_n = \\
&= (i_{n-1} + h_{n-2}A_{n-1}i_{n-1})(b_n + p_{n-1}A_n i_n) - (b_n + \delta_n)(i_n + h_{n-1}A_n i_n) \\
&= i_{n-1}b_n + i_{n-1}p_{n-1}A_n i_n + h_{n-2}A_{n-1}i_{n-1}b_n + h_{n-2}(A_{n-1}i_{n-1}p_{n-1}A_n)i_n \\
&\quad - b_n i_n - b_n h_{n-1}A_n i_n - \delta_n i_n - (\delta_n h_{n-1}A_n)i_n \\
&\stackrel{(3.3),(3.6)}{=} i_{n-1}p_{n-1}A_n i_n + h_{n-2}A_{n-1}i_{n-1}b_n - h_{n-2}(A_{n-1}b_n + b_{n-1}A_n)i_n \\
&\quad - b_n h_{n-1}A_n i_n - \delta_n i_n - (A_n - \delta_n)i_n \\
&= i_{n-1}p_{n-1}A_n i_n - h_{n-2}b_{n-1}A_n i_n - b_n h_{n-1}A_n i_n - A_n i_n \\
&= (i_{n-1}p_{n-1} - h_{n-2}b_{n-1} - b_n h_{n-1} - 1_{M_n})A_n i_n = 0.
\end{aligned}$$

3)  $p'$  is a chain map ( i.e.  $p'_{n-1}(b_n + \delta_n) = b'_n p'_n$  for all  $n$ ).

$$\begin{aligned}
& p'_{n-1}(b_n + \delta_n) - b'_n p'_n = \\
&= (p_{n-1} + p_{n-1}A_n h_{n-1})(b_n + \delta_n) - (b_n + p_{n-1}A_n i_n)(p_n + p_n A_{n+1} h_n) \\
&= p_{n-1}b_n + p_{n-1}\delta_n + p_{n-1}A_n h_{n-1}b_n + p_{n-1}(A_n h_{n-1}\delta_n) \\
&\quad - b_n p_n - b_n p_n A_{n+1} h_n - p_{n-1}A_n i_n p_n - p_{n-1}(A_n i_n p_n A_{n+1})h_n \\
&\stackrel{(3.3),(3.6)}{=} p_{n-1}\delta_n + p_{n-1}A_n h_{n-1}b_n + p_{n-1}(A_n - \delta_n) \\
&\quad - b_n p_n A_{n+1} h_n - p_{n-1}A_n i_n p_n + p_{n-1}(A_n b_{n+1} + b_n A_{n+1})h_n \\
&= p_{n-1}A_n h_{n-1}b_n + p_{n-1}A_n - p_{n-1}A_n i_n p_n + p_{n-1}A_n b_{n+1} h_n \\
&= p_{n-1}A_n (h_{n-1}b_n + 1_{M_n} - i_n p_n + b_{n+1} h_n) = 0.
\end{aligned}$$

(4)  $h'$  is a chain homotopy between  $i'p'$  and  $1_M$  ( i.e.  $i'_n p'_n - 1_{M_n} = h'_{n-1}(b_n + \delta_n) + (b_{n+1} + \delta_{n+1})h'_n$  for all  $n$ ).

$$\begin{aligned}
& i'_n p'_n - 1_{M_n} - h'_{n-1}(b_n + \delta_n) - (b_{n+1} + \delta_{n+1})h'_n = \\
&= (i_n + h_{n-1}A_n i_n)(p_n + p_n A_{n+1} h_n) - 1_{M_n} \\
&\quad - (h_{n-1} + h_{n-1}A_n h_{n-1})(b_n + \delta_n) - (b_{n+1} + \delta_{n+1})(h_n + h_n A_{n+1} h_n) \\
&= \underline{i_n p_n} + i_n p_n A_{n+1} h_n + h_{n-1}A_n i_n p_n + h_{n-1}(A_n i_n p_n A_{n+1})h_n - \underline{1_{M_n}} \\
&\quad - \underline{h_{n-1}b_n} - h_{n-1}\delta_n - h_{n-1}A_n h_{n-1}b_n - h_{n-1}(A_n h_{n-1}\delta_n) \\
&\quad - \underline{b_{n+1}h_n} - b_{n+1}h_n A_{n+1} h_n - \delta_{n+1}h_n - (\delta_{n+1}h_n A_{n+1})h_n \\
&= i_n p_n A_{n+1} h_n + h_{n-1}A_n i_n p_n - h_{n-1}b_n A_{n+1} h_n - h_{n-1}A_n b_{n+1} h_n \\
&\quad - h_{n-1}A_n h_{n-1}b_n - h_{n-1}A_n - b_{n+1}h_n A_{n+1} h_n - A_{n+1} h_n \\
&= h_{n-1}A_n (i_n p_n - b_{n+1}h_n - h_{n-1}b_n 1_{M_n})
\end{aligned}$$

$$+(i_n p_n - h_{n-1} b_n - b_{n+1} h_n - 1_{M_n}) A_{n+1} h_n = 0.$$

(5)  $p'$  and  $i'$  are quasi-isomorphisms From step 4 it follows that  $i'p'$  induces the identity in homology. So it suffices to show that  $i'$  is injective in homology. Assume that  $x \in \text{Ker } b'_n$  and  $i'_n(x) \in \text{Im}(b_{n+1} + \delta_{n+1})$ , so there exists  $y \in M_{n+1}$  satisfying  $i'_n(x) = (b_{n+1} + \delta_{n+1})(y)$ . Hence

$$b_n(x) + p_{n-1} A_n i_n(x) = 0, \quad (3.7)$$

$$i_n(x) + h_{n-1} A_n i_n(x) = b_{n+1}(y) + \delta_{n+1}(y). \quad (3.8)$$

Now we need to prove that  $x \in \text{Im } b'_{n+1}$ .

Applying  $\delta_n$  to (3.8), and replacing  $\delta_n h_{n-1} A_n$  by  $A_n - \delta_n$  (Lemma 3.1.2) and  $\delta_n b_{n+1} + \delta_n \delta_{n+1}$  by  $-b_n \delta_{n+1}$  (because  $(b_n + \delta_n)(b_{n+1} + \delta_{n+1}) = 0$ ), we obtain

$$A_n i_n(x) = -b_n \delta_{n+1}(y). \quad (3.9)$$

With this formula for  $A_n i_n(x)$  plugged into (3.8), we get

$$i_n(x) = b_{n+1}(y) + \delta_{n+1}(y) + h_{n-1} b_n \delta_{n+1}(y), \quad (3.10)$$

and, using  $h_{n-1} b_n = i_n p_n - 1_{M_n} - b_{n+1} h_n$ , we get

$$i_n(x - p_n \delta_{n+1}(y)) = b_{n+1}(y - h_n \delta_{n+1}(y)). \quad (3.11)$$

Next, plug the formula (3.9) for  $A_n i_n(x)$  into (3.7) to get

$$b_n(x) - p_{n-1} b_n \delta_{n+1}(y) = 0 \Leftrightarrow b_n(x) - b_n p_n \delta_{n+1}(y) = 0 \Leftrightarrow b_n(x - p_n \delta_{n+1}(y)) = 0.$$

From (3.11) we have  $i_n(x - p_n \delta_{n+1}(y)) \in \text{Im } b_{n+1}$ . Since  $i$  is a quasi-isomorphism and  $x - p_n \delta_{n+1}(y) \in \text{Ker } b_n$ , we obtain  $x - p_n \delta_{n+1}(y) \in \text{Im } b_{n+1}$ . So, there exists  $z \in L_{n+1}$  such that

$$b_{n+1}(z) = x - p_n \delta_{n+1}(y) \Leftrightarrow x = b_{n+1}(z) + p_n \delta_{n+1}(y) \quad (3.12)$$

Applying  $i_n$  to this formula and using (3.10),

$$\begin{aligned}
& b_{n+1}(y) + \delta_{n+1}(y) + h_{n-1}b_n\delta_{n+1}(y) = i_n p_n \delta_{n+1}(y) + i_n b_{n+1}(z) \\
\Leftrightarrow & i_n p_n \delta_{n+1}(y) + b_{n+1} i_{n+1}(z) - b_{n+1}(y) - \delta_{n+1}(y) - h_{n-1} b_n \delta_{n+1}(y) = 0.
\end{aligned}$$

Using now  $i_n p_n = 1_{M_n} + h_{n-1} b_n + b_{n+1} h_n$ , we deduce that

$$b_{n+1}(i_{n+1}(z) - y + h_n \delta_{n+1}(y)) = 0.$$

Since  $i$  is surjective in homology, we can write

$$i_{n+1}(z) - (1 - h_n \delta_{n+1})(y) = i_{n+1}(\alpha) + b_{n+2}(\beta)$$

for some  $\alpha \in L_{n+1}$  with  $b_{n+1}(\alpha) = 0$ , and some  $\beta \in M_{n+2}$ . Applying  $p_n A_{n+1}$  to this and using  $A_{n+1}(1_{M_{n+1}} - h_n \delta_{n+1}) = \delta_{n+1}$  and  $A_{n+1} b_{n+2} = -b_{n+1} A_{n+2} - A_{n+1} i_{n+1} p_{n+1} A_{n+2}$  (Lemma 3.1.2), we deduce

$$p_n A_{n+1} i_{n+1}(z) = p_n \delta_{n+1}(y) + p_n A_{n+1} i_{n+1}(\alpha) - p_n b_{n+1} A_{n+2}(\beta) - p_n A_{n+1} i_{n+1} p_{n+1} A_{n+2}(\beta).$$

From this we extract  $p_n \delta_{n+1}(y)$  and we plug the result in (3.12). Rearranging the terms we get

$$x = b_{n+1}(z + p_{n+1} A_{n+2}(\beta)) + p_n A_{n+1} i_{n+1}(z + p_{n+1} A_{n+2}(\beta) - \alpha).$$

Since  $b_{n+1}(\alpha) = 0$ , we conclude that  $x = b'_{n+1}(z + p_{n+1} A_{n+2}(\beta) - \alpha)$ . It means  $x \in \text{Im } b'_{n+1}$ .  $\square$

## 3.2 Chain complex for homology of simplicial groups

**Definition 3.2.1.** The functor

$$\mathcal{N} : (\text{Simplicial groups}) \rightarrow (\text{Bisimplicial sets})$$

sends a simplicial group  $G_*$  to the bisimplicial set

$$\mathcal{N}G_* : \begin{array}{c} \cdots \\ \cdots \\ \cdots \\ \cdots \\ \cdots \end{array} \mathcal{N}G_4 \begin{array}{c} \xrightarrow{d} \\ \xrightarrow{d} \\ \xrightarrow{d} \\ \xrightarrow{d} \\ \xrightarrow{d} \end{array} \mathcal{N}G_3 \begin{array}{c} \xrightarrow{d} \\ \xrightarrow{d} \\ \xrightarrow{d} \\ \xrightarrow{d} \\ \xrightarrow{d} \end{array} \mathcal{N}G_2 \begin{array}{c} \xrightarrow{d} \\ \xrightarrow{d} \\ \xrightarrow{d} \\ \xrightarrow{d} \\ \xrightarrow{d} \end{array} \mathcal{N}G_1 \begin{array}{c} \xrightarrow{d} \\ \xrightarrow{d} \\ \xrightarrow{d} \\ \xrightarrow{d} \\ \xrightarrow{d} \end{array} \mathcal{N}G_0$$

where  $\mathcal{N}G_n$  is the nerve of the group  $G_n$  for all  $n \geq 0$ .

**Definition 3.2.2.** The functor

$$\Delta: (\text{Bisimplicial sets}) \rightarrow (\text{Simplicial sets})$$

sends a bisimplicial set  $X_{**}$

$$\begin{array}{ccccccc}
 & & d_i^h & & & d_i^h & \\
 & & \cdots & & \cdots & \cdots & \\
 X_{n,n} & \xleftarrow{\cdots} & X_{n-1,n} & \cdots & X_{1,n} & \xleftarrow{\cdots} & X_{0,n} \\
 \uparrow s_i^v & \downarrow d_i^v & \uparrow s_i^v & \downarrow d_i^v & \uparrow s_i^v & \downarrow d_i^v & \uparrow s_i^v \\
 & & d_i^h & & d_i^h & & d_i^h \\
 & & \cdots & & \cdots & & \cdots \\
 X_{n,n-1} & \xleftarrow{\cdots} & X_{n-1,n-1} & \cdots & X_{1,n-1} & \xleftarrow{\cdots} & X_{0,n-1} \\
 \uparrow s_i^v & \downarrow d_i^v & \uparrow s_i^v & \downarrow d_i^v & \uparrow s_i^v & \downarrow d_i^v & \uparrow s_i^v \\
 & & d_i^h & & d_i^h & & d_i^h \\
 & & \cdots & & \cdots & & \cdots \\
 X_{n,1} & \xleftarrow{\cdots} & X_{n-1,1} & \cdots & X_{1,1} & \xleftarrow{\cdots} & X_{0,1} \\
 \uparrow s_i^v & \downarrow d_i^v & \uparrow s_i^v & \downarrow d_i^v & \uparrow s_i^v & \downarrow d_i^v & \uparrow s_i^v \\
 & & d_i^h & & d_i^h & & d_i^h \\
 & & \cdots & & \cdots & & \cdots \\
 X_{n,0} & \xleftarrow{\cdots} & X_{n-1,0} & \cdots & X_{1,0} & \xleftarrow{\cdots} & X_{0,0} \\
 & & s_i^h & & s_i^h & & s_i^h
 \end{array}$$

to  $\Delta X_{**}$  where  $\Delta_n X_{**} := X_{n,n}$ ,  $d_i := d_i^v d_i^h$  and  $s_i := s_i^v s_i^h$  for all  $n \geq 0, 0 \leq i \leq n$ .

**Definition 3.2.3.** The functor

$$\mathcal{F}: (\text{Simplicial sets}) \rightarrow (\text{Simplicial abelian groups})$$

sends a simplicial set  $X_*$  to  $\mathcal{F}X_*$  where  $\mathcal{F}X_*$  is the free abelian group generated by  $X_n$  for all  $n \geq 0$ .

**Definition 3.2.4.** The functor

$$\mathcal{A}: (\text{Simplicial abelian groups}) \rightarrow (\text{Chain complexes})$$

sends a simplicial abelian group  $G_*$  to  $\mathcal{A}G_*$ , the alternating chain complex of  $G_*$ .

We now come to the main object of study in this thesis.

**Definition 3.2.5.** For any simplicial group  $G_*$ , the *integral homology* of  $G_*$  is defined by

$$H_n(G_*, \mathbb{Z}) := H_n(\mathcal{A}\mathcal{F}\Delta\mathcal{N}G_*) \text{ for all } i \geq 0.$$

*Remark 3.2.1.* Note that, in topology, the integral homology of a simplicial group

$G_*$  is also defined by

$$H_n(G_*, \mathbb{Z}) := H_n(B(G_*), \mathbb{Z}) \text{ for all } n \geq 0,$$

where  $B(G_*)$  is the classifying space of  $G_*$ .

**Proposition 3.2.1.** *The  $n$ th integral homology  $H_n(-, \mathbb{Z})$  is a covariant functor from the category of simplicial groups to the category of abelian groups.*

**Proof.** It is obvious.

**Theorem 3.2.2** (well-known). *Let  $f: G_* \rightarrow G'_*$  be a morphism of simplicial groups. If  $f$  is a weak equivalence then  $f$  induces isomorphisms*

$$H_n(f): H_n(G_*, \mathbb{Z}) \xrightarrow{\cong} H_n(G'_*, \mathbb{Z}) \text{ for all } n \geq 0.$$

**Corollary 3.2.3.** *Let  $G_*$  and  $G'_*$  be simplicial groups. If  $G_*$  and  $G'_*$  are weakly equivalent then*

$$H_n(G_*, \mathbb{Z}) \cong H_n(G'_*, \mathbb{Z}) \text{ for all } n \geq 0.$$

**Lemma 3.2.4.** *Let  $X_{**}$  be a bisimplicial set. Then  $\mathcal{F}\Delta X_{**}$  and  $\Delta\mathcal{F}X_{**}$  are the same.*

**Proof.** It is obvious.

**Theorem 3.2.5.** [29] *Let  $X_{**}$  be a bisimplicial abelian group. Then the chain complex  $\mathcal{A}\Delta X_{**}$  and  $\text{Tot}(\mathcal{A}X_{**})$  are chain homotopy equivalent.*

For any simplicial group  $G_*$ , we apply the bar complex construction to the terms in  $G_*$  and obtain a chain complex  $\overline{B}_*^{G_*}$  of simplicial abelian groups. Taking the alternating chain complex of each simplicial abelian group  $\overline{B}_q^{G_*}$  yields a bicomplex which we denote by  $\mathcal{A}\overline{B}_*^{G_*}$ .

**Lemma 3.2.6.** *Let  $G$  be a group. Then  $\mathcal{A}\mathcal{F}NG$  and the bar complex  $\overline{B}_*^G$  are the same.*

**Proof.** It is obvious.

**Theorem 3.2.7.** *Let  $G_*$  be a simplicial group. Then*

$$H_n(G_*, \mathbb{Z}) \cong H_n(\text{Tot}(\mathcal{A}\overline{B}_*^{G_*})) \text{ for all } n \geq 0.$$

**Proof.** To prove this theorem, we use the following diagram.

$$\begin{array}{c}
 \begin{array}{c}
 \Delta \mathcal{N}G_* \\
 \nearrow^{\text{diagonal}} \quad \searrow^{\text{free abelian}} \\
 G_* \xrightarrow{\text{nerve}} \mathcal{N}G_* \quad \mathcal{F}\Delta \mathcal{N}G_* \xrightarrow{\text{alternate}} \mathcal{A}\mathcal{F}\Delta \mathcal{N}G_* \\
 \searrow^{\text{free abelian}} \quad \nearrow^{\text{diagonal}} \\
 \mathcal{F}\mathcal{N}G_* \quad \Delta \mathcal{F}\mathcal{N}G_* \xrightarrow{\text{alternate}} \mathcal{A}\Delta \mathcal{F}\mathcal{N}G_* \\
 \searrow^{\text{alternate}} \\
 \mathcal{A}\mathcal{F}\mathcal{N}G_* \xrightarrow{\text{Tot}} \text{Tot}(\mathcal{A}\mathcal{F}\mathcal{N}G_*) \\
 \parallel \quad \parallel \\
 \mathcal{A}\overline{B}_*^{G_*} \xrightarrow{\text{Tot}} \text{Tot}(\mathcal{A}\overline{B}_*^{G_*})
 \end{array} \\
 \text{Lemma 3.2.4} \\
 \text{Theorem 3.2.5} \\
 \text{Lemma 3.2.6}
 \end{array}$$

From this diagram, we have  $\mathcal{A}\mathcal{F}\Delta \mathcal{N}G_*$  and  $\text{Tot}(\mathcal{A}\overline{B}_*^{G_*})$  are chain homotopy equivalent. This implies  $H_n(\mathcal{A}\mathcal{F}\Delta \mathcal{N}G_*) \cong H_n(\text{Tot}(\mathcal{A}\overline{B}_*^{G_*}))$  for all  $n \geq 0$  or

$$H_n(G_*, \mathbb{Z}) \cong H_n(\text{Tot}(\mathcal{A}\overline{B}_*^{G_*})) \text{ for all } n \geq 0. \quad \square$$

We consider  $\mathcal{A}\overline{B}_*^{G_*}$  as a bicomplex  $(\overline{B}_q^{G_p}, \partial_q^p, \delta_q^p)_{p,q \geq 0}$  with vertical homomorphisms  $\partial_q^p$  and horizontal homomorphisms  $\delta_q^p$ .

$$\begin{array}{ccccccc}
 & & & & & & \uparrow q \\
 \overline{B}_3^{G_3} & \xrightarrow{\delta_3^3} & \overline{B}_3^{G_2} & \xrightarrow{\delta_3^2} & \overline{B}_3^{G_1} & \xrightarrow{\delta_3^1} & \overline{B}_3^{G_0} \\
 \partial_3^3 \downarrow & & \partial_3^2 \downarrow & & \partial_3^1 \downarrow & & \partial_3^0 \downarrow \\
 \overline{B}_2^{G_3} & \xrightarrow{\delta_2^3} & \overline{B}_2^{G_2} & \xrightarrow{\delta_2^2} & \overline{B}_2^{G_1} & \xrightarrow{\delta_2^1} & \overline{B}_2^{G_0} \\
 \partial_2^3 \downarrow & & \partial_2^2 \downarrow & & \partial_2^1 \downarrow & & \partial_2^0 \downarrow \\
 \overline{B}_1^{G_3} & \xrightarrow{\delta_1^3} & \overline{B}_1^{G_2} & \xrightarrow{\delta_1^2} & \overline{B}_1^{G_1} & \xrightarrow{\delta_1^1} & \overline{B}_1^{G_0} \\
 \partial_1^3 \downarrow & & \partial_1^2 \downarrow & & \partial_1^1 \downarrow & & \partial_1^0 \downarrow \\
 \leftarrow p \quad \overline{B}_0^{G_3} & \xrightarrow{\delta_0^3} & \overline{B}_0^{G_2} & \xrightarrow{\delta_0^2} & \overline{B}_0^{G_1} & \xrightarrow{\delta_0^1} & \overline{B}_0^{G_0}
 \end{array}$$

For each  $p \geq 0$ , the column  $(\overline{B}_*^{G_p}, \partial^p)$  is the bar complex of group  $G_p$ . Applying

Algorithm 2.3.5 we construct a homotopy equivalence data

$$(\overline{R}_*^{G_p}, d^p) \xleftarrow[\iota^p]{\psi^p} (\overline{B}_*^{G_p}, \partial^p), H^p.$$

where  $(\overline{R}_*^{G_p}, d^p)$  is a HAP complex of  $G_p$ .

We now build a chain complex  $T = (T_n, \partial_n)_{n \geq 0}$  as the total complex of the bicomplex  $(\overline{B}_q^{G_p}, \partial_q^p, 0)_{p, q \geq 0}$  where the horizontal homomorphisms are set equal to zero. This means that  $T$  is given by

$$T_n = \bigoplus_{p+q=n} \overline{B}_q^{G_p}$$

with differential homomorphism  $\partial_n(x) = \partial_q^p(x)$  for  $x \in \overline{B}_q^{G_p}$ .

In the same way, we also build a chain complex  $HT = (HT_n, d_n)_{n \geq 0}$  as the total complex of the bicomplex  $(\overline{R}_q^{G_p}, d_q^p, 0)_{p, q \geq 0}$ . Thus  $HT$  is given by

$$HT_n = \bigoplus_{p+q=n} \overline{R}_q^{G_p}$$

with differential homomorphism  $d_n(y) = d_q^p(y)$  for  $y \in \overline{R}_q^{G_p}$ .

For each  $p \geq 0$ , we have the homotopy equivalence data

$$(\overline{R}_*^{G_p}, d^p) \xleftarrow[\iota^p]{\psi^p} (\overline{B}_*^{G_p}, \partial^p), H^p.$$

This implies a homotopy equivalence data

$$(HT_*, d) \xleftarrow[\iota]{\psi} (T_*, \partial), H \tag{3.13}$$

with

- $\iota_n(y) = \iota_q^p(y)$  for all  $y \in \overline{R}_q^{G_p}, p + q = n$ ,
- $\psi_n(x) = \psi_q^p(x)$  for all  $x \in \overline{B}_q^{G_p}, p + q = n$ ,
- $H_n(x) = H_q^p(x)$  for all  $x \in \overline{B}_q^{G_p}, p + q = n$ .

The homotopy equivalence data 3.13 is illustrated in the following diagram

$$\begin{array}{cccccccc}
& & & \overset{q}{\uparrow} & & & & \overset{q}{\uparrow} \\
& \overline{R}_3^{G_3} & \overline{R}_3^{G_2} & \overline{R}_3^{G_1} & \overline{R}_3^{G_0} & \overline{B}_3^{G_3} & \overline{B}_3^{G_2} & \overline{B}_3^{G_1} & \overline{B}_3^{G_0} \\
& d_3^3 \downarrow & d_3^2 \downarrow & d_3^1 \downarrow & d_3^0 \downarrow & \partial_3^3 \downarrow & \partial_3^2 \downarrow & \partial_3^1 \downarrow & \partial_3^0 \downarrow \\
& \overline{R}_2^{G_3} & \overline{R}_2^{G_2} & \overline{R}_2^{G_1} & \overline{R}_2^{G_0} & \overline{B}_2^{G_3} & \overline{B}_2^{G_2} & \overline{B}_2^{G_1} & \overline{B}_2^{G_0} \\
& d_2^3 \downarrow & d_2^2 \downarrow & d_2^1 \downarrow & d_2^0 \downarrow & \partial_2^3 \downarrow & \partial_2^2 \downarrow & \partial_2^1 \downarrow & \partial_2^0 \downarrow \\
& \overline{R}_1^{G_3} & \overline{R}_1^{G_2} & \overline{R}_1^{G_1} & \overline{R}_1^{G_0} & \overline{B}_1^{G_3} & \overline{B}_1^{G_2} & \overline{B}_1^{G_1} & \overline{B}_1^{G_0} \\
& d_1^3 \downarrow & d_1^2 \downarrow & d_1^1 \downarrow & d_1^0 \downarrow & \partial_1^3 \downarrow & \partial_1^2 \downarrow & \partial_1^1 \downarrow & \partial_1^0 \downarrow \\
\overset{p}{\leftarrow} & \overline{R}_0^{G_3} & \overline{R}_0^{G_2} & \overline{R}_0^{G_1} & \overline{R}_0^{G_0} & \overline{B}_0^{G_3} & \overline{B}_0^{G_2} & \overline{B}_0^{G_1} & \overline{B}_0^{G_0}
\end{array}$$

$\psi_2^1$  (red arrow from  $\overline{R}_2^{G_0}$  to  $\overline{B}_2^{G_3}$ )  
 $H_1^2$  (red arrow from  $\overline{B}_2^{G_2}$  to  $\overline{B}_2^{G_1}$ )  
 $\iota_1^2$  (blue arrow from  $\overline{R}_1^{G_1}$  to  $\overline{B}_1^{G_2}$ )

We denote by  $\delta$  a sequence of homomorphisms  $\delta_n: T_n \rightarrow T_{n-1}$  defined by  $\delta_n(x) = \delta_n^p(x)$  for  $x \in \overline{B}_q^{G_p}$ . It is easy to see  $\delta$  is a perturbation of (3.13). Moreover, for all  $n \geq 0$ ,  $(\delta_{n+1}H_n)^{n+1} = 0$ . This implies that  $\delta$  is a small perturbation and

$$(1_{T_n} - \delta_{n+1}H_n)^{-1} = 1_{T_n} + \delta_{n+1}H_n + \cdots + (\delta_{n+1}H_n)^n.$$

Applying Theorem 3.1.1, we have a homotopy equivalence data

$$(HT_*, d') \xleftarrow[\iota']{\psi'} (T_*, \partial + \delta), H'$$

with

$$\bullet d'_n = d_n + \psi_{n-1}\delta_n\iota_n + \psi_{n-1}(\delta_n H_{n-1})\delta_n\iota_n + \cdots + \psi_{n-1}(\delta_n H_{n-1})^{n-1}\delta_n\iota_n, \quad (3.14)$$

$$\bullet \iota'_n = \iota_n + H_{n-1}\delta_n\iota_n + H_{n-1}(\delta_n H_{n-1})\delta_n\iota_n + \cdots + H_{n-1}(\delta_n H_{n-1})^{n-1}\delta_n\iota_n, \quad (3.15)$$

$$\bullet \psi'_n = \psi_n + \psi_n\delta_{n+1}H_n + \psi_n(\delta_{n+1}H_n)\delta_{n+1}H_n + \cdots + \psi_n(\delta_{n+1}H_n)^n\delta_{n+1}H_n, \quad (3.16)$$

$$\bullet H'_n = H_n + H_n\delta_{n+1}H_n + H_n(\delta_{n+1}H_n)\delta_{n+1}H_n + \cdots + H_n(\delta_{n+1}H_n)^n\delta_{n+1}H_n. \quad (3.17)$$

By using Formula 3.14, we deduce the following theorem

**Theorem 3.2.8.** *Suppose that  $G_*$  is a simplicial group and that we have a homotopy equivalence data  $(\overline{R}_*^{G_p}, d) \xleftarrow[\iota']{\psi} (\overline{B}_*^{G_p}, \partial), H$  for each  $p \geq 0$ . Then the total complex  $\text{Tot}(\mathcal{A}\overline{B}_*^{G_*})$  is chain homotopic to a chain complex  $(K_*, d')$  with*

$$K_n = \bigoplus_{p+q=n} \overline{R}_q^{G_p}$$



**Procedure:**

- For each  $p \geq 0$ , we use Algorithm 2.3.5 to construct a homotopy equivalence data

$$(\overline{R}_*^{G_p}, d) \xleftrightarrow[\iota]{\psi} (\overline{B}_*^{G_p}, \partial), H$$

- Using formulas in Theorem 3.2.8, we compute the dimension of  $K_i$  and the image of the  $k$ th generator of  $K_i$  under the chain map  $d'_i$ .

**Definition 3.2.7.** For any field  $\mathbb{K}$  we let

$$\mathbb{K}: (\text{Simplicial sets}) \rightarrow (\text{Simplicial vector spaces})$$

sends a simplicial set  $X_*$  to simplicial vector space  $\mathbb{K}X_*$  where  $\mathbb{K}_n X_*$  is the vector space over  $\mathbb{K}$  with basis  $X_n$ .

**Definition 3.2.8.** Let  $G_*$  be a simplicial group and  $\mathbb{K}$  be a field. The homology of  $G_*$  with coefficients in  $\mathbb{K}$  is defined by

$$H_n(G_*, \mathbb{K}) := H_n(\mathcal{A}\mathbb{K}\Delta\mathcal{N}G_*) \text{ for all } n \geq 0.$$

**Proposition 3.2.9.** *The  $n$ th homology  $H_n(-, \mathbb{K})$  is a covariant functor from the category of simplicial groups to the category of vector spaces over  $\mathbb{K}$ .*

**Proof.** It is obvious.

**Proposition 3.2.10.** *Let  $G_*$  be a simplicial group and  $K$  be a chain complex for homology of  $G_*$  obtained from Algorithm 3.2.1. Then*

$$H_n(G_*, \mathbb{F}_p) \cong H_n(K \otimes_{\mathbb{Z}} \mathbb{F}_p) \text{ for all } n \geq 0.$$

**Proof.** It is obvious.

## Chapter 4

# Eilenberg-Mac Lane spaces

## 4.1 Construction of Eilenberg-Mac Lane simplicial groups

For any abelian group  $A$  and  $n \geq 2$ , the Eilenberg-Mac Lane space  $K(A, n)$  is considered as a special case of  $n$ -types. It can be modeled algebraically by a simplicial group  $G_*$  with  $\pi_{n-1}G_* = A, \pi_i G_* = 0$  for  $i \neq n - 1$ . In this section, we construct the simplicial group  $G_*$  and use it for computing the integral homology of the Eilenberg-Mac Lane space  $K(A, n)$ .

**Definition 4.1.1.** Let  $A$  be an abelian group and  $n \geq 2$ . An *Eilenberg-Mac Lane simplicial group* is a simplicial group  $K$  such that  $\pi_{n-1}K = A$  and  $\pi_i K = 0$  for  $i \neq n - 1$ . Such a simplicial group is denoted by  $K(A, n)$ .

Let  $\mathbf{\Delta}$  the category whose objects are the finite ordered sets  $[n] = \{0, 1, \dots, n\}$  and whose morphisms are nondecreasing monotone functions. A simplicial object  $A$  in a category  $\mathcal{C}$  can be viewed as a contravariant functor from  $\mathbf{\Delta}$  to  $\mathcal{C}$ , that is,  $A: \mathbf{\Delta}^{\text{op}} \rightarrow \mathcal{C}$ . Every map  $\theta^*: A_n \rightarrow A_m$  corresponds to a map  $\theta: [m] \rightarrow [n]$ .

Let  $C = (C_n, d_n)_{n \geq 0}$  be a chain complex of abelian groups. Using a method of Dold and Kan (see [29]) we construct a simplicial abelian group  $G_*$  such that the Moore complex of  $G_*$  is the chain complex  $C$ . More precisely,

- $G_n = \bigoplus_{[n] \rightarrow [k]} C_k$  where  $[n] \rightarrow [k]$  ranges over all surjective maps from  $[n]$  to  $[k]$ , for  $0 \leq k \leq n$ .

- The map

$$\theta^*: \bigoplus_{[n] \rightarrow [k]} C_k \rightarrow \bigoplus_{[m] \rightarrow [r]} C_r$$

associated to the map  $\theta: [m] \rightarrow [n]$  given on the summand corresponding to  $\sigma: [n] \rightarrow [k]$  by the composite

$$C_k \xrightarrow{d^*} C_s \xrightarrow{in_t} \bigoplus_{[m] \rightarrow [r]} C_r$$

where

$$[m] \xrightarrow{t} [s] \xrightarrow{d} [k]$$

is the epic-monic factorization of the composite

$$[m] \xrightarrow{\theta} [n] \xrightarrow{\sigma} [k],$$

the map  $in_t$  sends  $C_s$  by identity map to the copy of  $C_s$  indexed by the epimorphism  $t : [m] \rightarrow [s]$ , and

$$d^* = \begin{cases} d_k & \text{if } s = k - 1, \\ 0 & \text{if } s \neq k - 1. \end{cases}$$

**Definition 4.1.2.** [29] The functor

$$\Gamma: \mathbf{Ch}_+ \rightarrow s\mathbf{Ab}$$

sends a chain complex of abelian groups  $C = (C_n, d_n)_{n \geq 0}$  to a simplicial abelian group  $G_*$  using the above recipe.

**Definition 4.1.3.** [29] The functor

$$M: s\mathbf{Ab} \rightarrow \mathbf{Ch}_+$$

sends a simplicial abelian group  $G_*$  to  $MG_*$ , the Moore complex of  $G_*$ .

**Theorem 4.1.1.** [29] (*Dold - Kan Correspondence*) *The functors*

$$\Gamma: \mathbf{Ch}_+ \rightarrow s\mathbf{Ab} \text{ and } M: s\mathbf{Ab} \rightarrow \mathbf{Ch}_+$$

*form an isomorphism of categories.*

**Theorem 4.1.2** (well-known). *Let  $A$  be an abelian group and  $n \geq 2$ . If  $K(A, n)$  and  $K'(A, n)$  are Eilenberg-Mac Lane simplicial groups then*

$$H_m(K(A, n), \mathbb{Z}) \cong H_m(K'(A, n), \mathbb{Z}) \text{ for all } m \geq 0.$$

**Proof.** This result is well-known. However, for completeness, we outline a proof in the case where  $K(A, n)$  and  $K'(A, n)$  are abelian simplicial groups. Since  $A$  is an abelian group, there exists a free abelian group  $F$  such that  $F/R \cong A$  with  $R$  a subgroup of  $F$ .

We consider the following chain complex

$$MA: \quad \cdots \rightarrow 0 \rightarrow \cdots \rightarrow 0 \rightarrow R \xrightarrow{i} F \rightarrow 0 \rightarrow \cdots \rightarrow 0.$$

$\uparrow$   
 $n-1$

By applying the functor  $\Gamma$ , we obtain the simplicial abelian group  $G_* = \Gamma(MA)$ .

Since  $K(A, n)$  is an Eilenberg-Mac Lane abelian simplicial group, the Moore complex of  $K(A, n)$  have the form

$$MK: \quad \cdots \rightarrow C_n \xrightarrow{d_n} C_{n-1} \xrightarrow{d_{n-1}} C_{n-2} \xrightarrow{d_{n-2}} \cdots \xrightarrow{d_2} C_1 \xrightarrow{d_1} C_0$$

with  $\text{Ker } d_{n-1}/\text{Im } d_n = A$  and  $\text{Ker } d_k/\text{Im } d_{k+1} = 0$  for  $k \neq n-1$ .

The isomorphism  $F/R \cong A$  implies that there exists an isomorphism  $h: F/R \rightarrow \text{Ker } d_{n-1}/\text{Im } d_n$ . This lifts to a homomorphism  $\tilde{h}: F \rightarrow \text{Ker } d_{n-1}$  with  $\tilde{h}(R) \subset \text{Im } d_n$ .

We consider the following diagram

$$\begin{array}{ccccccc} 0 & \longrightarrow & R & \xrightarrow{i} & F & \longrightarrow & 0 \\ & & \downarrow \tilde{h}_R & & \downarrow \tilde{h} & & \\ C_{n+1} & & \text{Im } d_n & \xrightarrow{i} & \text{Ker } d_{n-1} & & C_{n-2} \\ & \searrow d_{n+1} & \uparrow d_n & & \downarrow j & \nearrow d_{n-1} & \\ & & C_n & \xrightarrow{d_n} & C_{n-1} & & \end{array}$$

with  $\tilde{h}_R$  is the restriction of  $\tilde{h}$  to  $R$  and  $i, j$  are inclusions.

Since  $R$  is a subgroup of free abelian group  $F$ ,  $R$  is also a free abelian group. So there exists a homomorphism  $g: R \rightarrow C_n$  such that  $d_n g = \tilde{h}_R$ .

We prove  $MA$  and  $MK$  are quasi-isomorphic by constructing a chain map  $f$  between  $MA$  and  $MK$  as follows

$$f_k = \begin{cases} g & \text{if } k = n, \\ j\tilde{h} & \text{if } k = n-1, \\ 0 & \text{if } k \neq n, n-1. \end{cases}$$

It is easy to see that  $f$  is a quasi-isomorphism.

By applying the same argument to the Moore complex  $MK'$  of  $K'(A, n)$ , we also deduce  $MA$  and  $MK'$  are quasi-isomorphic. This implies  $MK$  is quasi-isomorphic

to  $MK'$ . From Theorem 4.1.1, we have  $K(A, n)$  and  $K'(A, n)$  are weakly equivalent. By using Corollary 3.2.3, we have

$$H_m(K(A, n), \mathbb{Z}) \cong H_m(K'(A, n), \mathbb{Z}) \text{ for all } m \geq 0. \quad \square$$

Now we apply the functor  $\Gamma$  to the chain complex

$$\cdots \rightarrow 0 \rightarrow \cdots \rightarrow 0 \rightarrow \underset{\substack{\uparrow \\ n-1}}{A} \rightarrow 0 \rightarrow \cdots \rightarrow 0,$$

we obtain an algorithm for constructing an Eilenberg-Mac Lane simplicial group  $K(A, n)$ . In fact, it will be a simplicial abelian group.

**Algorithm 4.1.1.**

**Input:** An abelian group  $A$  and two integers  $n \geq 2, l \geq 1$ .

**Output:** The Eilenberg-Mac Lane simplicial group  $K(A, n)$  of length  $l$  in the form:

- The groups  $K_i$  ( $0 \leq i \leq l$ ).
- The face maps  $d_j: K_i \rightarrow K_{i-1}$  ( $0 \leq j \leq i$ ).
- The degeneracy maps  $s_j: K_i \rightarrow K_{i+1}$  ( $0 \leq j \leq i$ ).

**Procedure:** Implement the method of Dold and Kan.

**Example 4.1.1.** The following GAP session illustrates how to compute the integral homology of an Eilenberg-Mac Lane simplicial group  $K(\mathbb{Z}_3, 2)$ .

```
gap> A:=CyclicGroup(3);;
gap> K:=EilenbergMacLaneSimplicialGroup(A,2,8);
Simplicial group of length 8
gap> C:=ChainComplexOfSimplicialGroup(K);
Chain complex of length 8 in characteristic 0
gap> Homology(C,6);;
[9]
gap> Homology(C,7);;
[3]
```

These commands took 33 seconds on a Windows Dual core 2.8 GHz desktop with 2GB RAM.

In the PhD thesis of Alain Clément [11], he introduced a method to compute the integral homology of the Eilenberg-Mac Lane space  $K(\mathbb{Z}_{2^s}, n)$  for all  $n \geq 1, s \geq 1$ . In addition, he listed the values of the integral homology of  $K(\mathbb{Z}_2, 2)$ ,  $K(\mathbb{Z}_2, 3)$ ,  $K(\mathbb{Z}_4, 2)$ ,  $K(\mathbb{Z}_4, 3)$  up to degree 200. We have used these values to test our results at degree  $0 \leq i \leq 9$ .

For any be a homomorphism of abelian groups  $f: A \rightarrow A'$ , we have the following chain map

$$\begin{array}{ccccccccccc} \cdots & \longrightarrow & 0 & \longrightarrow & \cdots & \longrightarrow & 0 & \longrightarrow & A & \longrightarrow & 0 & \longrightarrow & \cdots & \longrightarrow & 0 \\ & & & & & & & & \downarrow & & f & & & & \\ \cdots & \longrightarrow & 0 & \longrightarrow & \cdots & \longrightarrow & 0 & \longrightarrow & A' & \longrightarrow & 0 & \longrightarrow & \cdots & \longrightarrow & 0 \\ & & & & & & & & \uparrow & & n-1 & & & & \end{array}$$

Applying the functor  $\Gamma$  to the chain map, we obtain a morphism  $f_*: K(A, n) \rightarrow K(A', n)$ . We also give an algorithm to compute this morphism.

**Algorithm 4.1.2.**

**Input:** A homomorphism of abelian groups  $f: A \rightarrow A'$  and two integers  $n \geq 2, l \geq 1$ .

**Output:** The morphism  $f_*: K(A, n) \rightarrow K(A', n)$  of simplicial groups of length  $l$  in the form:

- Simlicial group  $K = K(A, n)$ .
- Simlicial group  $K' = K(A', n)$ .
- Group homomorphisms  $f_i: K_i \rightarrow K'_i$  ( $0 \leq i \leq l$ ).

**Procedure:** We implement the method of Dold and Kan.

**Example 4.1.2.** Let  $f: \mathbb{Z}_4 \rightarrow \mathbb{Z}_2$  give by  $\overline{m} \mapsto \overline{m \bmod 2}$ . The following GAP session illustrates how to compute the morphism

$$f_*: K(\mathbb{Z}_4, 2) \rightarrow K(\mathbb{Z}_2, 2).$$

```

gap> Z4:=CyclicGroup(4);;
gap> a:=Z4.1;;          ## Z4=<a>
gap> Z2:=CyclicGroup(2);;
gap> b:=Z2.1;;          ## Z2=<b>
gap> f:=GroupHomomorphismByImages(Z4,Z2,[a],[b]);;
gap> Kf:=EilenbergMacLaneSimplicialGroup(f,2,5);
Morphism of simplicial groups of length 5

```

These commands took 1 second.

## 4.2 Small chain complex for homology of $K(\mathbb{Z}_m, 2)$

The chain complex constructed for a simplicial group using Algorithm 3.2.1 is typically unnecessarily large. Given any chain complex  $(R_*, d)$  of finitely generated free abelian groups there are a number of ways in which one might attempt to produce a chain homotopy equivalence  $R_* \simeq R'_*$  where  $(R'_*, d')$  is a chain complex of free abelian groups of lower ranks. We shall describe one such algorithm which is based on idea of Pawel Dlotko, T. Kaczynski and Marian Mrozek in their paper [14]. It is extremely simple to implement yet surprisingly effective in many cases.

Let us denote by  $e_i^n$  the free generators of  $R_n$ . Let us define a pair  $(e_i^n, e_j^{n-1})$  to be *redundant* if  $d(e_i^n) = \pm e_j^{n-1}$ . A redundant pair generates a sub chain complex

$$\cdots \rightarrow 0 \rightarrow 0 \rightarrow \cdots \rightarrow 0 \rightarrow \langle e_i^n \rangle \rightarrow \langle e_j^{n-1} \rangle \rightarrow 0 \rightarrow \cdots$$

of  $R_*$  with trivial homology. The long exact homology sequence arising from a short exact sequence of chain complexes implies that the quotient chain map

$$\begin{array}{ccccccc} \cdots & \longrightarrow & R_{n+1} & \longrightarrow & R_n & \longrightarrow & R_{n-1} & \longrightarrow & R_{n-2} & \longrightarrow & \cdots \\ & & \downarrow & & \Pi_n \downarrow & & \Pi_{n-1} \downarrow & & \downarrow & & \\ \cdots & \longrightarrow & R_{n+1} & \longrightarrow & R_n / \langle e_i^n \rangle & \longrightarrow & R_{n-1} / \langle e_j^{n-1} \rangle & \longrightarrow & R_{n-2} & \longrightarrow & \cdots \end{array}$$

induces isomorphisms on homology. Moreover, the quotient chain complex is a chain complex of free abelian groups and hence  $\Pi_*$  must be a chain homotopy equivalence.

**Algorithm 4.2.1.****Input:** A finite dimensional chain complex  $R_*$ .**Output:** A reduced chain complex  $R'_*$  chain homotopy equivalent to  $R_*$ .**Procedure:** Repeatedly search for and remove redundant pairs as defined above.

An implementation of Algorithm 4.2.1 is available in the HAP package [20]. To illustrate its performance we consider the Eilenberg-Mac Lane space simplicial group  $K(\mathbb{Z}_2, 2)$ . Let  $R_*$  denote the chain complex for  $K(\mathbb{Z}_2, 2)$  constructed using Algorithm 4.1.1 and Algorithm 3.2.1. When Algorithm 4.2.1 is applied to this chain complex  $R_*$  it yields a chain homotopy equivalence  $R_* \simeq R'_*$  where the ranks of  $R_i$  and  $R'_i$  are listed in the following table for low degrees.

$i$	0	1	2	3	4	5	6	7	8	9	10
rank( $R_i$ )	1	1	2	4	8	16	32	64	128	256	512
rank( $R'_i$ )	1	0	1	1	2	3	5	8	13	21	34

Experimental evidence seems to suggest that Algorithm 4.2.1 yields a free abelian chain complex for  $K(\mathbb{Z}_m, 2)$ ,  $m \geq 2$ , whose terms have ranks equal to the Fibonacci numbers. We remark that Clemens Berger [4] has proved the existence of a CW-complex of type  $K(\mathbb{Z}_2, 2)$  whose terms have ranks equal to the Fibonacci numbers.

# Chapter 5

## Homology of 2-types

## 5.1 Group theoretic examples of crossed modules

**Definition 5.1.1.** A *crossed module* is a group homomorphism  $\partial: M \rightarrow P$  together with a group action  $P$  on  $M$ , denoted by  $(p, m) \rightarrow {}^p m$  and satisfying the following conditions:

$$(i) \quad \partial({}^p m) = p\partial(m)p^{-1};$$

$$(ii) \quad \partial({}^{m}m') = mm'm^{-1},$$

for all  $p \in P, m, m' \in M$ .

**Example 5.1.1.**

1. If  $G$  is any group with normal subgroup  $N$  then the inclusion  $i: N \hookrightarrow G$  is a crossed module with action  ${}^g n := gng^{-1}$ .
2. If  $G$  is any group and  $M$  any  $G$ -module then the trivial homomorphism  $\partial: M \xrightarrow{0} G$  is a crossed module.
3. The homomorphism  $\partial: M \rightarrow \text{Aut}(M), m \rightarrow \iota_m(x) = mxm^{-1}$  from a group  $M$  to its automorphism group is a crossed module.
4. If  $\partial: M \twoheadrightarrow G$  is a central group extension (i.e. if  $\text{Ker } \partial$  lies in  $Z(M)$  and  $\partial$  is surjective) then  $\partial$  is a crossed module in which  ${}^g m = \tilde{g}m\tilde{g}^{-1}$  with  $\tilde{g} \in M$  satisfying  $\partial(\tilde{g}) = g$ .

**Definition 5.1.2.** For any crossed module  $\partial: M \rightarrow P$ , the *order* of  $\partial$  is defined to be the product  $|\partial| = |M| \times |P|$  of the orders of the groups  $M, P$ .

**Lemma 5.1.1.** [8] For any crossed module  $\partial: M \rightarrow P$ ,  $\text{Im } \partial$  is a normal subgroup of  $P$ .

**Definition 5.1.3.** [8] For any crossed module  $\partial: M \rightarrow P$ , the *homotopy groups* of  $\partial$  are defined as

$$\pi_n(\partial) = \begin{cases} P/\text{Im } \partial & n = 1, \\ \text{Ker } \partial & n = 2, \\ 0 & n > 2. \end{cases}$$

**Lemma 5.1.2.** [8] Let  $\partial: M \rightarrow P$  be a crossed module. Then

- (i)  $\pi_2(\partial) \leq Z(M)$ .
- (ii)  $\pi_2(\partial)$  is a  $\pi_1(\partial)$ -module.

Note that  $\pi_2(\partial)$  is a  $\pi_1(\partial)$ -module with the action  $\pi_1(\partial)$  on  $\pi_2(\partial)$  defined by  ${}^g a := \tilde{g}a$  where  $\tilde{g}$  is an element chosen from the pre image of  $g$  in  $P$ .

**Definition 5.1.4.** A *morphism* between two crossed modules  $\partial: M \rightarrow P$  and  $\partial': M' \rightarrow P'$  is a pair  $(\mu, \eta)$  of homomorphisms of groups  $\mu: M \rightarrow M'$  and  $\eta: P \rightarrow P'$  such that

- (i) the diagram

$$\begin{array}{ccc} M & \xrightarrow{\mu} & M' \\ \partial \downarrow & & \downarrow \partial' \\ P & \xrightarrow{\eta} & P' \end{array}$$

commutes, i.e.  $\partial'\mu = \eta\partial$ , and

- (ii)  $\mu(p m) = {}^{\eta(p)}\mu(m)$  for all  $m \in M, p \in P$ .

A morphism is an *isomorphism* if  $\mu$  and  $\eta$  are isomorphisms.

Crossed modules and their morphisms form a category. We denote this category by **XMod**.

## 5.2 Nerve of $\text{cat}^1$ -groups

The following notion was introduced by J-L.Loday in [33].

**Definition 5.2.1.** A *cat<sup>1</sup>-group* is a triple  $(G, s, t)$  such that  $G$  is a group and  $s, t: G \rightarrow G$  are group homomorphisms satisfying conditions

- (i)  $st = t$  and  $ts = s$ ,
- (ii)  $[\text{Ker } s, \text{Ker } t] = \mathbf{1}$ .

**Definition 5.2.2.** For any  $\text{cat}^1$ -group  $(G, s, t)$ , the *order* of  $(G, s, t)$  is defined to be the order of group  $G$ .

**Proposition 5.2.1.** [33] *Let  $(G, s, t)$  be a  $\text{cat}^1$ -group. Then*

- (i)  $\text{Im } s = \text{Im } t$ ,
- (ii)  $ss = s, tt = t$ .

**Definition 5.2.3.** A *morphism* of  $\text{cat}^1$ -groups between  $(G, s, t)$  and  $(G', s', t')$  is a homomorphism of groups  $\phi: G \rightarrow G'$  such that  $\phi s = s' \phi$  and  $\phi t = t' \phi$ .

A morphism is an *isomorphism* if  $\phi$  is an isomorphism. It is easy to see that  $\text{cat}^1$ -groups and their morphisms form a category. The category is denoted by **Cat1**.

**Proposition 5.2.2.** [8] *There exists a functor*

$$\lambda: \mathbf{XMod} \rightarrow \mathbf{Cat1}$$

*which sends a crossed module  $\partial: M \rightarrow P$  to the  $\text{cat}^1$ -group  $(M \rtimes P, s, t)$ , where  $s(m, p) = (1, p)$  and  $t(m, p) = (1, \partial(m)p)$ .*

From this proposition, we obtain an algorithm for computing the functor  $\lambda$ .

**Algorithm 5.2.1.**

**Input:** A crossed module or a morphism of crossed modules.

**Output:**

- A  $\text{cat}^1$ -group if input a crossed module.
- A morphism of  $\text{cat}^1$ -groups if input a morphism of crossed modules.

**Procedure:** We implement the formulae of Proposition 5.2.2.

**Proposition 5.2.3.** [8] *There exists a functor*

$$\gamma: \mathbf{Cat1} \rightarrow \mathbf{XMod}$$

*which sends a  $\text{cat}^1$ -group  $(G, s, t)$  to the crossed module  $t|_{\text{Ker } s}: \text{Ker } s \rightarrow \text{Im } s$  where  $t|_{\text{Ker } s}$  is restriction of  $t$  to  $\text{Ker } s$  and  $\text{Im } s$  acts on  $\text{Ker } s$  by conjugation.*

From this proposition, we obtain an algorithm for computing the functor  $\gamma$ .

**Algorithm 5.2.2.****Input:** A  $\text{cat}^1$ -group or a morphism of  $\text{cat}^1$ -groups.**Output:**

- A crossed module if input a  $\text{cat}^1$ -group.
- A morphism of crossed modules if input a morphism of  $\text{cat}^1$ -groups.

**Procedure:** We implement the formulae of Proposition 5.2.3.**Proposition 5.2.4.** [8] *The functors*

$$\lambda: \mathbf{XMod} \rightarrow \mathbf{Cat1} \text{ and } \gamma: \mathbf{Cat1} \rightarrow \mathbf{XMod}$$

*form an isomorphism of categories.*

For any  $\text{cat}^1$ -group  $(G, s, t)$ , we can consider  $(G, s, t)$  to be a category with objects the elements of  $\text{Im } s$  and morphisms the elements of  $G$ . The source (respectively target) of the morphism  $g$  is  $s(g)$  (respective  $t(g)$ ). The morphisms  $g$  and  $h$  are composable if  $t(g) = s(h)$  and their composite  $h \circ g$  is  $gt(g^{-1})h$ . So we construct a simplicial group by taking the nerve of this category  $(G, s, t)$ . The construction is described in the following Proposition.

**Proposition 5.2.5.** [33] *If  $(G, s, t)$  is a  $\text{cat}^1$ -group then the nerve of  $(G, s, t)$  is the simplicial group  $\mathcal{N}G$  given as follows:*

$$\bullet \mathcal{N}_n G = \begin{cases} \text{Im } s & \text{if } n = 0, \\ G & \text{if } n = 1, \\ \{(g_1, g_2, \dots, g_n) \mid t(g_i) = s(g_{i+1}), g_i \in G\} & \text{if } n > 1. \end{cases}$$

- the face maps  $d_i: \mathcal{N}_n G \rightarrow \mathcal{N}_{n-1} G$  with

$$d_i(g_1, g_2, \dots, g_n) = \begin{cases} (g_2, \dots, g_n) & \text{if } i = 0, \\ (g_1, \dots, g_i t(g_i^{-1}) g_{i+1}, \dots, g_n) & \text{if } 0 < i < n, \\ (g_1, g_2, \dots, g_{n-1}) & \text{if } i = n. \end{cases}$$

- the degeneracy maps  $\eta_i: \mathcal{N}_n G \rightarrow \mathcal{N}_{n+1} G$  with

$$\eta_i(g_1, g_2, \dots, g_n) = \begin{cases} (s(g_1), g_1, g_2, \dots, g_n) & \text{if } i = 0, \\ (g_1, \dots, g_i, t(g_i), g_{i+1}, \dots, g_n) & \text{if } 0 < i \leq n. \end{cases}$$

In addition, the Moore complex of this simplicial group has the form

$$\dots \rightarrow \mathbf{1} \rightarrow \mathbf{1} \rightarrow \dots \rightarrow \mathbf{1} \rightarrow \text{Ker } s \xrightarrow{t} \text{Im } s.$$

**Proposition 5.2.6.** *Let  $(G, s, t)$  be a  $\text{cat}^1$ -group and  $M = \text{Ker } t$ . Then, for  $n \geq 2$ ,*

$$\mathcal{N}_n G \cong M \rtimes_{\varphi_{n-1}} (M \rtimes_{\varphi_{n-2}} \dots (M \rtimes_{\varphi_1} G))$$

where

$$\varphi_1: G \rightarrow \text{Aut } M \text{ is given by } g \mapsto (m \mapsto s(g)ms(g^{-1}))$$

and for  $i \geq 2$

$$\varphi_i: M \rtimes_{\varphi_{i-1}} (M \rtimes_{\varphi_{i-2}} \dots (M \rtimes_{\varphi_1} G)) \rightarrow \text{Aut } M$$

is given by

$$(m_1, m_2, \dots, m_{i-1}, g) \mapsto (m \mapsto s(m_1 m_2 \dots m_{i-1} g) m s((m_1 m_2 \dots m_{i-1} g)^{-1})).$$

**Proof.** We consider the function

$$\psi: \mathcal{N}_n G \rightarrow M \rtimes_{\varphi_{n-1}} (M \rtimes_{\varphi_{n-2}} \dots (M \rtimes_{\varphi_1} G))$$

defined by

$$(g_1, g_2, \dots, g_{n-1}, g_n) \mapsto (g_1 s(g_2^{-1}), \dots, g_{n-1} s(g_n^{-1}), g_n).$$

Let  $x = (x_1, x_2, \dots, x_{n-1}, g)$  and  $y = (y_1, y_2, \dots, y_{n-1}, h)$  be in  $M \rtimes_{\varphi_{n-1}} (M \rtimes_{\varphi_{n-2}} \dots (M \rtimes_{\varphi_1} G))$ . We find the product  $xy$  as follows.

- For  $n = 2$ ,

$$xy = (x_1, g)(y_1, h) = (x_1 \varphi_1(g)(y_1), gh) = (x_1 s(g)y_1 s(g^{-1}), gh).$$

- For  $n = 3$ ,

$$\begin{aligned} xy &= (x_1, x_2, g)(y_1, y_2, h) \\ &= ((x_1)\varphi_2(x_2, g)(y_1), (x_2, g)(y_2, h)) \\ &= (x_1s(x_2g)y_1s((x_2g)^{-1}), x_2s(g)y_2s(g^{-1}), gh) \end{aligned}$$

By using induction, we obtain

$$xy = (S_1^{xy}, S_2^{xy}, \dots, S_{n-1}^{xy}, gh)$$

with

$$S_i^{xy} = x_i s(x_{i+1} \dots x_{n-1} g) y_i s((x_{i+1} \dots x_{n-1} g)^{-1}).$$

Let  $u = (g_1, g_2, \dots, g_n), v = (h_1, h_2, \dots, h_n) \in \mathcal{N}_n G$ , we have

$$\begin{aligned} \psi(u)\psi(v) &= \psi(g_1, g_2, \dots, g_n)\psi(h_1, h_2, \dots, h_n) \\ &= (g_1s(g_2^{-1}), \dots, g_{n-1}s(g_n^{-1}), g_n)(h_1s(h_2^{-1}), \dots, h_{n-1}s(h_n^{-1}), h_n) \\ &= (S_1, S_2, \dots, S_{n-1}, g_n h_n) \end{aligned}$$

with

$$\begin{aligned} S_i &= g_i s(g_{i+1}^{-1}) s(g_{i+1} s(g_{i+2}^{-1}) \dots g_{n-1} s(g_n^{-1}) g_n) h_i s(h_{i+1}^{-1}) s(g_{i+1} s(g_{i+2}^{-1}) \dots g_{n-1} s(g_n^{-1}) g_n)^{-1} \\ &= g_i h_i s(h_{i+1}^{-1}) s(g_{i+1}^{-1}) \text{ (as } s s = s) \\ &= (g_i h_i) s((g_{i+1} h_{i+1})^{-1}). \end{aligned}$$

So

$$\psi(u)\psi(v) = ((g_1 h_1) s((g_2 h_2)^{-1}), \dots, (g_{n-1} h_{n-1}) s((g_n h_n)^{-1}), g_n h_n).$$

On the other hand, we have

$$\begin{aligned} \psi(uv) &= \psi(g_1 h_1, g_2 h_2, \dots, g_n h_n) \\ &= ((g_1 h_1) s((g_2 h_2)^{-1}), \dots, (g_{n-1} h_{n-1}) s((g_n h_n)^{-1}), g_n h_n). \end{aligned}$$

This implies  $\psi$  is a homomorphism.

Furthermore, if  $\psi(g_1, g_2, \dots, g_n) = (1, 1, \dots, 1)$  then

$$(g_1 s(g_2^{-1}), \dots, g_{n-1} s(g_n^{-1}), g_n) = (1, 1, \dots, 1).$$

So  $g_i = 1$  for  $1 \leq i \leq n$ . It means  $\text{Ker } \psi = \mathbf{1}$  or  $\psi$  is injective.

Moreover, for  $x = (x_1, x_2, \dots, x_{n-1}, g) \in M \rtimes_{\varphi_{n-1}} (M \rtimes_{\varphi_{n-2}} \dots (M \rtimes_{\varphi_1} G))$ , it is easy to see

$$\psi(x_1 s(x_2, \dots, x_{n-1} g), \dots, x_i s(x_{i+1}, \dots, x_{n-1} g), \dots, g) = x.$$

So  $\psi$  is surjective. □

We can implement this Proposition on computer in the form of the following algorithm.

**Algorithm 5.2.3.**

**Input:** A finite cat<sup>1</sup>-group  $(G, s, t)$  and an integer  $n \geq 0$ .

**Output:** The nerve of  $(G, s, t)$  as the first  $n$  terms of a simplicial group  $\mathcal{N}G$  in the form:

- The groups  $\mathcal{N}_i G$  ( $0 \leq i \leq n$ ).
- The face homomorphisms  $d_j: \mathcal{N}_i G \rightarrow \mathcal{N}_{i-1} G$  ( $0 \leq j \leq i$ ).
- The degeneracy homomorphisms  $s_j: \mathcal{N}_i G \rightarrow \mathcal{N}_{i+1} G$  ( $0 \leq j \leq i$ ).

**Procedure:** We implement the formulae in Proposition 5.2.6 and in its proof.

## 5.3 Cat<sup>1</sup>-groups and crossed modules of low order

The construction of cat<sup>1</sup>-groups and crossed modules of low order have been studied by Alp and Wensley [3]. They have developed the XMod package [2] in the GAP computer systems and the XMod package provides data of all cat<sup>1</sup>-groups and crossed modules of order  $m \leq 70$ .

In this section, we give a new algorithm to compute all non-isomorphic  $\text{cat}^1$ -group structures on a finite group. By using this algorithm, we construct data of all  $\text{cat}^1$ -groups and crossed modules of order  $m \leq 255$ .

An important resource for finite group theorists is the computer classification of all groups of low order. This classification is available in the **GAP** computer systems [28] and, for example, can be used to:

- (i) list non-isomorphic groups of a given order  $m$ ;
- (ii) identify the isomorphism class of a user-defined group  $G$  in terms of a pair  $(m, k)$  where  $m$  is the order of  $G$  and  $k$  is a catalogue number.

**Example 5.3.1.**

```
gap> G:=SmallGroup(128,5);
<pc group of size 128 with 7 generators>
gap> H:=DihedralGroup(56);;
gap> IdGroup(H);
[ 56, 5 ]
```

For any finite group  $G$ , a  $\text{cat}^1$ -group with underlying group  $G$  is called a *cat<sup>1</sup>-group structure on group  $G$* . To compute all non-isomorphic  $\text{cat}^1$ -group structures on group  $G$  we perform the following two steps:

**Step 1.** Compute all possible  $\text{cat}^1$ -group structures on group  $G$ .

We begin by computing a list  $\mathbb{L}$  of all normal subgroups  $N$  in  $G$  and a list  $\mathbb{L}'$  of subgroups  $K$  in  $G$  representing all subgroup conjugacy classes. We then do:

1. For each  $N \in \mathbb{L}$ . Let  $p: G \rightarrow G/N$  be the quotient homomorphism. We find all  $K \in \mathbb{L}'$  satisfying
  - $K$  is isomorphic to  $G/N$  (use `IdGroup()`).
  - $|p(K)| = |G/N|$ .

For each such  $K$  the quotient homomorphism  $p$  restricts to an isomorphism  $p|_K: K \rightarrow G/N$ . We form the inverse isomorphism  $(p|_K)^{-1}: G/N \rightarrow K$  and set  $\sigma := (p|_K)^{-1}p: G \rightarrow G$ . By construction we have  $\text{Ker } \sigma = N$ ,  $\text{Im } \sigma = K$

and  $\sigma\sigma = \sigma$ . For each normal subgroup  $N$  we compute the list  $\mathbb{L}_N$  of such homomorphisms  $\sigma$ .

2. For each pair of normal subgroups  $N, M$  in  $G$  satisfying  $[N, M] = \mathbf{1}$  we consider all  $s \in \mathbb{L}_N, t \in \mathbb{L}_M$ . If  $\text{Im } s = \text{Im } t$  we add the data  $(G, s, t)$  to our list of cat<sup>1</sup>-group structures on  $G$ .

In this manner, all possible cat<sup>1</sup>-group structures on  $G$  are produced, though isomorphic copies may have been produced.

**Step 2.** Compute a list of non-isomorphic cat<sup>1</sup>-group structures on  $G$  from Step 1.

We use an algorithm to test whether two cat<sup>1</sup>-group structures on group  $G$  are isomorphic. To do this we need to access the automorphism group  $\text{Aut}(G)$  of the group  $G$ . As this automorphism group can be large we follow a suggestion of Alexander Hulpke and use:

- (i) the action  ${}^f K = f(K)$  of  $f \in \text{Aut}(G)$  on subgroups  $K \leq G$ ;
- (ii) the action  ${}^f s = fsf^{-1}$  of  $f \in \text{Aut}(G)$  on endomorphisms  $s: G \rightarrow G$ .

For each action we have adapted a GAP implementation of an orbit-stabilizer algorithm written by Alexander Hulpke and used it to

- (i) compute the orbit  $\text{Orb}(x)$  of an element  $x$  under the action;
- (ii) compute the stabilizer subgroup  $\text{Stab}(x)$ ;
- (iii) find  $f \in \text{Aut}(G)$  if  $x' \in \text{Orb}(x)$  such that  ${}^f x = x'$ .

A description of the orbit-stabilizer algorithm can be found in [31].

To test if two cat<sup>1</sup>-group structures  $(G, s, t)$  and  $(G, s', t')$  are isomorphic we perform the following steps.

1. We first use GAP's `IdGroup()` function to check that  $\text{Im } s \cong \text{Im } s'$  and  $\text{Ker } s \cong \text{Ker } s'$  and  $\text{Ker } t \cong \text{Ker } t'$ . If this check fails then the two cat<sup>1</sup>-groups are not isomorphic and we return *false*.

2. Otherwise we compute the orbit of  $\text{Ker } s$  under the action of  $\text{Aut}(G)$ . If  $\text{Ker } s'$  is not in this orbit then the two  $\text{cat}^1$ -groups are not isomorphic and we return *false*. Otherwise we can find an element  $f \in \text{Aut}(G)$  such that  $\text{Ker } s' = f(\text{Ker } s)$ . We then define  $s'' = f^{-1}s'$ ,  $t'' = f^{-1}t'$  to obtain a  $\text{cat}^1$ -group  $(G, s'', t'')$  which is isomorphic to  $(G, s', t')$  and which has the property that  $\text{Ker } s'' = \text{Ker } s$ . For ease of notation we redefine  $s' := s''$ ,  $t' := t''$ . In other words, we replace  $(G, s', t')$  by an isomorphic  $\text{cat}^1$ -group satisfying  $\text{Ker } s' = \text{Ker } s$ .
3. We compute the stabilizer subgroup  $\text{Stab}(\text{Ker } s) \leq \text{Aut}(G)$  and the orbit of  $\text{Im } s$  under the action of  $\text{Stab}(\text{Ker } s)$ . If  $\text{Im } s'$  is not in this orbit then the two  $\text{cat}^1$ -groups are not isomorphic and we return *false*. Otherwise we can find an element  $f \in \text{Stab}(\text{Ker } s)$  such that  $\text{Im } s' = f(\text{Im } s)$  and then replace  $(G, s', t')$  by an isomorphic  $\text{cat}^1$ -group satisfying  $\text{Im } s' = \text{Im } s$  and  $\text{Ker } s' = \text{Ker } s$ .
4. We compute the stabilizer subgroup  $\text{Stab}(\text{Im } s) \leq \text{Stab}(\text{Ker } s)$  and the orbit of  $\text{Ker } t$  under the action of  $\text{Stab}(\text{Im } s)$ . If  $\text{Ker } t'$  is not in this orbit then the two  $\text{cat}^1$ -groups are not isomorphic and we return *false*. Otherwise we replace  $(G, s', t')$  by an isomorphic  $\text{cat}^1$ -group satisfying  $\text{Ker } t' = \text{Ker } t$ ,  $\text{Im } s' = \text{Im } s$  and  $\text{Ker } s' = \text{Ker } s$ .
5. We compute the stabilizer  $\text{Stab}(\text{Ker } t) \leq \text{Stab}(\text{Im } s)$  and the orbit of  $s$  under the action of  $\text{Stab}(\text{Ker } t)$ . If  $s'$  is not in this orbit the two  $\text{cat}^1$ -groups are not isomorphic and we return *false*. Otherwise we replace  $(G, s', t')$  by an isomorphic  $\text{cat}^1$ -group satisfying  $\text{Ker } t' = \text{Ker } t$ ,  $s' = s$ .
6. We compute the stabilizer  $\text{Stab}(s) \leq \text{Stab}(\text{Ker } t)$  and the orbit of  $t$  under the action of  $\text{Stab}(s)$ . If  $t'$  is not in this orbit then the two  $\text{cat}^1$ -groups are not isomorphic and we return *false*. Otherwise we return *true*.

**Algorithm 5.3.1.**

**Input:** A finite group  $G$ .

**Output:** A list of all non-isomorphic  $\text{cat}^1$ -group structures on group  $G$ .

**Procedure:** We implement the above two steps.

By using the small group database of the GAP computer systems, we know that there are:

- 7012 non-isomorphic groups of order  $m \leq 255$ .
- 56092 non-isomorphic groups of order 256.

So, in this thesis, we only list all non-isomorphic groups of order  $m \leq 255$ . Then we implement the above algorithm and perform it to construct data of all  $\text{cat}^1$ -groups of order  $m \leq 255$ . The computation of all these  $\text{cat}^1$ -group took one month. Furthermore, the data of these  $\text{cat}^1$ -groups is stored in the HAP package [20].

**Example 5.3.2.** The following GAP session illustrates how to compute a list of all non-isomorphic  $\text{cat}^1$ -group structures on group  $G$  where  $G$  is equal to the 500th group of order 2000 from the database of small groups.

```
gap> G:=SmallGroup(2000,500);;
gap> L:=CatOneGroupsByGroup(G);;
gap> Length(L);
16
```

We adapt Step 2 of the above algorithm to give two algorithms relating  $\text{cat}^1$ -groups.

**Algorithm 5.3.2.**

**Input:** Two finite  $\text{cat}^1$ -groups  $(G, s, t)$  and  $(G', s', t')$ .

**Output:** An isomorphism of  $\text{cat}^1$ -groups if  $(G, s, t)$  and  $(G', s', t')$  are isomorphic and *fail* otherwise.

**Procedure:**

- If group  $G$  is not isomorphic to group  $G'$  then return *fail*. Otherwise we compute an isomorphism  $f: G \rightarrow G'$ . Then set  $s'' := fs'f^{-1}$  and  $t'' := ft'f^{-1}$ . Thus  $(G, s'', t'')$  is a  $\text{cat}^1$ -group structure on  $G$ .
- We use Step 2 of Algorithm 5.3.1, if  $(G, s, t)$  and  $(G, s'', t'')$  are not isomorphic then return *fail*. Otherwise, we find  $h \in \text{Aut}(G)$  such that  $s'' := h^{-1}sh, t'' := h^{-1}th$ . We then set  $f := fh$  then return  $f: (G, s, t) \rightarrow (G', s', t')$ .

**Algorithm 5.3.3.**

**Input:** A finite cat<sup>1</sup>-group  $(G, s, t)$  of order less than or equal to 255.

**Output:** A triple  $(m, k, i)$  where  $G$  is isomorphic to the  $k$ th group of order  $m$  in the database of small groups and  $(G, s, t)$  is isomorphic to the  $i$ th cat<sup>1</sup>-group structure on group  $G$ .

**Procedure:**

- Use `IdGroup()` to identify group  $G$  by positive integers  $m, k$ .
- We compute the list  $L$  of all non-isomorphic cat<sup>1</sup>-group structures on  $G$  (by using the database of small cat<sup>1</sup>-groups). If  $(G, s, t)$  is isomorphic to  $i$ th cat<sup>1</sup>-group of the list  $L$  then return  $(m, k, i)$ .

**Example 5.3.3.** The following GAP session illustrates how to compute an isomorphism between  $G_1$  and  $G_2$  where  $G_1$  is the cat<sup>1</sup>-group corresponding to the crossed module  $\partial: D_{12} \rightarrow \text{Aut}(D_{12})$  and  $G_2$  is the 8th cat<sup>1</sup>-group structure on the 154th group of the library of groups of order 144.

```
gap> XG1:=CrossedModuleByAutomorphismGroup(DihedralGroup(12));;
gap> G1:=CatOneGroupByCrossedModule(XG1);;
gap> G2:=SmallCatOneGroup(144,154,8);;
gap> f:=IsomorphismCatOneGroups(G1,G2);
Morphism of two cat-1-groups
```

**Example 5.3.4.** The following GAP session illustrates how to identify the cat<sup>1</sup>-group  $G_3$  where  $G_3$  is the cat<sup>1</sup>-group corresponding the crossed module  $\partial: C_{30} \rightarrow \text{Aut}(C_{30})$ .

```
gap> XG3:=CrossedModuleByAutomorphismGroup(CyclicGroup(30));;
gap> G3:=CatOneGroupByCrossedModule(XG3);;
gap> IdCatOneGroup(C3);
[ 240, 195, 8 ]
```

As we know there is an isomorphism between the category of cat<sup>1</sup>-groups and the category of crossed modules. By using the functors  $\lambda, \gamma$  in Section 5.2 and algorithms on cat<sup>1</sup>-groups, we obtain the following algorithms.

**Algorithm 5.3.4.****Input:** Two finite crossed modules  $\partial$  and  $\partial'$ .**Output:** A isomorphism of crossed modules if  $\partial$  and  $\partial'$  are isomorphic and *fail* otherwise.**Procedure:** We do the following steps:

- Use Algorithm 5.2.1 to compute cat<sup>1</sup>-group  $C^\partial$  and  $C^{\partial'}$  corresponding to  $\partial$  and  $\partial'$ .
- Use Algorithm 5.3.2 to compute an isomorphism  $f: C^\partial \rightarrow C^{\partial'}$  if it exists and return *fail* otherwise.
- Use Algorithm 5.2.2 to compute the isomorphism  $Xf: \partial \rightarrow C^{\partial'}$  corresponding to  $f: C^\partial \rightarrow C^{\partial'}$ .

**Algorithm 5.3.5.****Input:** A finite crossed module  $\partial: M \rightarrow P$  of order less than or equal to 255.**Output:** A pair  $(m, k)$  where  $\partial$  is isomorphic to the  $k$ th crossed module of order  $m$  in the database of crossed modules.**Procedure:**

- Use Algorithm 5.2.1 to compute cat<sup>1</sup>-group  $C^\partial$  corresponding to  $\partial$ .
- Identify  $C^\partial$  by using Algorithm 5.3.3 and find the catalogue number  $k$  of  $C^\partial$  in the list of cat<sup>1</sup>-groups of order  $m$ .

**Example 5.3.5.** The following GAP session illustrates how to identify the crossed module  $\partial: D_{12} \rightarrow \text{Aut}(D_{12})$ . And then compute an isomorphism between  $\partial$  and  $\partial'$  where  $\partial'$  is the 891th crossed module of order 144.

```

gap> XC:=CrossedModuleByAutomorphismGroup(DihedralGroup(12));;
gap> IdCrossedModule(XC);
[ 144, 891 ]
gap> XD:=SmallCrossedModule(144,891);;

```

```
gap> f:=IsomorphismCrossedModules(XC,XD);
Morphism of two crossed modules
```

## 5.4 Quasi-isomorphisms of crossed modules

**Definition 5.4.1.** A morphism  $(\mu, \eta)$  of crossed modules  $\partial: M \rightarrow P$  and  $\partial': M' \rightarrow P'$  is said to be a *quasi-isomorphism* if  $(\mu, \eta)$  induces isomorphisms  $\mu_*: \pi_2(\partial) \xrightarrow{\cong} \pi_2(\partial')$  and  $\eta_*: \pi_1(\partial) \xrightarrow{\cong} \pi_1(\partial')$ .

**Definition 5.4.2.** Two crossed modules  $\partial: M \rightarrow P$  and  $\partial': M' \rightarrow P'$  are said to be *quasi-isomorphic* if there is a sequence of morphisms of crossed modules

$$\begin{array}{ccccccc} M & \xrightarrow{\mu_1} & M_1 & \xleftarrow{\mu_2} & M_2 & \xrightarrow{\mu_3} & \dots & \xleftarrow{\mu_k} & M' \\ \partial \downarrow & & \partial_1 \downarrow & & \partial_2 \downarrow & & & & \partial' \downarrow \\ P & \xrightarrow{\eta_1} & P_1 & \xleftarrow{\eta_2} & P_2 & \xrightarrow{\eta_3} & \dots & \xleftarrow{\eta_k} & P' \end{array}$$

such that each  $(\mu_i, \eta_i)$  is a quasi-isomorphism for  $1 \leq i \leq k$ .

We write  $\partial \simeq \partial'$  to denote that  $\partial$  is quasi-isomorphic to  $\partial'$ . Note that  $\simeq$  is an equivalence relation on crossed modules; the corresponding equivalence classes are called *quasi-isomorphism classes*.

Since the category **Cat1** is isomorphic to the category **XMod**, we also give corresponding definitions such as “homotopy groups”, “quasi-isomorphism”, “quasi-isomorphic” in the category **Cat1**.

**Definition 5.4.3.** For any  $\text{cat}^1$ -group  $(G, s, t)$ , the *homotopy groups* of  $(G, s, t)$  are defined as

$$\pi_n(G, s, t) = \begin{cases} \text{Im } s/t(\text{Ker } s) & \text{if } n = 1, \\ \text{Ker } s \cap \text{Ker } t & \text{if } n = 2, \\ 0 & \text{if } n > 2. \end{cases}$$

**Definition 5.4.4.** A morphism  $\phi: (G, s, t) \rightarrow (G', s', t')$  of  $\text{cat}^1$ -groups is said to be a *quasi-isomorphism* if  $\phi$  induces isomorphisms  $\phi_1^*: \pi_1(G, s, t) \xrightarrow{\cong} \pi_1(G', s', t')$  and  $\phi_2^*: \pi_2(G, s, t) \xrightarrow{\cong} \pi_2(G', s', t')$ .

**Definition 5.4.5.** Two  $\text{cat}^1$ -groups  $(G, s, t)$  and  $(G', s', t')$  are said to be *quasi-*

*isomorphic* if there is a sequence of morphisms of  $\text{cat}^1$ -groups

$$(G, s, t) \xrightarrow{\phi_1} (G_1, s_1, t_1) \xleftarrow{\phi_2} (G_2, s_2, t_2) \xrightarrow{\phi_3} \cdots \xleftarrow{\phi_k} (G', s', t')$$

such that each  $\phi_i$  is a quasi-isomorphism for  $1 \leq i \leq k$ .

We now give an algorithm which inputs a finite  $\text{cat}^1$ -group  $(G, s, t)$  and outputs a quasi-isomorphic  $\text{cat}^1$ -group  $(G', s', t')$  where  $G'$  has order less than or equal to the order of  $G$ . In some case the order of  $G'$  will be significantly smaller than that of  $G$ .

To find the  $\text{cat}^1$ -group  $(G', s', t')$ , we first need to solve the following two problems:

**Problem 1.** Let  $H$  be a normal subgroup of  $G$ . If  $(G/H, s^*, t^*)$  is a  $\text{cat}^1$ -group where  $s^*, t^*$  are defined by  $s^*(gH) = s(g)H$ ,  $t^*(gH) = t(g)H$  for all  $g \in G$ , then how to check if the natural homomorphism  $p: G \rightarrow G/H$  is a quasi-isomorphism?

**Problem 2.** Let  $K$  be a subgroup of  $G$ . If  $(K, s_*, t_*)$  is a  $\text{cat}^1$ -group where  $s_*, t_*$  are the restriction of  $s, t$  to  $K$ , then how to check if the inclusion  $i: K \rightarrow G$  is a quasi-isomorphism?

Solution to **Problem 1.**

Note that  $\pi_1(G, s, t) = \text{Im } s/t(\text{Ker } s)$  and  $\pi_2(G, s, t) = \text{Ker } s \cap \text{Ker } t$ . To test if  $p$  is a quasi-isomorphism it suffices to check the following four sets of conditions:

1.  $s(H) \subset H$  and  $t(H) \subset H$ .
2.  $[s^{-1}(H), t^{-1}(H)] \subset H$ .
3.  $\frac{|\text{Im } s|}{|\text{Im } s \cap H|} = |\pi_1(G, s, t)| \frac{|t(s^{-1}(H))|}{|t(s^{-1}(H) \cap H)|}$ .
4.  $\left| \frac{s^{-1}(H)}{H} \cap \frac{t^{-1}(H)}{H} \right| = |\pi_2(G, s, t)|$  and  $\frac{|\text{Ker } s \cap \text{Ker } t|}{|(\text{Ker } s \cap \text{Ker } t) \cap H|} = |\pi_2(G, s, t)|$ .

Conditions 1 ensure that  $s^*$  and  $t^*$  are homomorphisms.

Condition 2 is equivalent to  $[\text{Ker } s^*, \text{Ker } t^*] = \mathbf{1}$  because

$$[\text{Ker } s^*, \text{Ker } t^*] = \left[ \frac{s^{-1}(H)}{H}, \frac{t^{-1}(H)}{H} \right] = \frac{[s^{-1}(H), t^{-1}(H)]H}{H}.$$

So

$$[\text{Ker } s^*, \text{Ker } t^*] = \mathbf{1} \Leftrightarrow \frac{[s^{-1}(H), t^{-1}(H)]H}{H} = \mathbf{1} \Leftrightarrow [s^{-1}(H), t^{-1}(H)] \subset H.$$

Condition 3 ensures that  $p$  induces an isomorphism  $p_1: \pi_1(G, s, t) \rightarrow \pi_1(G/H, s^*, t^*)$  because

$\pi_1(G/H, s^*, t^*) = \text{Im } s^*/t^*(\text{Ker } s^*)$  and  $p_1: \pi_1(G, s, t) \rightarrow \pi_1(G/H, s^*, t^*)$  given by

$$g t(\text{Ker } s) \mapsto (gH)(t^*(\text{Ker } s^*)).$$

Clearly,  $p_1$  is a surjective. So  $p_1$  is an isomorphism if  $|\text{Im } s^*/t^*(\text{Ker } s^*)| = |\pi_1(G, s, t)|$  or  $|\text{Im } s^*| = |\pi_1(G, s, t)| |t^*(\text{Ker } s^*)|$ . Moreover,

- $\text{Im } s^* = \frac{\text{Im } sH}{H} \cong \frac{\text{Im } s}{\text{Im } s \cap H}$ .
- $t^*(\text{Ker } s^*) = t^*(s^{-1}(H)/H) = \frac{t(s^{-1}(H))H}{H} \cong \frac{t(s^{-1}(H))}{t(s^{-1}(H)) \cap H}$ .

So

$$\frac{|\text{Im } s^*|}{|\text{Im } s \cap H|} = |\pi_1(G, s, t)| \frac{|t(s^{-1}(H))|}{|t(s^{-1}(H)) \cap H|}.$$

Conditions 4 ensure that  $p$  induces an isomorphism  $p_2: \pi_2(G, s, t) \rightarrow \pi_2(G/H, s^*, t^*)$  because

$$\pi_2(G/H, s^*, t^*) = \text{Ker } s^* \cap \text{Ker } t^* = \frac{s^{-1}(H)}{H} \cap \frac{t^{-1}(H)}{H} \text{ and}$$

$$p_2: \pi_2(G, s, t) \rightarrow \pi_2(G/H, s^*, t^*) \text{ given by } g \mapsto gH.$$

So,  $p_2$  is an isomorphism if it satisfies  $|\pi_2(G/H, s^*, t^*)| = |\pi_2(G, s, t)|$  and  $|\text{Im } p_2| = |\pi_2(G, s, t)|$ . Moreover,  $\text{Im } p_2 = \frac{(\text{Ker } s \cap \text{Ker } t)H}{H} \cong \frac{\text{Ker } s \cap \text{Ker } t}{(\text{Ker } s \cap \text{Ker } t) \cap H}$ . So

$$\left| \frac{s^{-1}(H)}{H} \cap \frac{t^{-1}(H)}{H} \right| = |\pi_2(G, s, t)| \text{ and } \frac{|\text{Ker } s \cap \text{Ker } t|}{|(\text{Ker } s \cap \text{Ker } t) \cap H|} = |\pi_2(G, s, t)|.$$

Solution to **Problem 2**.

Recall that  $\pi_1(G) = \text{Im } s/t(\text{Ker } s)$  and  $\pi_2(G) = \text{Ker } s \cap \text{Ker } t$ . To test if  $i$  is a quasi-isomorphism it suffices to check the following three sets of conditions:

1.  $s(K) \subset K$  and  $t(K) \subset K$ .
2.  $\frac{|s(K)|}{|t(\text{Ker } s \cap K)|} = |\pi_1(G, s, t)|$  and  $\frac{|s(K)|}{|s(K) \cap t(\text{Ker } s)|} = |\pi_1(G, s, t)|$ .
3.  $\text{Ker } s \cap \text{Ker } t \subset K$ .

Conditions 1 ensure that  $s_*$  and  $t_*$  are homomorphisms.

Conditions 2 ensure that  $i$  induces an isomorphism  $i_1: \pi_1(K, s_*, t_*) \rightarrow \pi_1(G, s, t)$  because

$$\pi_1(K, s_*, t_*) = \frac{\text{Im } s_*}{t_*(\text{Ker } s_*)} = \frac{s(K)}{t(\text{Ker } s \cap K)} \text{ and } i_1: \pi_1(K, s_*, t_*) \rightarrow \pi_1(G, s, t) \text{ given by}$$

$$ht(\text{Ker } s \cap K) \mapsto ht(\text{Ker } s).$$

We have  $\text{Im } i_1 = \frac{s(K)t(\text{Ker } s)}{t(\text{Ker } s)} \cong \frac{s(K)}{s(K) \cap t(\text{Ker } s)}$ . The homomorphism  $i_1$  is an isomorphism if  $|\text{Im } i_1| = |\pi_1(G, s, t)|$  and  $|\pi_1(K, s_*, t_*)| = |\pi_1(G, s, t)|$ . So  $\frac{|s(K)|}{|t(\text{Ker } s \cap K)|} = |\pi_1(G, s, t)|$  and  $\frac{|s(K)|}{|s(K) \cap t(\text{Ker } s)|} = |\pi_1(G, s, t)|$ .

Condition 3 ensures that  $i$  induces an isomorphism  $i_2: \pi_2(K, s_*, t_*) \rightarrow \pi_2(G, s, t)$  because

$$\pi_2(K, s_*, t_*) = \text{Ker } s_* \cap \text{Ker } t_* = (\text{Ker } s \cap K) \cap (\text{Ker } t \cap K) = (\text{Ker } s \cap \text{Ker } t) \cap K \text{ and}$$

$$i_2: \pi_2(K, s_*, t_*) \rightarrow \pi_2(G, s, t) \text{ given by } h \mapsto h.$$

It is easy to see that  $i_2$  is injective. So  $i_2$  is an isomorphism if  $\text{Im } i_2 = \pi_2(G, s, t)$ .

Thus  $(\text{Ker } s \cap \text{Ker } t) \cap K = \text{Ker } s \cap \text{Ker } t$  or  $\text{Ker } s \cap \text{Ker } t \subset K$ .

We implement the above two solutions as the following tests.

**Test 1.**

**Input:** A finite  $\text{cat}^1$ -group  $(G, s, t)$  and a normal subgroup  $H \triangleleft G$ .

**Output:** *True* if the natural morphism  $p: G \rightarrow G/H$  is a quasi-isomorphism and *false* otherwise.

**Test 2.**

**Input:** A finite  $\text{cat}^1$ -group  $(G, s, t)$  and a subgroup  $K \leq G$ .

**Output:** *True* if the inclusion  $i : K \hookrightarrow G$  is a quasi-isomorphism and *false* otherwise.

By using the above two Tests, we give an algorithm to compute a quasi-isomorphic  $\text{cat}^1$ -group of a finite  $\text{cat}^1$ -group.

**Algorithm 5.4.1.**

**Input:** A finite  $\text{cat}^1$ -group  $(G, s, t)$ .

**Output:** A quasi-isomorphic  $\text{cat}^1$ -group  $(G', s', t')$  and  $|G'| \leq |G|$ .

**Procedure:**

Step 1. Search through the normal subgroups of  $G$  and use Test 1 to find a biggest normal subgroup  $H$  of  $G$  such that the natural morphism  $p: G \rightarrow G/H$  is a quasi-isomorphism. We set  $G := G/H$ .

Search through the subgroups of  $G$  and use Test 2 to find a smallest subgroup  $K \leq G$  such that the inclusion  $i: K \hookrightarrow G$  is a quasi-isomorphism.

Step 2. While the order of  $K$  is less than the order  $G$ , we set  $G := K$  and repeat Step 1.

**Example 5.4.1.** The following GAP session illustrates how to find a quasi-isomorphic  $\text{cat}^1$ -group of the  $\text{cat}^1$ -group corresponding to the crossed module  $\partial: D_{24} \rightarrow \text{Aut}(D_{24})$ .

```
gap> XC:=CrossedModuleByAutomorphismGroup(DihedralGroup(24));;
gap> C:=CatOneGroupByCrossedModule(XC);;
gap> Order(C);
1152
gap> CQ:=QuotientQuasiIsomorph(C);;
gap> Order(CQ);
128
gap> CS:=SubQuasiIsomorph(CQ);;
gap> Order(CS);
8
```

```

gap> D:=QuasiIsomorph(C);
gap> Order(D);
8

```

These commands took 6 seconds.

Now we give an algorithm which inputs a finite crossed module  $\partial: M \rightarrow P$  and outputs a quasi-isomorphic crossed module  $\partial': M' \rightarrow P'$  where  $\partial'$  has order less than or equal to the order of  $\partial$ .

**Algorithm 5.4.2.**

**Input:** A finite crossed module  $\partial: M \rightarrow P$ .

**Output:** A quasi-isomorphic crossed module  $\partial': M' \rightarrow P'$  and  $|\partial'| \leq |\partial|$ .

**Procedure:**

- Applying the functor  $\gamma$  to  $\partial$  (see Algorithm 5.2.1), we obtain the  $\text{cat}^1$ -group  $C^\partial$  corresponding to  $\partial$ .
- Use Algorithm 5.4.1 to find a quasi-isomorphic  $\text{cat}^1$ -group of  $C^\partial$ . We call this  $\text{cat}^1$ -group  $D^\partial$ .
- Applying the functor  $\lambda$  to  $D^\partial$  (see Algorithm 5.2.2), we obtain the crossed module  $\partial'$  correspondent to  $D^\partial$ .

**Example 5.4.2.** The following GAP session illustrates how to find a quasi-isomorphic crossed module of the crossed module  $\partial: D_{32} \rightarrow \text{Aut}(D_{32})$ .

```

gap> XC:=CrossedModuleByAutomorphismGroup(DihedralGroup(32));;
gap> Order(XC);
4096
gap> XD:=QuasiIsomorph(XC);;
gap> Order(XD);
64

```

These commands took 54 seconds.

## 5.5 Homology of crossed modules

Theoretical aspects of homology of crossed modules have been studied in several papers [13, 10, 36]. Ana Romero [38, 39, 40] has written Lisp code for computing the integral homology of a crossed module  $X$  with a trivial action of  $\pi_2(X)$  on  $\pi_1(X)$ . Her method uses the fibration sequence

$$K(\pi_2(X), 2) \rightarrow X \rightarrow K(\pi_1(X), 1)$$

associated to any 2-type  $X$  and the classical homological perturbation lemma [6] to obtain a small algebraic model for  $X$  in terms of small models for the Eilenberg-Mac Lane spaces  $K(\pi_2(X), 2)$  and  $K(\pi_1(X), 1)$ . Her code is available as a module for the Kenzo system [27] for computations in Algebraic Topology.

Let  $\partial: M \rightarrow P$  be a crossed module. By applying the functor  $\lambda$  of Proposition 5.2.2 we obtain a  $\text{cat}^1$ -group. Then take the nerve of this  $\text{cat}^1$ -group we obtain a simplicial group. We define the integral homology of  $\partial: M \rightarrow P$  by using the integral homology of this simplicial group.

**Definition 5.5.1.** [26] For any crossed module  $\partial: M \rightarrow P$ , the *integral homology* of  $\partial: M \rightarrow P$  is defined by:

$$H_n(\partial: M \rightarrow P, \mathbb{Z}) := H_n(\mathcal{N}\lambda(\partial: M \rightarrow P), \mathbb{Z}) \text{ for all } n \geq 0.$$

**Lemma 5.5.1.** *The  $n$ th homology is a covariant functor from the category of crossed modules to the category of abelian groups.*

**Proof.** It is obvious.

**Theorem 5.5.2.** *If two crossed modules  $\partial: M \rightarrow P$  and  $\partial': M' \rightarrow P'$  are quasi-isomorphic then*

$$H_n(\partial: M \rightarrow P, \mathbb{Z}) \cong H_n(\partial': M' \rightarrow P', \mathbb{Z}) \text{ for all } n \geq 0.$$

**Proof.** Suppose that  $\partial: M \rightarrow P$  and  $\partial': M' \rightarrow P'$  are quasi-isomorphic then there is a sequence of morphisms of crossed modules

$$\begin{array}{ccccccc}
M & \xrightarrow{\mu_1} & M_1 & \xleftarrow{\mu_2} & M_2 & \xrightarrow{\mu_3} & \cdots \xleftarrow{\mu_k} & M' \\
\partial \downarrow & & \partial_1 \downarrow & & \partial_2 \downarrow & & & \partial' \downarrow \\
P & \xrightarrow{\eta_1} & P_1 & \xleftarrow{\eta_2} & P_2 & \xrightarrow{\eta_3} & \cdots \xleftarrow{\eta_k} & P'
\end{array} \tag{1}$$

such that each  $(\mu_i, \eta_i)$  is a quasi-isomorphism for  $1 \leq i \leq k$ .

Applying the functor  $\lambda$  in Proposition 5.2.2, we obtain a sequence of morphisms of  $\text{cat}^1$ -groups. We take the nerve of this sequence, we obtain a sequence of simplicial groups

$$G_* \xrightarrow{\phi_1} G_*^1 \xleftarrow{\phi_2} G_*^2 \xrightarrow{\phi_3} \cdots \xleftarrow{\phi_k} G_*'. \tag{2}$$

From Proposition 5.2.5, we see that (1) is the sequence of Moore complex of (2). This implies that the two simplicial groups  $G_*$  and  $G_*'$  are weakly equivalent. Applying Theorem 3.2.2, we have

$$H_n(G_*, \mathbb{Z}) \cong H_n(G_*', \mathbb{Z}) \text{ for all } n \geq 0.$$

or

$$H_n(\partial: M \rightarrow P, \mathbb{Z}) = H_n(\partial': M' \rightarrow P', \mathbb{Z}) \text{ for all } n \geq 0. \quad \square$$

Now we give an algorithm for computing the integral homology of a cross module.

**Algorithm 5.5.1.**

**Input:** A finite crossed module  $\partial$  and an integer  $n \geq 0$ .

**Output:** The integral homology  $H_n(\partial, \mathbb{Z})$ .

**Procedure:** We do the following steps:

- Applying the functor  $\lambda$  to  $\partial$  (see Algorithm 5.2.1), we obtain the  $\text{cat}^1$ -group  $C^\partial$  corresponding to  $\partial$ .
- Use Algorithm 5.2.3 to compute the nerve of the  $\text{cat}^1$ -group  $C^\partial$ . We call this simplicial group  $\mathcal{N}C^\partial$ .
- Use Algorithm 3.2.1 to compute the first  $n + 1$  terms of chain complex of the simplicial group  $\mathcal{N}C^\partial$  then return the homology of this chain complex at degree  $n$ .

The method which is presented in this algorithm relies on the theoretical representation of a crossed module as the diagonal of the bisimplicial set arising from the nerve of the corresponding  $\text{cat}^1$ -group of the crossed module. The Romero's method relies on the explicit representation of a crossed module by a certain twisting cocycle which is then used to construct a twisted cartesian product. At present the process of representing a crossed module by an explicit twisting cocycle has not been automated and hence a direct comparison of the computational performance of the two approaches and implementations is, at present, not practical. The paper [40] gives one example of the computation of the degree 5 integral homology of a crossed module  $X$  with  $\pi_1(X) = C_3$ ,  $\pi_2(X) = \mathbb{Z}_3$  and trivial action of  $\pi_1(X)$  on  $\pi_2(X)$ . The group-theoretic structure of this crossed module is not given in [40] but an analysis shows that it is a 2-type of order 81. The computation of the degree 5 integral homology of all 2-types of order 81 is certainly within the scope of the method presented in this thesis (see Section 5.6).

**Example 5.5.1.** The following GAP session illustrates how to compute

$$H_4(\partial: D_{32} \rightarrow \text{Aut}(D_{32}), \mathbb{Z}) = \mathbb{Z}_2 \oplus \mathbb{Z}_2 \oplus \mathbb{Z}_2.$$

The crossed module  $\partial$  has order 4096, homotopy groups  $\pi_1(\partial) = C_4 \times C_2$ ,  $\pi_2(\partial) = \mathbb{Z}_2$ , and  $\pi_1(\partial)$  acts non-trivially on  $\pi_2(\partial)$ . To reduce computations a quasi-isomorphism of crossed module  $\partial \simeq \partial'$  is constructed.

```
gap> XC:=CrossedModuleByAutomorphismGroup(DihedralGroup(32));;
gap> Order(XC);
4096
gap> StructureDescription(HomotopyGroup(XC,1));
"C4 x C2"
gap> StructureDescription(HomotopyGroup(XC,2));
"C2"
gap> XD:=QuasiIsomorph(XC);;
gap> Order(XD);
64
gap> Homology(XD,4);
[ 2, 2, 2 ]
```

These commands took 75 seconds.

## 5.6 2-types of low order

Recall that a 2-type  $X$  is a CW-space with homotopy groups  $\pi_n X = 0$  for  $n \neq 1, 2$ . Mac Lane and Whitehead [34] showed that there is a one-one correspondence between 2-types and quasi-isomorphism classes of crossed modules. When the homotopy groups  $\pi_1 X$  and  $\pi_2 X$  are finite one can represent the homotopy type  $X$  by a crossed module  $\partial: M \rightarrow P$  in which  $M$  and  $P$  are finite groups. We define the *order* of a quasi-isomorphism class of crossed modules to be the least order of any crossed module in the class. We then define the *order* of a 2-type  $X$  to be the order of the corresponding quasi-isomorphism class of crossed modules. In this section we describe a method to find smallest representatives of quasi-isomorphism classes of order  $m \leq 255$ . Moreover, this method is used to enumerate most of the 2-types of order  $m \leq 255$ .

**Definition 5.6.1.** Let  $\partial: M \rightarrow P$  be a crossed module. Then the  $\pi_1(\partial)$ -module  $\pi_2(\partial)$  induces the crossed module  $\pi_2(\partial) \xrightarrow{0} \pi_1(\partial)$  and this crossed module is defined to be *homotopy crossed module* of  $\partial$ . We denote it by  $H(\partial)$ .

**Lemma 5.6.1.** Let  $\partial: M \rightarrow P$  and  $\partial': M' \rightarrow P'$  be crossed modules. If  $\partial$  is quasi-isomorphic to  $\partial'$  then the homotopy crossed module  $H(\partial)$  of  $\partial$  is isomorphic to the homotopy crossed module  $H(\partial')$  of  $\partial'$ .

**Proof.** We only need to check if  $(\mu, \eta)$  is a quasi-isomorphism between  $\partial: M \rightarrow P$  and  $\partial': M' \rightarrow P'$  then  $(\mu, \eta)$  induces an isomorphism between two crossed modules  $\pi_2(\partial) \xrightarrow{0} \pi_1(\partial)$  and  $\pi_2(\partial') \xrightarrow{0} \pi_1(\partial')$ .

We consider the following digram

$$\begin{array}{ccccccc} \pi_2(\partial) & \xrightarrow{i} & M & \xrightarrow{\partial} & P & \xrightarrow{p} & \pi_1(\partial) \\ \downarrow \mu_* & & \downarrow \mu & & \downarrow \eta & & \downarrow \eta_* \\ \pi_2(\partial') & \xrightarrow{i'} & M' & \xrightarrow{\partial'} & P' & \xrightarrow{p'} & \pi_1(\partial') \end{array}$$

For  $g \in \pi_1(\partial), a \in \pi_2(\partial)$ , we have

$$\begin{aligned}\mu_*(^g a) &= \mu_*(\tilde{g}a) \text{ (where } \tilde{g} \in p^{-1}(g)\text{)} \\ &= \mu(\tilde{g}a) = \eta(\tilde{g})\mu(a) \\ &= \eta(\tilde{g})\mu_*(a) = p'\eta(\tilde{g})\mu_*(a) \\ &= \eta_*p(\tilde{g})\mu_*(a) = \eta_*(g)\mu_*(a).\end{aligned}$$

Furthermore,  $\mu_*, \eta_*$  are isomorphisms. Thus  $(\mu_*, \eta_*)$  is an isomorphism. It means  $H(\partial)$  is isomorphic to  $H(\partial')$ .  $\square$

**Lemma 5.6.2.** [8] *A quasi-isomorphism class  $X$  of crossed modules can be represented by the fundamental group  $\pi_1 X$ , the  $\pi_1 X$ -module  $\pi_2 X$  and a cohomology class  $\kappa \in H^3(\pi_1 X, \pi_2 X)$ .*

Let us denote

$$\begin{aligned}Iso2(m) &= \text{number of isomorphism classes of crossed modules of order } m. \\ QIso2(m) &= \text{number of homotopy 2-types of order } m \\ &= \text{number of quasi-isomorphism classes of order } m.\end{aligned}$$

**Proposition 5.6.3.** *Let  $p, q$  be primes and  $p < q$ . Then*

- (i)  $Iso2(p) = QIso2(p) = 2$ .
- (ii)  $Iso2(p^2) = 6$  and  $QIso2(p^2) = 5$ .
- (iii)  $Iso2(pq) = QIso2(pq) = 6$  when  $p$  divides  $q-1$  and  $Iso2(pq) = QIso2(pq) = 4$  when  $p$  does not divide  $q-1$

**Proof.**

(i) There are only two crossed modules of order  $p$ . They are  $C_p \rightarrow 0$  and  $0 \rightarrow C_p$ . So  $Iso2(p) = QIso2(p) = 2$ .

(ii) There are only six crossed modules of order  $p^2$ . They are  $C_p \xrightarrow{0} C_p, C_p \xrightarrow{1} C_p, C_{p^2} \rightarrow 0, 0 \rightarrow C_{p^2}, C_p \times C_p \rightarrow 0$  and  $0 \rightarrow C_p \times C_p$ . Clearly,  $C_p \xrightarrow{1} C_p$  is quasi-isomorphic to  $0 \rightarrow 0$ . Thus  $Iso2(p^2) = 6$  and  $QIso2(p^2) = 5$ .

(iii) We know that the cyclic group of order  $p$  can act non-trivially on the cyclic group of order  $q$  when  $p$  divides  $q-1$ . Moreover, the only groups of order  $pq$

with  $p$  not dividing  $q - 1$  are the cyclic groups; the only groups of order  $pq$  with  $p$  dividing  $q - 1$  are the cyclic group and one non-abelian semi-direct product of cyclic groups. Therefore, in the case  $p$  is not a divisor of  $q - 1$  there are four crossed modules  $C_p \xrightarrow{0} C_q$ ,  $C_q \xrightarrow{0} C_p$ ,  $C_{pq} \rightarrow 0$  and  $0 \rightarrow C_{pq}$ . In the case  $p$  is a divisor of  $q - 1$ , there are two more crossed modules  $C_p \xrightarrow{0} C_q$  with the non-trivial action of  $C_q$  on  $C_p$  and  $0 \rightarrow C_p \times C_q$ . It is easy to see that these crossed modules are not quasi-isomorphic.  $\square$

We now describe a method to find smallest representatives of quasi-isomorphism classes of order  $m \leq 255$ . To do this we perform the following steps:

**Step 1.** From Section 5.3, a table of all isomorphism types of crossed modules of order  $m \leq 255$  has been computed. This table immediately yields the upper bound  $Iso2(m) \geq QIso2(m)$ . We denote the table by  $\mathbb{T}$ .

**Step 2.** We apply Algorithm 5.4.2 to each crossed module  $\partial$  in  $\mathbb{T}$ , and then discarding  $\partial$  if the algorithm succeeds in finding a smaller crossed module quasi-isomorphic to  $\partial$ .

**Step 3.** We partition the table  $\mathbb{T}$  into classes with two crossed modules in the same class if and only if their homotopy crossed module are isomorphic. The class of  $\partial$  is denoted  $\mathbb{H}_\partial$ .

**Step 4.** For each class  $\mathbb{H}_\partial$ , if the class only contains one crossed module  $\partial$  of order  $m$ , we add  $\partial$  into the list of smallest representatives of quasi-isomorphism classes of order  $|\partial|$ . If the class contains more than one crossed module, we use Algorithm 5.5.1 to compute the abelian invariants of the integral homology group  $H_n(\partial, \mathbb{Z})$  for  $n \leq 4$ . Then use the cohomology function in the HAP package [20] to compute  $H^3(\pi_1(\partial), \pi_2(\partial))$ . The order of this cohomology group provides an upper bound on the number of quasi-isomorphism classes of crossed modules with given fundamental group  $\pi_1(\partial)$  and given second homotopy group  $\pi_2(\partial)$ . In some cases this upper bound is sufficient to conclude that two crossed module in the class  $\mathbb{H}_\partial$  are quasi-isomorphic. If  $\partial$  is quasi-isomorphic to  $\partial'$  and  $|\partial| \leq |\partial'|$  then we delete  $\partial'$  from  $\mathbb{H}_\partial$ . Therefore, we obtain a list of smallest representatives of quasi-isomorphism classes in the class  $\mathbb{H}_\partial$ . For each representative  $\partial$ , we add  $\partial$  into the list of smallest representatives of quasi-isomorphism classes of order  $|\partial|$ . For example

```

gap> X1:=SmallCrossedModule(4,4);;
gap> X2:=SmallCrossedModule(16,6);;
gap> X3:=SmallCrossedModule(16,11);;
gap> X4:=SmallCrossedModule(16,14);;
gap> IdCrossedModule(HomotopyCrossedModule(X1));
[ 4, 4 ]
gap> IdCrossedModule(HomotopyCrossedModule(X2));
[ 4, 4 ]
gap> IdCrossedModule(HomotopyCrossedModule(X3));
[ 4, 4 ]
gap> IdCrossedModule(HomotopyCrossedModule(X4));
[ 4, 4 ]

```

From the above GAP session, the homotopy crossed modules of  $X_1, X_2, X_3$  and  $X_4$  are isomorphic. This implies  $X_1, X_2, X_3$  and  $X_4$  are in same class  $\mathbb{H}_\partial$ . In addition,  $H^3(\pi_1(\partial), \pi_2(\partial)) = \mathbb{Z}_2$  and

```

gap> Homology(X1,2);
[ 2 ]
gap> Homology(X2,2);
[ 2 ]
gap> Homology(X3,2);
[ ]
gap> Homology(X4,2);
[ ]

```

Thus,  $X_1$  is quasi-isomorphic to  $X_2$  and  $X_3$  is quasi-isomorphic to  $X_4$ . So we only add  $X_1$  into the list of smallest representatives of quasi-isomorphism classes of order 4 and add  $X_3$  into the list of smallest representatives of quasi-isomorphism classes of order 16.

By using the above method the list of smallest representatives of all quasi-isomorphism classes of order  $m$  are computed and recorded in the HAP package [20] for most  $m \leq 255$ .

We also use this record to implement the function `SmallQuasiCrossedModule(m,k)`

which inputs a pair  $(m, k)$  and returns the smallest representative of  $k$ th quasi-isomorphism classes of order  $m \leq 255$ . Furthermore, for each crossed module  $\partial$  of order less than or equal to 255, we can find the order  $m$  of quasi-isomorphism class of  $\partial$  and the catalogue number  $k$  of this class. Then this data is also stored in the HAP package [20].

Now we give an algorithm for identifying the quasi-isomorphism class of a crossed module.

**Algorithm 5.6.1.**

**Input:** A finite crossed module  $\partial : M \rightarrow P$ .

**Output:** If successful in finding the smallest representative of the quasi-isomorphism class of  $\partial$ , it outputs a pair of integers  $(m, k)$  with  $m$  the order of this class and  $k$  the number uniquely identifying this class, and *fail* otherwise.

**Procedure:**

- Use Algorithm 5.4.2 to find a quasi-isomorphic crossed module  $\partial'$ .
- If the order  $\partial'$  is greater than 255 then return *fail*. Otherwise, we use the above data to find a pair  $(m, k)$ . Then return  $(m, k)$  if it exists and return *fail* otherwise.

**Example 5.6.1.** The following GAP session illustrates how to find smallest representatives of quasi-isomorphism classes of the crossed module  $\partial : D_{30} \rightarrow \text{Aut}(D_{30})$ .

```
gap> XC:=CrossedModuleByAutomorphismGroup(DihedralGroup(30));;
gap> IdQuasiCrossedModule(XC);
[ 4, 1 ]
gap> XD:=SmallQuasiCrossedModule(4,1);
Crossed module with group homomorphism 1 -> C4
```

On the other hand, we use the above method and obtain the value table of  $Iso2(m)$  and  $QIso2(m)$  for  $m \leq 255$ . From Proposition 5.6.3 we omit values for  $m = p, p^2, pq$  from the table.

$m$	1	8	12	16	18	20	24	27	28	30	32	36	40	42	44	45
$Iso2(m)$	1	18	20	62	22	20	73	18	18	20	251	78	72	26	18	12
$QIso2(m)$	1	14	18	43	19	18	61	14	16	20	?	63	60	26	16	10
$m$	48	50	52	54	56	60	63	64	66	68	70	72	75	76	78	
$Iso2(m)$	296	22	20	81	68	77	18	1276	20	20	20	325	14	18	26	
$QIso2(m)$	224	19	18	65	56	73	16	?	20	18	20	251	12	16	26	
$m$	80	81	84	88	90	92	96	98	99	100	102	104	105	108		
$Iso2(m)$	302	64	90	66	76	18	1446	22	12	87	20	72	12	308		
$QIso2(m)$	230	?	84	54	66	16	?	19	10	71	20	60	12	238		
$m$	110	112	114	116	117	120	124	125	126	128	130	132	135			
$Iso2(m)$	26	270	26	20	18	342	18	18	102	9120	20	68	36			
$QIso2(m)$	26	202	26	18	16	302	16	14	92	?	20	64	28			
$m$	136	138	140	144	147	148	150	152	153	154	156	160	162			
$Iso2(m)$	74	20	72	1469	25	20	84	66	12	20	100	1507	335			
$QIso2(m)$	62	20	68	?	22	18	74	54	10	20	94	?	?			
$m$	164	165	168	170	171	172	174	175	176	180	182	184	186			
$Iso2(m)$	20	12	397	20	20	18	20	12	266	328	20	66	26			
$QIso2(m)$	18	12	345	20	18	16	20	10	198	280	20	54	26			
$m$	188	189	190	192	195	196	198	200	204	207	208	210	212			
$Iso2(m)$	18	70	20	9219	12	72	76	349	76	12	300	92	20			
$QIso2(m)$	16	58	20	?	12	57	66	271	72	10	228	92	18			
$m$	216	220	222	224	225	228	230	231	232	234	236	238	240			
$Iso2(m)$	1430	90	26	1334	42	90	20	12	72	102	18	20	1705			
$QIso2(m)$	1064	84	26	?	31	84	20	12	60	92	16	20	1391			
$m$	242	243	244	245	246	248	250	252	255							
$Iso2(m)$	22	280	20	12	20	66	81	388	8							
$QIso2(m)$	19	?	18	10	20	54	65	338	8							

In addition, we also give a partial result for  $QIso2(m)$  of order that is missing on the above table.

---

$m$	32	64	81	96	128	144	160	162	192	224	243
<i>MIN</i>	158	726	45	996	4811	1057	1045	249	5854	904	183
<i>MAX</i>	171	831	46	1052	6105	1061	1101	251	6557	960	201

## Chapter 6

### Homology of maps of $n$ -types

## 6.1 Algorithm for homology of maps of $n$ -types

Let  $f: G_* \rightarrow G'_*$  be a morphism of simplicial groups. As we know, for  $n \geq 0$ ,  $H_n(-, \mathbb{Z})$  is a functor from the category of simplicial groups to the category of abelian groups. So  $f$  induces homomorphisms

$$H_n(f): H_n(G_*, \mathbb{Z}) \rightarrow H_n(G'_*, \mathbb{Z}) \text{ for all } n \geq 0.$$

In this section, we give an algorithm for computing a chain map between a chain complex for homology  $KG_*$  of  $G_*$  and a chain complex for homology  $KG'_*$  of  $G'_*$

$$f_*: KG_* \rightarrow KG'_*.$$

Then we use this chain map to compute  $H_n(f)$ .

From Theorem 3.2.8 we can find the integral homology of  $G_*$  by using the total complex of the bicomplex  $\mathcal{A}\overline{B}_*^{G_*}$

$$\begin{array}{ccccccc}
 & & & & & & \overset{q}{\vdots} \\
 & & & & & & \delta_3^2 \rightarrow \overline{B}_3^{G_0} \\
 \overline{B}_3^{G_3} & \xrightarrow{\delta_3^3} & \overline{B}_3^{G_2} & \xrightarrow{\delta_3^2} & \overline{B}_3^{G_1} & \xrightarrow{\delta_3^1} & \overline{B}_3^{G_0} \\
 \partial_3^3 \downarrow & & \partial_3^2 \downarrow & & \partial_3^1 \downarrow & & \partial_3^0 \downarrow \\
 \overline{B}_2^{G_3} & \xrightarrow{\delta_2^3} & \overline{B}_2^{G_2} & \xrightarrow{\delta_2^2} & \overline{B}_2^{G_1} & \xrightarrow{\delta_2^1} & \overline{B}_2^{G_0} \\
 \partial_2^3 \downarrow & & \partial_2^2 \downarrow & & \partial_2^1 \downarrow & & \partial_2^0 \downarrow \\
 \overline{B}_1^{G_3} & \xrightarrow{\delta_1^3} & \overline{B}_1^{G_2} & \xrightarrow{\delta_1^2} & \overline{B}_1^{G_1} & \xrightarrow{\delta_1^1} & \overline{B}_1^{G_0} \\
 \partial_1^3 \downarrow & & \partial_1^2 \downarrow & & \partial_1^1 \downarrow & & \partial_1^0 \downarrow \\
 \overline{B}_0^{G_3} & \xrightarrow{\delta_0^3} & \overline{B}_0^{G_2} & \xrightarrow{\delta_0^2} & \overline{B}_0^{G_1} & \xrightarrow{\delta_0^1} & \overline{B}_0^{G_0} \\
 \leftarrow \dots & & \dots & & \dots & & \dots
 \end{array}$$

with columns

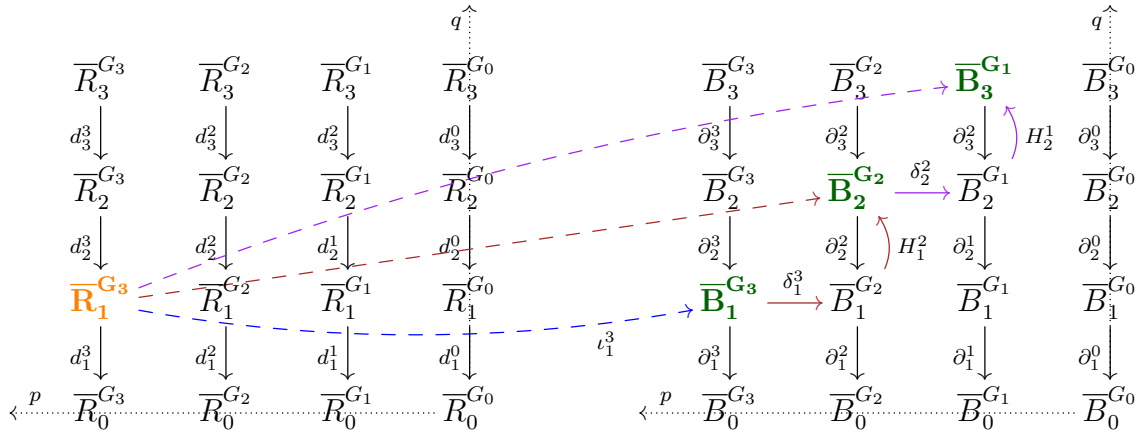
$$\overline{B}_*^{G_p}: \dots \rightarrow \overline{B}_3^{G_p} \xrightarrow{\partial_3^p} \overline{B}_2^{G_p} \xrightarrow{\partial_2^p} \overline{B}_1^{G_p} \xrightarrow{\partial_1^p} \overline{B}_0^{G_p}$$

the bar complex of  $G_p$  for all  $p \geq 0$ .

On the other hand, the morphism  $f$  induces homomorphisms

$$f_j^i: \overline{B}_j^{G_i} \rightarrow \overline{B}_j^{G'_i} \text{ for all } i, j \geq 0.$$

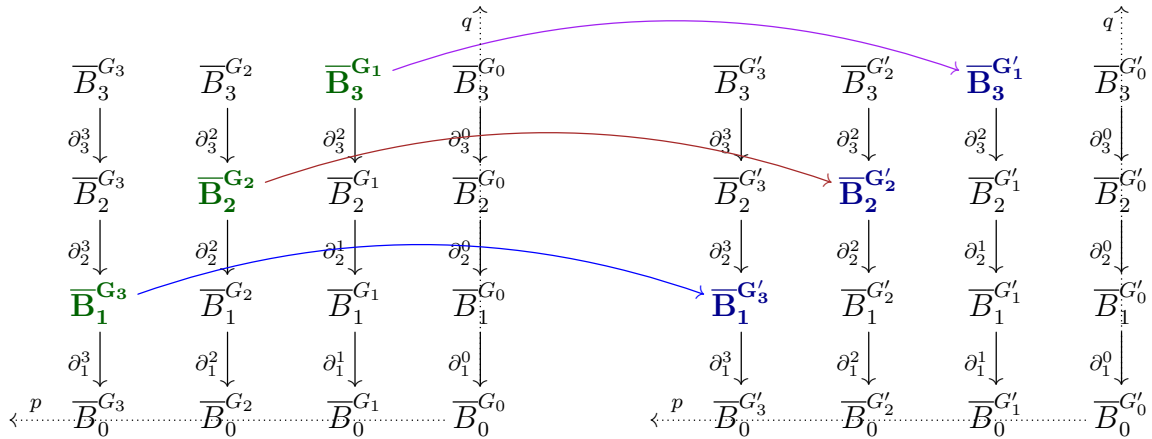




**Step 2.** We construct the chain map

$$\Phi_n: \bigoplus_{p+q=n} \overline{B}_p^{G_q} \rightarrow \bigoplus_{p+q=n} \overline{B}_p^{G'_q}$$

given by  $\Phi_n(x) = f_p^p(x)$  with  $x \in \overline{B}_p^{G_q}$ .



**Step 3.** We construct the chain map

$$\delta_n: \bigoplus_{p+q=n} \overline{B}_p^{G'_q} \rightarrow \bigoplus_{p+q=n} \overline{R}_p^{G'_q}$$

by using Equation 3.16. For example, the homomorphism  $\Delta_4$  sends  $x \in \overline{B}_1^{G'_3}$  to

$$\Delta_4(x) = x_1 + x_2 + x_3$$

with  $x_1 \in \overline{R}_1^{G'_3}, x_2 \in \overline{R}_2^{G'_2}, x_3 \in \overline{R}_3^{G'_1}$ . This is illustrated in the following diagram

$$\begin{array}{cccccccccccc}
& & & & \overset{q}{\uparrow} & & & & \overset{q}{\uparrow} & & & & & \\
& & & & \psi_3^1 & & & & & & & & & \\
\overline{B}_3^{G'_3} & \overline{B}_3^{G'_2} & \xrightarrow{\delta_2^3} & \overline{B}_3^{G'_1} & \overline{B}_3^{G'_0} & & \overline{R}_3^{G'_3} & \overline{R}_3^{G'_2} & \overline{R}_3^{G'_1} & \overline{R}_3^{G'_0} & & & & \\
d_3^3 \downarrow & d_3^2 \downarrow & \curvearrowright H_2^2 & d_3 \downarrow & d_3^0 \downarrow & & \partial_3^3 \downarrow & \partial_3^2 \downarrow & \partial_3^1 \downarrow & \partial_3^0 \downarrow & & & & \\
\overline{B}_2^{G'_3} & \overline{B}_2^{G'_2} & \xrightarrow{\delta_2^3} & \overline{B}_2^{G'_1} & \overline{B}_2^{G'_0} & & \overline{R}_2^{G'_3} & \overline{R}_2^{G'_2} & \overline{R}_2^{G'_1} & \overline{R}_2^{G'_0} & & & & \\
d_2^3 \downarrow & d_2^2 \downarrow & \curvearrowright H_1^3 & d_2^1 \downarrow & d_2^0 \downarrow & & \partial_2^3 \downarrow & \partial_2^2 \downarrow & \partial_2^1 \downarrow & \partial_2^0 \downarrow & & & & \\
\overline{B}_1^{G'_3} & \overline{B}_1^{G'_2} & & \overline{B}_1^{G'_1} & \overline{B}_1^{G'_0} & & \overline{R}_1^{G'_3} & \overline{R}_1^{G'_2} & \overline{R}_1^{G'_1} & \overline{R}_1^{G'_0} & & & & \\
d_1^3 \downarrow & d_1^2 \downarrow & & d_1^1 \downarrow & d_1^0 \downarrow & & \partial_1^3 \downarrow & \partial_1^2 \downarrow & \partial_1^1 \downarrow & \partial_1^0 \downarrow & & & & \\
\leftarrow^p & \overline{B}_0^{G'_3} & \overline{B}_0^{G'_2} & \overline{B}_0^{G'_1} & \overline{B}_0^{G'_0} & & \leftarrow^p & \overline{R}_0^{G'_3} & \overline{R}_0^{G'_2} & \overline{R}_0^{G'_1} & \overline{R}_0^{G'_0} & & & \\
& & & & & & & & & & & & & 
\end{array}$$

Finally, we compute the chain map  $f_n = \Delta_n \Phi_n \Pi_n$ .

### Algorithm 6.1.1.

**Input:** A morphism  $f: G_* \rightarrow G'_*$  of simplicial groups and an integer  $n \geq 0$ .

**Output:** A chain map  $f_*: KG_* \rightarrow KG'_*$  between a chain complex for homology of  $G_*$  and a chain complex for homology of  $G'_*$ .

**Procedure:** We implement the above three steps.

**Example 6.1.1.** Let  $f: \mathbb{Z}_4 \rightarrow \mathbb{Z}_2$  give by  $\overline{m} \mapsto \overline{m \bmod 2}$ . The following GAP session illustrates how to compute the homology map

$$H_4(f): H_4(K(\mathbb{Z}_4, 2), \mathbb{Z}) \rightarrow H_4(K(\mathbb{Z}_2, 2), \mathbb{Z}).$$

```

gap> Z4:=CyclicGroup(4);
gap> a:=Z4.1;;          ## Z4=<a>
gap> Z2:=CyclicGroup(2);
gap> b:=Z2.1;;          ## Z2=<b>
gap> f:=GroupHomomorphismByImages(Z4,Z2,[a],[b]);
gap> Kf:=EilenbergMacLaneSimplicialGroup(f,2,5);
Morphism of simplicial groups of length 5
gap> HKf:=ChainComplexOfSimplicialGroup(fK);
Chain map between complexes of length 5 .
gap> Homology(HKf,4);
C8 -> C4

```

These commands took 1 second.

# Chapter 7

## Persistent homology of 2-types

In this chapter, we introduce a new quasi-isomorphism invariant of crossed modules (see Definition 7.1.4, Theorem 7.1.10). It is called *persistent homology*. In addition, it might be used to determine whether two crossed modules are not quasi-isomorphic.

Let  $\partial: M \rightarrow P$  be a crossed module. We denote  $\pi_1(\partial)$  by  $G$  and denote  $\pi_2(\partial)$  by  $A$ .

## 7.1 Definition of persistent homology of crossed modules

**Definition 7.1.1.** Let  $\partial: M \rightarrow P$  be a crossed module. For any group  $H \leq G$  let  $\overline{H}$  be the preimage of  $H$  in  $P$ . Then the crossed module  $\partial$  restricts to a crossed module

$$\partial_H: M \rightarrow \overline{H}.$$

Let us call  $\partial_H$  the *covering crossed module* corresponding to the subgroup  $H$ .

Note that  $\pi_1(\partial_H) = H, \pi_2(\partial_H) = A$  and that there is an inclusion morphism of crossed modules

$$\begin{array}{ccc} M & \xrightarrow{=} & M \\ \downarrow \partial_H & & \downarrow \partial \\ \overline{H} & \longrightarrow & P \end{array}$$

Let  $\{\gamma_i G\}_{i \geq 1}$  be the lower central series of  $G$ . Then it gives rise to a sequence of inclusions of covering morphisms of crossed modules

$$\cdots \rightarrow \partial_{\gamma_i G} \rightarrow \partial_{\gamma_{i-1} G} \rightarrow \cdots \rightarrow \partial_{\gamma_3 G} \rightarrow \partial_{\gamma_2 G} \rightarrow \partial. \quad (7.1)$$

**Proposition 7.1.1.** Let  $(\mu, \eta)$  be a quasi-isomorphism between  $\partial: M \rightarrow P$  and  $\partial': M' \rightarrow P'$ . For  $i \geq 1$ , we define

$$\eta_i: \overline{\gamma_i G} \rightarrow \overline{\gamma_i G'} \text{ by } \eta_i(p) := \eta(p) \text{ for all } p \in \overline{\gamma_i G}.$$

Then  $(\mu, \eta_i)$  is a quasi-isomorphism between  $\partial_{\gamma_i G}: M \rightarrow \overline{\gamma_i G}$  and  $\partial'_{\gamma_i G'}: M' \rightarrow \overline{\gamma_i G'}$ .

**Proof.** Firstly, we need to prove that  $(\mu, \eta_i)$  is a morphism of crossed modules.

- Note that  $G = P/\text{Im } \partial$  and  $G' = P'/\text{Im } \partial'$ . By applying the Correspondence Theorem in group theory, we have

$$\overline{\gamma_i G}/\text{Im } \partial = \gamma_i G \text{ and } \overline{\gamma_i G'}/\text{Im } \partial' = \gamma_i G'.$$

Since  $(\mu, \eta)$  is a quasi-isomorphism,  $\eta$  induces the isomorphism  $\eta^*: G \xrightarrow{\cong} G'$ . Furthermore,  $\gamma_i G$  and  $\gamma_i G'$  are  $i$ th terms of the lower central series of  $G$  and  $G'$ , so  $\eta^*$  induces an isomorphism  $\eta_i^*: \gamma_i G \rightarrow \gamma_i G'$  or

$$\eta_i^*: \overline{\gamma_i G}/\text{Im } \partial \rightarrow \overline{\gamma_i G'}/\text{Im } \partial'$$

given by  $\eta_i^*(p \text{Im } \partial) := \eta^*(p \text{Im } \partial) = \eta(p) \text{Im } \partial'$  for all  $p \in \overline{\gamma_i G}$ .

Let  $p \in \overline{\gamma_i G}$  then  $p \text{Im } \partial \in \overline{\gamma_i G}/\text{Im } \partial$ , so  $\eta(p) \text{Im } \partial' \in \overline{\gamma_i G'}/\text{Im } \partial'$ . Thus  $\eta(p) \in \overline{\gamma_i G'}$  or  $\eta_i(p) \in \overline{\gamma_i G'}$ . This implies  $\eta_i$  is a homomorphism.

- It is easy to see that  $\partial_{\gamma_i G'} \mu = \eta_i \partial_{\gamma_i G}$ .
- Let  $m \in M$  and  $p \in \overline{\gamma_i G}$ , we have  $\mu({}^p m) = \eta({}^p) \mu(m) = \eta_i({}^p) \mu(m)$ .

Secondly, we prove  $(\mu, \eta_i)$  induces two isomorphisms  $\mu^*: \pi_2(\partial_{\gamma_i G}) \xrightarrow{\cong} \pi_2(\partial'_{\gamma_i G'})$  and  $\eta_i^*: \pi_1(\partial_{\gamma_i G}) \xrightarrow{\cong} \pi_1(\partial'_{\gamma_i G'})$ .

- We have  $\eta_i^*: \gamma_i G \rightarrow \gamma_i G'$  is an isomorphism; furthermore,  $\pi_1(\partial_{\gamma_i G}) = \gamma_i G$  and  $\pi_1(\partial'_{\gamma_i G'}) = \gamma_i G'$ . We conclude that  $\eta_i$  induces an isomorphism  $\eta_i^*: \pi_1(\partial_{\gamma_i G}) \xrightarrow{\cong} \pi_1(\partial'_{\gamma_i G'})$ .
- Since  $(\mu, \eta)$  is quasi-isomorphic,  $\mu$  induces  $\mu^*: A \xrightarrow{\cong} A'$ . Furthermore  $\pi_2(\partial_{\gamma_i G}) = A$  and  $\pi_2(\partial'_{\gamma_i G'}) = A'$  so  $\mu^*: \pi_2(\partial_{\gamma_i G}) \xrightarrow{\cong} \pi_2(\partial'_{\gamma_i G'})$ .  $\square$

**Lemma 7.1.2.** *Let  $(\mu, \eta)$  be a morphism between  $\partial: M \rightarrow P$  and  $\partial': M' \rightarrow P'$ . Then, for  $i \geq 1$ ,*

$$(\mu, \eta_i)(1_M, i_{i+1}) = (1_{M'}, i'_{i+1})(\mu, \eta_{i+1})$$

where  $i_{i+1}: \overline{\gamma_{i+1} G} \hookrightarrow \overline{\gamma_i G}$  and  $i'_{i+1}: \overline{\gamma_{i+1} G'} \hookrightarrow \overline{\gamma_i G'}$  are two inclusion homomorphisms. In other words, the following diagram commutes

$$\begin{array}{ccccc}
M & \xrightarrow{1_M} & M & & \\
\downarrow \partial_{\gamma_{i+1}G} & \searrow \mu & \downarrow \partial_{\gamma_i G} & \searrow \mu & \\
M & \xrightarrow{1_{M'}} & M & & \\
\downarrow \partial_{\gamma_{i+1}G'} & \searrow \eta_{i+1} & \downarrow \partial_{\gamma_i G'} & \searrow \eta_i & \\
\gamma_{i+1}G & \xrightarrow{i_{i+1}} & \gamma_i G & & \\
\downarrow \eta_{i+1} & \searrow \eta_{i+1} & \downarrow \eta_i & \searrow \eta_i & \\
\gamma_{i+1}G' & \xrightarrow{i'_{i+1}} & \gamma_i G' & & 
\end{array}$$

**Proof.** It is obvious.  $\square$

**Definition 7.1.2.** Let  $B$  be a  $G$ -module. For  $b \in B$  and  $g \in G$ , we define the commutator  $[b, g] = b^g b^{-1}$  and

$$[B, G] = \langle [b, g] \mid b \in B, g \in G \rangle$$

the subgroup of  $B$  generated by all commutators  $[b, g]$  with  $b \in B$  and  $g \in G$ .

**Lemma 7.1.3.** Let  $B$  be a  $G$ -module. Then  $[B, G]$  is also a  $G$ -module.

**Proof.** Let any  $b \in B$ ,  $g, h \in G$ , we have

$$\begin{aligned}
r^h [b, g] &= {}^h(b^g b^{-1}) = {}^h b {}^h (g b^{-1}) \\
&= {}^h b ({}^{hg} b^{-1}) = {}^h b (b^{-1} b) ({}^{hg} b^{-1}) \\
&= ({}^h b b^{-1}) (b ({}^{hg} b^{-1})) = (b^{-1} {}^h b) (b ({}^{hg} b^{-1})) \\
&= [b^{-1}, h] [b, hg] \in [B, G].
\end{aligned}$$

This implies  ${}^h [b, g] \in [B, G]$  for all  $b \in B$ ,  $h, g \in G$ ; furthermore,  $[b, g]$  is a generator of  $[B, G]$ . So  $[B, G]$  is a  $G$ -module.  $\square$

Let  $\partial: M \rightarrow P$  be a crossed module. Recall that  $A$  is a  $G$ -module with the action  $G$  on  $A$  defined by  ${}^g a := \tilde{g} a$  where  $\tilde{g}$  is an element chosen from the preimage of  $g$  in  $P$ . Moreover,  ${}^p a = {}^{p \text{Im } \partial} a$  for all  $p \in P, a \in A$ .

**Definition 7.1.3.** For  $i \geq 1$ , we define  $\beta_i A$  as follows:

$$\beta_1 A = A \text{ and } \beta_{i+1} A = [\beta_i A, G] \text{ for all } i \geq 1.$$

**Lemma 7.1.4.** For  $i \geq 1$ , we have

$$(i) \quad \beta_{i+1} A \leq \beta_i A.$$

(ii)  $\beta_i A \triangleleft M$ .

**Proof.** It is obvious. □

**Lemma 7.1.5.** *Let  $\partial: M \rightarrow P$  be a crossed module. Then, for  $i \geq 1$ ,*

$$\partial^{\beta_i A}: M/\beta_i A \rightarrow P \text{ defined by } \partial^{\beta_i A}(m\beta_i A) := \partial(m)$$

*together with the action  ${}^p(m\beta_i A) := ({}^p m)\beta_i A$  is a crossed module.*

**Proof.** We only need to prove that for  $p \in P$  and  $m\beta_i A \in M/\beta_i A$ , the definition  ${}^p(m\beta_i A) := ({}^p m)\beta_i A$  yields an action  $P$  on  $M/\beta_i A$ . This means, if  $m_1\beta_i A = m_2\beta_i A$  then  $({}^p m_1)\beta_i A = ({}^p m_2)\beta_i A$ .

Let  $m_1\beta_i A = m_2\beta_i A$ . There exist  $a \in \beta_i A$  such that  $m_1 = m_2 a$ . Then

$${}^p m_1 = {}^p(m_2 a) = {}^p m_2 {}^p a = {}^p m_2 ({}^{p \operatorname{Im} \partial} a).$$

Since  $\beta_i A$  is a  $G$ -module and  $p \operatorname{Im} \partial \in G$ , we have  ${}^{p \operatorname{Im} \partial} a \in \beta_i A$ . Thus  $({}^p m_1)\beta_i A = ({}^p m_2)\beta_i A$ . □

For any crossed module  $\partial: M \rightarrow P$  and  $i \geq 1$ , we obtain the following morphism of crossed modules.

$$\begin{array}{ccc} M & \twoheadrightarrow & M/\beta_i A \\ \downarrow \partial & & \downarrow \partial^{\beta_i A} \\ P & \xrightarrow{=} & P \end{array}$$

Moreover,  $\{\beta_i A\}_{i \geq 1}$  gives rise to a sequence of morphisms of crossed modules

$$\dots \rightarrow \partial^{\beta_i A} \rightarrow \partial^{\beta_{i-1} A} \rightarrow \dots \rightarrow \partial^{\beta_3 A} \rightarrow \partial^{\beta_2 A} \rightarrow \partial^{\beta_1 A}. \quad (7.2)$$

**Lemma 7.1.6.** *Let  $(\mu, \eta)$  be a quasi-isomorphism between  $\partial: M \rightarrow P$  and  $\partial': M' \rightarrow P'$ . For  $i \geq 1$ , we define*

$$\mu_i: \beta_i A \rightarrow \beta_i A' \text{ by } \mu_i(a) := \mu(a) \text{ for all } a \in \beta_i A.$$

*Then  $\mu_i$  is an isomorphism.*

**Proof.** We prove that  $\mu_i$  is an isomorphism by induction on  $i$ .

Since  $(\mu, \eta)$  is quasi-isomorphic, then  $\mu$  induces an isomorphism  $\mu_*: A \xrightarrow{\cong} A'$  with  $\mu_*(a) = \mu(a)$  for all  $a \in A$ . Note that  $\beta_1 A = A$ ,  $\beta_1 A' = A'$ . This implies  $\mu_1: \beta_1 A \rightarrow \beta_1 A'$  is an isomorphism.

We suppose that  $\mu_i: \beta_i A \rightarrow \beta_i A'$  is an isomorphism for all  $i \geq 1$ .

Let  $[a, g]$  be a generator of  $\beta_{i+1} A$  ( $a \in \beta_i A, g \in G$ ). Then

$$\begin{aligned} \mu_{i+1}([a, g]) &= \mu(a {}^g a^{-1}) = \mu(a) \mu({}^g a^{-1}) \\ &= \mu(a) \mu({}^{\tilde{g}} a^{-1}) = \mu(a) {}^{\eta(\tilde{g})} \mu(a^{-1}) \\ &= \mu(a) {}^{\eta(\tilde{g})} (\mu(a))^{-1} = \mu_i(a) {}^{\eta(\tilde{g})} (\mu_i(a))^{-1} \quad (\text{as } a \in \beta_i A) \\ &= \mu_i(a) {}^{(\eta(\tilde{g}) \text{ Im } \partial')} (\mu_i(a))^{-1} = [\mu_i(a), \eta(\tilde{g}) \text{ Im } \partial'] \in [\beta_i A', G'] = \beta_{i+1} A'. \end{aligned}$$

It is easy to see that  $\mu_{i+1}$  is the restriction of  $\mu_*$  to  $\beta_{i+1} A$ . So  $\mu_{i+1}$  is injective.

Let  $[a', g']$  be a generator of  $\beta_{i+1} A'$  ( $a' \in \beta_i A', g' \in G'$ ). Since  $\mu_i$  is an isomorphism there exists  $a \in \beta_i A$  such that  $\mu_i(a) = a'$ . Because  $(\mu, \eta)$  is quasi-isomorphic,  $\eta$  induces an isomorphism  $\eta_*: G \rightarrow G'$ . So there exists  $g \in G$  such that  $\eta_*(g) = g'$ . We have  $g = \tilde{g} \text{ Im } \partial$ ,  $\eta_*(g) = \eta_*(\tilde{g} \text{ Im } \partial) = \eta(\tilde{g}) \text{ Im } \partial'$ . This implies  $\eta(\tilde{g}) \text{ Im } \partial' = g'$ . We consider the element  $[a, g]$ . Clearly,  $[a, g] \in \beta_{i+1} A$ . Then

$$\begin{aligned} \mu_{i+1}([a, g]) &= \mu(a {}^g a^{-1}) = \mu(a) \mu({}^g a^{-1}) \\ &= \mu(a) \mu({}^{\tilde{g}} a^{-1}) = \mu(a) {}^{\eta(\tilde{g})} \mu(a^{-1}) \\ &= \mu(a) {}^{\eta(\tilde{g}) \text{ Im } \partial'} (\mu(a))^{-1} = a' {}^{g'} a'^{-1} \\ &= [a', g']. \end{aligned}$$

So  $\mu_{i+1}$  is surjective; furthermore,  $\mu_{i+1}$  is injective. Thus  $\mu_{i+1}$  is an isomorphism.  $\square$

**Proposition 7.1.7.** *Let  $(\mu, \eta)$  be a quasi-isomorphism between  $\partial: M \rightarrow P$  and  $\partial': M' \rightarrow P'$ . We define*

$$\bar{\mu}_i: M/\beta_i A \rightarrow M'/\beta_i A' \text{ by } \bar{\mu}_i(m\beta_i A) = \mu(m)\beta_i A'.$$

*Then  $(\bar{\mu}_i, \eta)$  is a quasi-isomorphism between  $\partial^{\beta_i A}: M/\beta_i A \rightarrow P$  and  $\partial'^{\beta_i A'}: M'/\beta_i A' \rightarrow P'$ .*

**Proof.** Firstly, we need to prove that  $(\bar{\mu}_i, \eta)$  is a morphism of crossed modules.

- Let  $m\beta_i A \in M/\beta_i A$ ,

$$\begin{aligned}\partial'^{\beta_i A'} \bar{\mu}_i(m\beta_i A) &= \partial'^{\beta_i A'} (\mu(m)\beta_i A') = \partial' \mu(m), \\ \eta \partial'^{\beta_i A'} (m\beta_i A) &= \eta \partial(m) = \partial' \mu(m).\end{aligned}$$

So  $\partial'^{\beta_i A'} \bar{\mu}_i = \eta \partial'^{\beta_i A'} (m\beta_i A)$ .

- Let  $m\beta_i A \in M/\beta_i A$  and  $p \in P$ .

$$\begin{aligned}\bar{\mu}_i({}^p(m\beta_i A)) &= \bar{\mu}_i({}^p(m)\beta_i A) = \mu({}^p(m)\beta_i A) = ({}^{\eta(p)}\mu(m))\beta_i A, \\ \eta({}^p)\bar{\mu}_i(m\beta_i A) &= \eta({}^p)(\mu(m)\beta_i A) = ({}^{\eta(p)}\mu(m))\beta_i A.\end{aligned}$$

So  $\bar{\mu}_i({}^p(m\beta_i A)) = \eta({}^p)\bar{\mu}_i(m\beta_i A)$ .

Secondly, we prove  $(\bar{\mu}_i, \eta)$  induces two isomorphisms  $\bar{\mu}_i^* : \pi_2(\partial^{\beta_i A}) \xrightarrow{\cong} \pi_2(\partial'^{\beta_i A'})$  and  $\eta^* : \pi_1(\partial^{\beta_i A}) \xrightarrow{\cong} \pi_1(\partial'^{\beta_i A'})$ .

- It is easy to see  $\pi_1(\partial^{\beta_i A}) = G$  and  $\pi_1(\partial'^{\beta_i A'}) = G'$ . Since  $(\bar{\mu}_i, \eta)$  is a quasi-isomorphism,  $\eta$  induces an isomorphism  $\eta_* : \pi_1(\partial^{\beta_i A}) \xrightarrow{\cong} \pi_1(\partial'^{\beta_i A'})$ .
- We have  $\pi_2(\partial^{\beta_i A}) = A/\beta_i A$  and  $\pi_2(\partial'^{\beta_i A'}) = A'/\beta_i A'$ . We consider the homomorphism  $\bar{\mu}_i^* : A/\beta_i A \rightarrow A'/\beta_i A'$  defined by  $\bar{\mu}_i^*(a\beta_i A) := \bar{\mu}_i(a\beta_i A) = \mu(a)\beta_i A$ . By using Lemma 7.1.6, we have the isomorphism  $\mu_i : \beta_i A \xrightarrow{\cong} \beta_i A'$ . Furthermore,  $\mu$  induces an isomorphism  $\mu^* : A \xrightarrow{\cong} A'$ . Thus  $\bar{\mu}_i^*$  is an isomorphism.  $\square$

**Lemma 7.1.8.** *Let  $(\mu, \eta)$  be a morphism of  $\partial : M \rightarrow P$  and  $\partial' : M' \rightarrow P'$ . Then for  $i \geq 1$ ,*

$$(p'_{i+1}, 1_{P'}) (\bar{\mu}_{i+1}, \eta_{i+1}) = (\bar{\mu}_i, \eta_i) (p_{i+1}, 1_P)$$

with  $p_{i+1} : M/\beta_{i+1} A \rightarrow M/\beta_i A$  defined by  $p_{i+1}(m\beta_{i+1} A) := m\beta_i A$  for all  $m \in M$  and  $p'_{i+1} : M'/\beta_{i+1} A' \rightarrow M'/\beta_i A'$  defined by  $p'_{i+1}(m\beta_{i+1} A') := m\beta_i A'$  for all  $m' \in M'$ . In other words, the following diagram commutes

$$\begin{array}{ccccc} M/\beta_{i+1} A & \xrightarrow{p_{i+1}} & M/\beta_i A & & \\ \downarrow \partial^{\beta_{i+1} A} & \searrow \bar{\mu}_{i+1} & \downarrow \partial^{\beta_i A} & \searrow \bar{\mu}_i & \\ P & \xrightarrow{\eta_{i+1}} & P & \xrightarrow{\eta_i} & P' \\ \downarrow \partial'^{\beta_{i+1} A'} & \downarrow \partial'^{\beta_{i+1} A'} & \downarrow \partial'^{\beta_i A'} & \downarrow \partial'^{\beta_i A'} & \\ P' & \xrightarrow{1_{P'}} & P' & \xrightarrow{1_{P'}} & P' \end{array}$$

**Proof.** It is obvious. □

**Lemma 7.1.9.** *Let  $p$  be a prime number and  $\partial: M \rightarrow P$  be a crossed module with  $A$  and  $G$   $p$ -groups. Then there exists an integer  $n$  such that  $\beta_n A = \mathbf{1}$ .*

**Proof.** We proceed by induction on the order of  $A$ . If  $|A| = 1$  then immediately  $\beta_1 A = \mathbf{1}$  (as  $\beta_1 A = A$ ).

Now suppose  $|A| > 1$ . Let

$$T = \text{Fix}_A(G) = \{a \in A \mid {}^g a = a \text{ for all } g \in G\}.$$

It is easy to see that  $T$  is a  $G$ -module and  $[T, G] = \mathbf{1}$ . From Lemma 5.2 [41], we obtain  $|T| \equiv |A| \pmod{p}$ . Since  $|A|$  has order of  $p$  power, so  $T$  also has order of  $p$  power. Clearly,  $T$  is a normal subgroup of  $M$ . So  $\partial$  induces the crossed module  $\bar{\partial}: M/T \rightarrow P$ . Note that  $\pi_1(\bar{\partial}) = G$  and  $\pi_2(\bar{\partial}) = A/T$ . We call  $f$  the natural homomorphism from  $M$  to  $M/T$ . We consider the following morphism of crossed modules

$$\begin{array}{ccc} M & \xrightarrow{f} & M/T \\ \downarrow \partial & & \downarrow \bar{\partial} \\ P & \xrightarrow{=} & P \end{array}$$

Now we prove  $f(\beta_i A) = \beta_i(A/T)$  for all  $i \geq 1$  by induction on  $i$ . For  $i = 1$ ,

$$f(\beta_1 A) = f(A) = A/T = \beta_1(A/T).$$

Suppose that  $f(\beta_i A) = \beta_i(A/T)$  for  $i \geq 1$ .

$$f(\beta_{i+1} A) = f([\beta_i A, G]) \subset [f(\beta_i A), G] = [\beta_i(A/T), G] = \beta_{i+1}(A/T).$$

Let  $[aT, g]$  be a generator of  $\beta_{i+1}(A/T)$ . Since  $aT \in \beta_i(A/T)$ , there exists  $x \in \beta_i A$  such that  $f(x) = aT$ . Clearly,  $f([x, g]) = [f(x), g] = [aT, g]$ . Therefore  $f(\beta_{i+1} A) = \beta_{i+1}(A/T)$ .

Since  $A, T$  have order of  $p$  power, then  $|A/T| < |A|$ . By induction hypothesis, there exists  $k$  such that  $\beta_k(A/T) = \mathbf{1}$ . This implies  $f(\beta_k A) = \mathbf{1}$ . Thus  $\beta_k A \subset T$ . We have

$$\beta_{k+1} A = [\beta_k A, G] \subset [T, G] = \mathbf{1}.$$

Therefore there is a number  $n$  such that  $\beta_n A = \mathbf{1}$ .  $\square$

Let  $p$  be prime number and  $\partial: M \rightarrow P$  be a crossed module with  $A$  and  $G$   $p$ -groups. Then there exist  $k, l$  such that  $\gamma_k G = \mathbf{1}, \beta_l A = \mathbf{1}$ . So the sequences (7.1) and (7.2) are of finite length and can be spliced together

$$\partial_{\gamma_k G} \rightarrow \partial_{\gamma_{k-1} G} \rightarrow \cdots \rightarrow \partial_{\gamma_2 G} \rightarrow \partial \rightarrow \partial^{\beta_{l-1} A} \rightarrow \cdots \rightarrow \partial^{\beta_2 A} \rightarrow \partial^{\beta_1 A}.$$

We set  $m = k + l - 1$  and change the notations for  $\partial_{\gamma_i G}, \partial^{\beta_i A}$  by  $\partial_i$ , we obtain a sequence of morphisms of crossed modules

$$\partial_1 \rightarrow \partial_2 \rightarrow \partial_3 \rightarrow \cdots \rightarrow \partial_{m-1} \rightarrow \partial_m.$$

We define this sequence to be the *homotopy lower series* of crossed module  $\partial$ .

We now give the main new definition of the chapter.

**Definition 7.1.4.** Let  $p$  be a prime number and  $\partial: M \rightarrow P$  be a crossed module with  $G, A$   $p$ -groups. By applying the functor  $H_n(-, \mathbb{F}_p)$  to the homotopy lower series

$$\partial_1 \rightarrow \partial_2 \rightarrow \partial_3 \rightarrow \cdots \rightarrow \partial_{m-1} \rightarrow \partial_m$$

of  $\partial$ , we obtain a sequence of linear maps of vector spaces over  $\mathbb{F}_p$ .

$$H_n(\partial_1, \mathbb{F}_p) \rightarrow H_n(\partial_2, \mathbb{F}_p) \rightarrow H_n(\partial_3, \mathbb{F}_p) \rightarrow \cdots \rightarrow H_n(\partial_{m-1}, \mathbb{F}_p) \rightarrow H_n(\partial_m, \mathbb{F}_p).$$

We let  $PH_n(G)$  be the upper triangular matrix  $PH_n(\partial) = (p_{ij})_{1 \leq i, j \leq m}$  with

- $p_{ij}$  = the rank of  $H_n(\partial_i, \mathbb{F}_p) \rightarrow H_n(\partial_j, \mathbb{F}_p)$  for  $i < j$ .
- $p_{ii}$  = the dimension of  $H_n(\partial_i, \mathbb{F}_p)$ .
- $p_{ij} = 0$  for  $i > j$ .

The entries  $p_{ij}$  of the matrix  $PH_n(\partial)$  are called the *persistent Betti numbers* of  $\partial$  at degree  $n$ .

**Theorem 7.1.10.** *Quasi-isomorphic crossed modules have identical persistent Betti numbers.*

**Proof.** Since  $\partial$  is quasi-isomorphic to  $\partial'$ , we obtain the homotopy lower series of  $\partial$  and  $\partial'$  have the same length.

$$\begin{aligned} \partial_1 &\longrightarrow \partial_2 \longrightarrow \partial_3 \cdots \longrightarrow \partial_{m-1} \longrightarrow \partial_m \\ \partial'_1 &\longrightarrow \partial'_2 \longrightarrow \partial'_3 \cdots \longrightarrow \partial'_{m-1} \longrightarrow \partial'_m. \end{aligned}$$

By using Lemma 7.1.2 and Lemma 7.1.8, we see that the following diagram commutes

$$\begin{array}{ccccccccc} \partial_1 & \longrightarrow & \partial_2 & \longrightarrow & \partial_3 & \cdots \longrightarrow & \partial_{m-1} & \longrightarrow & \partial_m \\ \downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow \\ \partial'_1 & \longrightarrow & \partial'_2 & \longrightarrow & \partial'_3 & \cdots \longrightarrow & \partial'_{m-1} & \longrightarrow & \partial'_m \end{array}$$

Applying the functor  $H_n(-, \mathbb{F}_p)$ , we obtain the commutative diagram as below

$$\begin{array}{ccccccccc} H_n(\partial_1, \mathbb{F}_p) & \xrightarrow{d_1} & H_n(\partial_2, \mathbb{F}_p) & \xrightarrow{d_2} & H_n(\partial_3, \mathbb{F}_p) & \cdots \longrightarrow & H_n(\partial_{m-1}, \mathbb{F}_p) & \xrightarrow{d_{m-1}} & H_n(\partial_m, \mathbb{F}_p) \\ \downarrow f_1 & & \downarrow f_2 & & \downarrow f_3 & & \downarrow f_{m-1} & & \downarrow f_m \\ H_n(\partial'_1, \mathbb{F}_p) & \xrightarrow{d'_1} & H_n(\partial'_2, \mathbb{F}_p) & \xrightarrow{d'_2} & H_n(\partial'_3, \mathbb{F}_p) & \cdots \longrightarrow & H_n(\partial'_{m-1}, \mathbb{F}_p) & \xrightarrow{d'_{m-1}} & H_n(\partial'_m, \mathbb{F}_p) \end{array}$$

For  $1 \leq i \leq m$ , by using Proposition 7.1.1 and Proposition 7.1.7, we obtain  $\partial_i$  is quasi-isomorphic to  $\partial'_i$ . This implies  $f_i: H_n(\partial_i, \mathbb{Z}) \rightarrow H_n(\partial'_i, \mathbb{Z})$  is an isomorphism. Thus  $H_n(\partial_i, \mathbb{F}_p)$  and  $H_n(\partial'_i, \mathbb{F}_p)$  have the same dimension.

For  $1 \leq i < j \leq m$ , since the above diagram commutes, we have

$$d'_{ij} f_i = f_j d_{ij}$$

with  $d_{ij} = d_i \dots d_{j-1}$  and  $d'_{ij} = d'_i \dots d'_{j-1}$ . Furthermore,  $f_i$  and  $f_j$  are isomorphisms, then the rank of  $H_n(\partial_i, \mathbb{F}_p) \rightarrow H_n(\partial_j, \mathbb{F}_p)$  equals to the rank of  $H_n(\partial'_i, \mathbb{F}_p) \rightarrow H_n(\partial'_j, \mathbb{F}_p)$ . Therefore  $\partial$  and  $\partial'$  have the same persistent Betti numbers.  $\square$

## 7.2 Algorithm for persistent homology of crossed modules

Let  $\partial: M \rightarrow P$  be a crossed module with  $G, A$   $p$ -groups. By the construction in Section 7.1, we obtain the homotopy lower series

$$\partial_1 \rightarrow \partial_2 \rightarrow \partial_3 \rightarrow \cdots \rightarrow \partial_{m-1} \rightarrow \partial_m$$

of  $\partial$ . Now we give an algorithm for computing the homotopy lower series of crossed modules.

### Algorithm 7.2.1.

**Input:** A crossed module  $\partial: M \rightarrow P$  with  $G, A$   $p$ -groups

**Output:** The homotopy lower series of  $\partial$ .

**Procedure:** We do the following steps.

- Implement the sequence (7.1)

$$\partial_{\gamma_k G} \rightarrow \partial_{\gamma_{k-1} G} \rightarrow \cdots \rightarrow \partial_{\gamma_2 G} \rightarrow \partial.$$

- Implement the sequence (7.2)

$$\partial \rightarrow \partial^{\beta_{l-1} A} \rightarrow \cdots \rightarrow \partial^{\beta_2 A} \rightarrow \partial^{\beta_1 A}.$$

- Combine the above two sequences.

Now we give an algorithm to compute the persistent homology of crossed modules.

### Algorithm 7.2.2.

**Input:** A crossed module  $\partial: M \rightarrow P$  with  $G, A$   $p$ -groups and an integer  $n \geq 0$ .

**Output:** The matrix of persistent Betti numbers of  $\partial$  at degree  $n$ .

**Procedure:** We do the following steps.

- Use Algorithm 7.2.1 to compute the homotopy lower series of  $\partial$ . This sequence is denoted by  $L^\partial$ .
- Applying the functor  $\gamma$  to  $L^\partial$  (see Algorithm 5.2.1), we obtain a sequence of morphisms of  $\text{cat}^1$ -groups. This sequence is denoted by  $LC^\partial$ .
- Applying functor  $\mathcal{N}$  to  $LC^\partial$  (see 5.2.3), we get a sequence of morphisms of simplicial groups. This sequence is denoted by  $\mathcal{N}L^\partial$ .
- Applying Algorithm 6.1.1 for every morphisms of  $\mathcal{N}L^\partial$ , we get a sequence of chain maps. We denote this sequence by  $KL^\partial$ .
- Take the tensor product  $KL^\partial$  with  $\mathbb{F}_p$ .
- Then we compute the persistent homology of the sequence of chain maps  $KL^\partial$ .

**Example 7.2.1.** The following GAP session illustrates how to compute the persistent homology of the 171th crossed module of order 72 at degree  $n = 2$ .

```
gap> X:=SmallCrossedModule(72,171);;
gap> Size(HomotopyGroup(X,1));
2
gap> Size(HomotopyGroup(X,2));
4
gap> PersistentHomologyOfCrossedModule(X,2);
[ [ 1, 1, 1, 0 ], [ 0, 2, 2, 1 ],
[ 0, 0, 2, 1 ], [ 0, 0, 0, 1 ] ]
```

These commands took 14 minutes.

Then

$$PH_2(X) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 2 & 2 & 1 \\ 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

**Example 7.2.2.** The following GAP session illustrates how to compute the persistent homology at degree 3 of the 6th crossed module and the 11th crossed module of order 16.

```
gap> X1:=SmallCrossedModule(16,6);;
gap> X2:=SmallCrossedModule(16,11);;
gap> PersistentHomologyOfCrossedModule(X1,3);
[ [ 1, 1, 0 ], [ 0, 3, 1 ], [ 0, 0, 1 ] ]
gap> PersistentHomologyOfCrossedModule(X2,3);
[ [ 1, 1, 0 ], [ 0, 1, 0 ], [ 0, 0, 1 ] ]
```

Since  $PH_3(X_1) \neq PH_3(X_2)$ ,  $X_1$  is not quasi-isomorphic to  $X_2$ .

# Further Works

In this thesis we have developed computational tools for the classification of 2-types. We also provided a classification of most of the 2-types of order  $m \leq 255$ . By using this result, in many cases, we can determine if two crossed modules are quasi-isomorphic or not. We will investigate further on the quasi-isomorphism of two crossed modules. In particular, we will use theory and computation to answer the following questions:

- How to determine whether two crossed modules are quasi-isomorphic or not?
- Let  $\partial$  and  $\partial'$  be quasi-isomorphic. How to find a finite sequence of quasi-isomorphisms

$$\partial \xrightarrow{\phi_1} \partial_1 \xleftarrow{\phi_2} \partial_2 \xrightarrow{\phi_3} \dots \xleftarrow{\phi_k} \partial'$$

connecting  $\partial$  to  $\partial'$ ?

Then we apply these answers to the classification of the 2-types that are not classified in this thesis.

Moreover, every 2-type  $X$  can be represented by the fundamental group  $\pi_1 X$ , the  $\pi_1 X$ -module  $\pi_2 X$  and a cohomology class  $\kappa \in H^3(\pi_1 X, \pi_2 X)$ . We will construct a crossed module corresponding to the 2-type  $X$ . Then we will use this crossed module to compute the integral homology of  $X$ .

Furthermore, we will apply the concept of persistent homology to the coclass theory. Such as:

- Computing the persistent homology  $PH_*(G)$  of groups  $G$  in the coclass graph  $\mathcal{G}(p, r)$ . This persistent homology is an infinite graded module defined in

[22] and is based on the work of H. Edelsbrunner, G. Carlsson and others in applied topology.

- In the paper [9] Carlson proved that there exist only finitely many isomorphism types of mod-2-cohomology rings of 2-groups of a fixed coclass. We will extend the finiteness result of Carlson to  $p$ -groups for odd primes  $p$ .
- Eick and Feichtenschlager [17] developed and implemented an algorithm in the GAP system to compute the Schur multipliers of almost all groups in an infinite coclass sequence simultaneously. Based on this, they also obtained some results on the low-dimensional mod  $p$  cohomology groups  $H^n(G_i, \mathbb{F}_p)$ ,  $n = 0, 1, 2$  where  $(G_i \mid i \in \mathbb{N})$  is an infinite coclass sequence. We will provide an algorithm for computing  $H^*(G_i, \mathbb{F}_p)$  for almost all  $i$ . We conjecture that **almost all** the rings  $H^*(G_i, \mathbb{F}_p)$  are isomorphic to each other.

Also, in this thesis, we have just focused on 2-types. We will extend the results of 2-types to  $n$ -types ( $n \geq 3$ ). In particular, we will

- Develop an algorithm for classification of  $n$ -types;
- Investigate the persistent homology of  $n$ -types.

# Bibliography

- [1] Alejandro Adem and R. James Milgram. *Cohomology of finite groups*, volume 309 of *Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences]*. Springer-Verlag, Berlin, second edition, 2004.
- [2] Murat Alp and Chris Wensley. *XMod, a GAP4 package for crossed modules and  $\text{cat}^1$ -groups, version 2.26*, 2013. (<http://www.maths.bangor.ac.uk/chda/gap4/xmod>).
- [3] Murat Alp and Christopher D. Wensley. Enumeration of  $\text{cat}^1$ -groups of low order. *Internat. J. Algebra Comput.*, 10(4):407–424, 2000.
- [4] Clemens Berger. Iterated wreath product of the simplex category and iterated loop spaces. *Adv. Math.*, 213(1):230–270, 2007.
- [5] Kenneth S. Brown. *Cohomology of groups*, volume 87 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 1982.
- [6] R. Brown. The twisted Eilenberg-Zilber theorems. *Celebrazioni Arch. Secolo XX*, page 3437, 1967.
- [7] R. Brown, D. L. Johnson, and E. F. Robertson. Some computations of non-abelian tensor products of groups. *J. Algebra*, 111(1):177–202, 1987.
- [8] Ronald Brown, Philip J. Higgins, and Rafael Sivera. *Nonabelian algebraic topology*, volume 15 of *EMS Tracts in Mathematics*. European Mathematical Society (EMS), Zürich, 2011. Filtered spaces, crossed complexes, cubical homotopy groupoids, With contributions by Christopher D. Wensley and Sergei V. Soloviev.
- [9] Jon F. Carlson. Coclass and cohomology. *J. Pure Appl. Algebra*, 200(3):251–266, 2005.

- 
- [10] P. Carrasco, A. M. Cegarra, and A. R.-Grandjeán. (Co)homology of crossed modules. *J. Pure Appl. Algebra*, 168(2-3):147–176, 2002. Category theory 1999 (Coimbra).
- [11] A. Clement. *Integral cohomology of finite Postnikov towers*. PhD thesis, University of Lausanne, Switzerland, 2002.
- [12] Marius Crainic. On the perturbation lemma, and deformations. *arXiv preprint math/0403266*, page 13, 2004.
- [13] T. Datuashvili and T. Pirashvili. On (co)homology of 2-types and crossed modules. *J. Algebra*, 244(1):352–365, 2001.
- [14] Paweł Dłotko, Tomasz Kaczynski, Marian Mrozek, and Thomas Wanner. Coreduction homology algorithm for regular CW-complexes. *Discrete Comput. Geom.*, 46(2):361–388, 2011.
- [15] Mathieu Dutour Sikirić and Graham Ellis. Wythoff polytopes and low-dimensional homology of Mathieu groups. *J. Algebra*, 322(11):4143–4150, 2009.
- [16] Mathieu Dutour Sikirić, Graham Ellis, and Achill Schürmann. On the integral homology of  $\mathrm{PSL}_4(\mathbb{Z})$  and other arithmetic groups. *J. Number Theory*, 131(12):2368–2375, 2011.
- [17] Bettina Eick and Dörte 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.
- [18] Graham Ellis. Computing group resolutions. *J. Symbolic Comput.*, 38(3):1077–1118, 2004.
- [19] Graham Ellis. Homological algebra programming. In *Computational group theory and the theory of groups*, volume 470 of *Contemp. Math.*, pages 63–74. Amer. Math. Soc., Providence, RI, 2008.
- [20] Graham Ellis. HAP – Homological Algebra Programming, Version 1.10.13, 2013. (<http://www.gap-system.org/Packages/hap.html>).
- [21] Graham Ellis, James Harris, and Emil Sköldberg. Polytopal resolutions for finite groups. *J. Reine Angew. Math.*, 598:131–137, 2006.

- 
- [22] Graham Ellis and Simon King. Persistent homology of groups. *J. Group Theory*, 14(4):575–587, 2011.
- [23] Graham Ellis and Le Van Luyen. Computational homology of  $n$ -types. *J. Symbolic Comput.*, 47(11):1309–1317, 2012.
- [24] Graham Ellis and Emil Sköldbberg. The  $K(\pi, 1)$  conjecture for a class of Artin groups. *Comment. Math. Helv.*, 85(2):409–415, 2010.
- [25] Graham Ellis and Gerald Williams. On the cohomology of generalized triangle groups. *Comment. Math. Helv.*, 80(3):571–591, 2005.
- [26] Graham J. Ellis. Homology of 2-types. *J. London Math. Soc. (2)*, 46(1):1–27, 1992.
- [27] F.Sergeraert. *Kenzo software system for computations in algebraic topology.* (<http://www-fourier.ujf-grenoble.fr/~sergerar/Kenzo>).
- [28] The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.7.2*, 2013.
- [29] Paul G. Goerss and John F. Jardine. *Simplicial homotopy theory.* Modern Birkhuser Classics. Birkhuser Verlag, Basel, 2009. Reprint of the 1999 edition [MR1711612].
- [30] David J. Green. *Gröbner bases and the computation of group cohomology*, volume 1828 of *Lecture Notes in Mathematics*. Springer-Verlag, Berlin, 2003.
- [31] Alexander Hulpke. Notes on computational group theory, 2010. (<http://www.math.colostate.edu/~hulpke/CGT/cgtnotes.pdf>).
- [32] C. R. Leedham-Green and M. F. Newman. Space groups and groups of prime-power order. I. *Arch. Math. (Basel)*, 35(3):193–202, 1980.
- [33] Jean-Louis Loday. Spaces with finitely many nontrivial homotopy groups. *J. Pure Appl. Algebra*, 24(2):179–202, 1982.
- [34] Saunders MacLane and J. H. C. Whitehead. On the 3-type of a complex. *Proc. Nat. Acad. Sci. U. S. A.*, 36:41–48, 1950.

- 
- [35] J. Peter May. *Simplicial objects in algebraic topology*. Chicago Lectures in Mathematics. University of Chicago Press, Chicago, IL, 1992. Reprint of the 1967 original.
- [36] Simona Paoli. (Co)homology of crossed modules with coefficients in a  $\pi_1$ -module. *Homology Homotopy Appl.*, 5(1):261–296, 2003.
- [37] Marc Röder. Geometric algorithms for resolutions for Bieberbach groups. In *Computational group theory and the theory of groups, II*, volume 511 of *Contemp. Math.*, pages 167–178. Amer. Math. Soc., Providence, RI, 2010.
- [38] A. Romero. *Effective homology and spectral sequences*. PhD thesis, Universidad de La Rioja, Spain, 2007.
- [39] Ana Romero, Graham Ellis, and Julio Rubio. Interoperating between computer algebra systems: computing homology of groups with **kenzo** and **GAP**. In *ISSAC 2009—Proceedings of the 2009 International Symposium on Symbolic and Algebraic Computation*, pages 303–310. ACM, New York, 2009.
- [40] Ana Romero and Julio Rubio. Computing the homology of groups: the geometric way. *J. Symbolic Comput.*, 47(7):752–770, 2012.
- [41] John S. Rose. *A course on group theory*. Cambridge University Press, 1978.
- [42] Joseph J. Rotman. *An introduction to homological algebra*. Universitext. Springer, New York, second edition, 2009.
- [43] Julio Rubio and Francis Sergeraert. Algebraic models for homotopy types. *Homology Homotopy Appl.*, 7(2):139–160, 2005.
- [44] Julio Rubio and Francis Sergeraert. Constructive homological algebra and applications. *arXiv preprint arXiv:1208.3816*, 2012.
- [45] Sebastian Thomas. (Co)homology of crossed modules. Diploma thesis, RWTH Aachen University, 2007.
- [46] C. T. C. Wall. Resolutions for extensions of groups. *Proc. Cambridge Philos. Soc.*, 57:251–255, 1961.
- [47] Charles A. Weibel. *An introduction to homological algebra*, volume 38 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, Cambridge, 1994.

# Index

- 1-type, 9
- $\phi$ -equivariant, 9
- $n$ -constructible
  - group, 18
  - simplicial group, 37
- $n$ -simplices, 5
- $n$ -type, 26
- HAP complex, 20
- HAP resolution, 19
- sequence, 3
- alternating chain complex, 7
- augmentation, 9
- bar complex, 20
- bar resolution, 20
- bicomplex, 4
- bisimplicial set, 5
- $\text{cat}^1$ -group, 49
- $\text{cat}^1$ -group structure on group, 55
- category
  - $\text{cat}^1$ -groups, 50
  - crossed modules, 49
  - simplicial objects, 5
- chain complex, 3
- chain homotopic, 3
- chain homotopy, 3
- chain homotopy equivalence, 4
- chain map, 3
- class, 12
- commutator, 85
- contracting homotopy
  - free  $\mathbb{Z}G$ -resolution, 9
- covering crossed module, 83
- crossed module, 48
- degeneracy map, 5
- dihedral group, 10
- Eilenberg-Mac Lane simplicial group, 40
- face map, 5
- finitely generated
  - free  $\mathbb{Z}G$ -resolution, 18
- free  $\mathbb{Z}G$ -resolution of  $\mathbb{Z}$ , 9
- homology
  - chain complex, 3
- homology of groups, 9
- homotopy crossed module, 70
- homotopy equivalence data, 26
- homotopy groups
  - $\text{cat}^1$ -group, 61
  - crossed module, 48
  - simplicial group, 6
- homotopy lower series
  - crossed module, 90
- integral homology
  - crossed module, 67
  - simplicial group, 32
- isomorphism

- cat<sup>1</sup>-groups, 50
- crossed modules, 49
- lower central series, 12
- Moore complex, 6
- morphism
  - cat<sup>1</sup>-groups, 50
  - crossed modules, 49
- nerve
  - cat<sup>1</sup>-group, 51
  - category, 5
  - group, 6
- order
  - 2-type, 70
  - cat<sup>1</sup>-group, 49
  - crossed module, 48
  - quasi-isomorphism class, 70
- persistent Betti numbers
  - crossed module, 90
  - group, 13
- persistent homology
  - coclass tree, 18
- perturbation
  - homotopy equivalence data, 26
- quasi-isomorphic
  - cat<sup>1</sup>-groups, 62
  - chain complexes, 4
  - crossed modules, 61
- quasi-isomorphism
  - chain map, 4
  - morphism of cat<sup>1</sup>-groups, 61
  - morphism of crossed modules, 61
- quasi-isomorphism classes, 61
- rank, 12
- simplicial
  - group, 5
  - identities, 5
  - map, 5
  - objects, 5
  - set, 5
- small perturbation, 26
- total complex, 4
- weak equivalence
  - morphism of simplicial groups, 7
- weakly equivalent
  - simplicial groups, 7

# Chapter 8

## Appendix: GAP code

## 8.1 Data types

### 1. Simplicial group

A simplicial group  $(G_*, d, s)$  is represented by a component object  $G$  with the following components:

- $G!.groupsList(n)$ : is a function that returns the group  $G_n$ .
- $G!.boundariesList(n, i)$ : is a function that returns the face map  $d_n^i: G_n \rightarrow G_{n-1}$ .
- $G!.degeneraciesList(n, i)$ : is a function that returns the degeneracy map  $s_n^i: G_n \rightarrow G_{n+1}$ .
- $G!.properties$ : is a list of pairs [“name”,value] where “name” is a string and value is a numerical or boolean value.

### 2. $\text{Cat}^1$ -group

A  $\text{cat}^1$ -group  $(G, s, t)$  is represented as a component object  $C$  with the following components:

- $C!.sourceMap$ : is the group homomorphism  $s: G \rightarrow G$ .
- $C!.targetMap$ : is the group homomorphism  $t: G \rightarrow G$ .

### 3. Crossed module

A crossed module  $\partial: M \rightarrow P$  is represented as a component object  $X$  with the following components:

- $X!.map$ : is the group homomorphism  $\partial: M \rightarrow P$ .
- $X!.action(p, m)$ : is a function that returns the image  ${}^p m$  of  $m$  under the action of  $p$ .

### 4. Morphism of $\text{cat}^1$ -groups

A morphism of  $\text{cat}^1$ -groups  $\phi: (G, s, t) \rightarrow (G', s', t')$  is represented as a component object  $F$  with the following components:

- $F!.source$ : is the  $\text{cat}^1$ -group  $(G, s, t)$ .
- $F!.target$ : is the  $\text{cat}^1$ -group  $(G', s', t')$ .
- $F!.mapping$ : is the group homomorphism  $\phi: G \rightarrow G'$ .

## 5. Morphism of crossed modules

A morphism of crossed modules

$$\begin{array}{ccc} M & \xrightarrow{\mu} & M' \\ \partial \downarrow & & \downarrow \partial' \\ P & \xrightarrow{\eta} & P' \end{array}$$

is represented as a component object  $F$  with the following components:

- $F!.source$ : is the crossed module  $\partial: M \rightarrow P$ .
- $F!.target$ : is the crossed module  $\partial': M' \rightarrow P'$ .
- $F!.mapping(n)$ : is a function that returns the group homomorphism  $\mu: M \rightarrow M'$  if  $n = 1$  and returns the group homomorphism  $\eta: P \rightarrow P'$  if  $n = 2$ .

## 6. Morphism of simplicial groups

A morphism of simplicial groups  $f: G_* \rightarrow G'_*$  is represented as a component object  $F$  with the following components:

- $F!.source$ : is the simplicial group  $G_*$ .
- $F!.target$ : is the simplicial group  $G'_*$ .
- $F!.mapping(n)$ : is a function that returns the group homomorphism  $f_n: G_n \rightarrow G'_n$ .
- $F!.properties$ : is a list of pairs [“name”, value] where “name” is a string and value is a numerical or boolean value.

## 8.2 List of functions

### 1. BarResolutionEquivalence(R) (see GAP code 8.3.1)

- Input: A HAP resolution  $R_*^G$  of group  $G$ .
- Output: A component object  $HE$  with components
  - $HE!.phi(n, w)$ : is a function which inputs an integer  $n \geq 0$  and an element  $w$  in  $B_n^G$ . It returns the image of  $w$  in  $R_n^G$  under a chain map  $\phi: B_n^G \rightarrow R_n^G$ .
  - $HE!.psi(n, w)$ : is a function which inputs an integer  $n \geq 0$  and an element  $w$  in  $R_n^G$ . It returns the image of  $w$  in  $B_n^G$  under a chain map  $\psi_n: R_n^G \rightarrow B_n^G$ .
  - $HE!.equiv(n, w)$ : is a function which inputs an integer  $n \geq 0$  and an element  $w$  in  $B_n^G$ . It returns the image of  $w$  in  $B_{n+1}^G$  under a homomorphism  $H_n: B_n^G \rightarrow B_{n+1}^G$  satisfying

$$w - \psi_n \phi_n(w) = d_{n+1} H_n(w) + H_{n-1} d_n(w)$$

where  $d_n: B_n^G \rightarrow B_{n-1}^G$  is the boundary homomorphism in the bar resolution. (See Algorithm 2.3.4.)

### 2. BarComplexEquivalence(R) (see GAP code 8.3.2)

- Input: A HAP resolution  $R_*^G$  of group  $G$ .
- Output: It first constructs the chain complexes  $\overline{R}_*^G := R_*^G \otimes_{\mathbb{Z}G} \mathbb{Z}$  and  $\overline{B}_*^G := B_*^G \otimes_{\mathbb{Z}G} \mathbb{Z}$ . The function returns a component object  $HE$  with components
  - $HE!.phi(n, w)$ : is a function which inputs an integer  $n \geq 0$  and an element  $w$  in  $\overline{B}_n^G$ . It returns the image of  $w$  in  $\overline{R}_n^G$  under a chain map  $\phi_n: \overline{B}_n^G \rightarrow \overline{R}_n^G$ .
  - $HE!.psi(n, w)$ : is a function which inputs an integer  $n \geq 0$  and an element  $w$  in  $\overline{R}_n^G$ . It returns the image of  $w$  in  $\overline{B}_n^G$  under a chain map  $\psi_n: \overline{R}_n^G \rightarrow \overline{B}_n^G$ .

- *HE!.equiv*( $n, w$ ): is a function which inputs an integer  $n \geq 0$  and an element  $w$  in  $\overline{B}_n^G$ . It returns the image of  $w$  in  $\overline{B}_{n+1}^G$  under a homomorphism  $H_n: \overline{B}_n^G \rightarrow \overline{B}_{n+1}^G$  satisfying

$$w - \psi_n \phi_n(w) = d_{n+1} H_n(w) + H_{n-1} d_n(w)$$

where  $d_n: \overline{B}_n^G \rightarrow \overline{B}_{n-1}^G$  is the boundary homomorphism in the bar complex.

(See Algorithm 2.3.5.)

### 3. ChainComplexOfSimplicialGroup(X) (see GAP code 8.3.3)

- Input: A simplicial group  $X = G$ , or a morphism of simplicial groups  $X = (f: G_1 \rightarrow G_2)$ , or a sequence of morphisms of simplicial groups  $X = (G_1 \xrightarrow{f_1} G_2 \xrightarrow{f_2} \dots \xrightarrow{f_{n-1}} G_n)$ .
- Output: The image of the input under the map

$$K: (\text{Simplicial groups}) \rightarrow (\text{Chain complexes}).$$

(See Algorithm 3.2.1, 6.1.1.)

### 4. EilenbergMacLaneSimplicialGroup(X,n,l) (see GAP code 8.3.4)

- Input: An abelian group  $X = A$  or a homomorphism of abelian groups  $X = (f: A \rightarrow B)$ , and two integers  $n \geq 2, l \geq 1$ .
- Output: The Eilenberg-MacLane simplicial group  $K(A, n)$  of length  $l$  or the morphism  $f_*: K(A, n) \rightarrow K(B, n)$  of simplicial groups of length  $l$  by applying the functor

$$K(-, n): (\text{Abelian groups}) \rightarrow (\text{Simplicial abelian groups}).$$

(See Algorithms 4.1.1, 4.1.2.)

### 5. CrossedModuleByAutomorphismGroup(G) (see GAP code 8.3.5)

- Input: A group  $G$ .

- Output: The crossed module  $\partial: G \rightarrow \text{Aut}(G)$ .  
(See Example 5.1.1.)

### 6. `CrossedModuleByNormalSubgroup(G,N)` (see GAP code 8.3.6)

- Input: A group  $G$  with a normal subgroup  $N$ .
- Output: The inclusion crossed module  $i: N \hookrightarrow G$ .  
(See Example 5.1.1.)

### 7. `Order(X)` (see GAP code 8.3.7)

- Input: A crossed module  $X$ .
- Output: The order of  $X$ .  
(See Definition 5.1.2.)

### 8. `HomotopyGroup(X,n)` (see GAP code 8.3.8)

- Input: A crossed module  $X$  and an integer  $n = 1, 2$ .
- Output: The  $n$ th homotopy group of  $X$ .  
(See Definition 5.1.3.)

### 9. `CatOneGroupByCrossedModule(X)` (see GAP code 8.3.9)

- Input: A crossed module  $X$ , or a morphism of crossed modules  $X = (f: X_1 \rightarrow X_2)$  or a sequence of morphisms of crossed modules  $X = (X_1 \xrightarrow{f_1} X_2 \xrightarrow{f_2} \dots \xrightarrow{f_{n-1}} X_n)$ .
- Output: The image of the input under the functor

$$\lambda: (\text{Crossed modules}) \rightarrow (\text{Cat}^1\text{-groups}).$$

(See Proposition 5.2.2.)

### 10. `CrossedModuleByCatOneGroup(X)` (see GAP code 8.3.10)

- Input: A  $\text{cat}^1$ -group  $X$ , or a morphism of  $\text{cat}^1$ -groups  $X = (f: C_1 \rightarrow C_2)$ , or a sequence of morphisms of  $\text{cat}^1$ -groups  $X = (C_1 \xrightarrow{f_1} C_2 \xrightarrow{f_2} \dots \xrightarrow{f_{n-1}} C_n)$ .
- Output: The image of the input under the functor

$$\gamma: (\text{Cat}^1\text{-groups}) \rightarrow (\text{Crossed modules}).$$

(See Proposition 5.2.3.)

### 11. NerveOfCatOneGroup(X,n) (see GAP code 8.3.11)

- Input: A  $\text{cat}^1$ -group  $X = C$ , or a morphism of  $\text{cat}^1$ -groups  $X = (f: C_1 \rightarrow C_2)$ , or a sequence of morphisms of  $\text{cat}^1$ -groups  $X = (C_1 \xrightarrow{f_1} C_2 \xrightarrow{f_2} \dots \xrightarrow{f_{n-1}} C_n)$ , and an integer  $n \geq 0$ .
- Output: The image of the input of length  $n$  under the functor

$$\mathcal{N}: (\text{Cat}^1\text{-groups}) \rightarrow (\text{Simplicial groups}).$$

(See Proposition 5.2.6.)

### 12. CatOneGroupsByGroup(G) (see GAP code 8.3.12)

- Input: A finite group  $G$ .
- Output: A list of all non-isomorphic  $\text{cat}^1$ -group structures on group  $G$ .  
(See Algorithm 5.3.1.)

### 13. NumberSmallCatOneGroups(arg) (see GAP code 8.3.13)

- Input: A positive integer  $m \leq 255$ , or two positive integers  $m, k$  with  $m \leq 255$ .
- Output: The number of  $\text{cat}^1$ -groups of order  $m$ , or the number of non-isomorphic  $\text{cat}^1$ -group structures on the  $k$ th group of order  $m$ .

### 14. SmallCatOneGroup (see GAP code 8.3.14)

- Input: Three positive integers  $m, k, i$  with  $m \leq 255$ .

- Output: The  $i$ th  $\text{cat}^1$ -group structure on the  $k$ th group of order  $m$ .

**15. IsomorphismCatOneGroups(C,D)** (see GAP code [8.3.15](#))

- Input: Two finite  $\text{cat}^1$ -groups  $C, D$ .
- Output: An isomorphism between  $C$  and  $D$  if they are isomorphic and *fail* otherwise.  
(See Algorithm [5.3.2](#).)

**16. IdCatOneGroup(C)** (see GAP code [8.3.16](#))

- Input: A  $\text{cat}^1$ -group  $C$  of order less than or equal to 255.
- Output: A triple  $(m, k, i)$  where  $C$  is isomorphic to the  $i$ th  $\text{cat}^1$ -group structure on the  $k$ th group of order  $m$ .  
(See Algorithm [5.3.3](#).)

**17. NumberSmallCrossedModules(m)** (see GAP code [8.3.17](#))

- Input: A positive integer  $m \leq 255$ .
- Output: The number of crossed modules of order  $m$ .

**18. SmallCrossedModule(m,k)** (see GAP code [8.3.18](#))

- Input: Two positive integers  $m, k$  with  $m \leq 255$ .
- Output: The  $k$ th crossed module of order  $m$ .

**19. IsomorphismCrossedModules(XC,XD)** (see GAP code [8.3.19](#))

- Input: Two finite crossed modules  $XC, XD$ .
- Output: An isomorphism between  $XC$  and  $XD$  if they are isomorphic and *fail* otherwise.  
(See Algorithm [5.3.4](#).)

**20. IdCrossedModule(X)** (see GAP code [8.3.20](#))

- Input: A finite crossed module  $X$  of order less than or equal to 255.
- Output: A pair  $(m, k)$  where  $X$  is isomorphic to the  $k$ th crossed module of order  $m$ .  
(See Algorithm [5.3.5](#).)

**21. SubQuasiIsomorph(C)** (see GAP code [8.3.21](#))

- Input: A finite  $\text{cat}^1$ -group  $C$ .
- Output: A sub  $\text{cat}^1$ -group  $D$  of  $C$  such that  $D$  is quasi-isomorphic to  $C$ .

**22. QuotientQuasiIsomorph(C)** (see GAP code [8.3.22](#))

- Input: A finite  $\text{cat}^1$ -group  $C$ .
- Output: A quotient  $\text{cat}^1$ -group  $D$  of  $C$  such that  $D$  is quasi-isomorphic to  $C$ .

**23. QuasiIsomorph(X)** (see GAP code [8.3.23](#))

- Input: A finite crossed module or a finite  $\text{cat}^1$ -group  $X$ .
- Output: A crossed module or  $\text{cat}^1$ -group  $QX$  such that  $QX$  is quasi-isomorphic to  $X$  and  $|QX| \leq |X|$ .  
(See Algorithms [5.4.1](#), [5.4.2](#).)

**24. Homology(X,n)** (see GAP code [8.3.24](#))

- Input: A crossed module  $X$  and an integer  $n \geq 0$ .
- Output: The integral homology  $H_n(X, \mathbb{Z})$ .  
(See Definition [5.5.1](#) and Algorithm [5.5.1](#).)

**25. HomotopyCrossedModule(X)** (see GAP code [8.3.25](#))

- Input: A crossed module  $X$

- Output: The homotopy crossed module  $\pi_2(X) \xrightarrow{0} \pi_1(X)$  of  $X$ .  
(See Definition 5.6.1.)

**26. NumberSmallQuasiCrossedModules( $m$ )** (see GAP code 8.3.26)

- Input: A positive integer  $m \leq 255$ .
- Output: The number of quasi-isomorphism classes of order  $m$ .

**27. SmallQuasiCrossedModule( $m,k$ )** (see GAP code 8.3.27)

- Input: Two positive integers  $m, k$  with  $m \leq 255$ .
- Output: The smallest representative of the  $k$ th quasi-isomorphism classes of order  $m$ .

**28. IdQuasiCrossedModule( $X$ )** (see GAP code 8.3.28)

- Input: A finite crossed module  $X$ .
- Output: If successful in finding the smallest representative of the quasi-isomorphism class of  $X$ , it outputs a pair of integers  $(m, k)$  with  $m$  the order of the quasi-isomorphism class of  $X$  and  $k$  the number uniquely identifying this class, and *fail* otherwise.  
(See Algorithm 5.6.1.)

**29. HomotopyLowerCentralSeriesOfCrossedModule( $X$ )** (see GAP code 8.3.29)

- Input: A crossed module  $X$  with  $\pi_1 X, \pi_2 X$   $p$ -groups.
- Output: The homotopy lower central series of  $X$ .  
(See Algorithm 7.2.1.)

**30. PersistentHomologyOfCrossedModule( $X,n$ )** (see GAP code 8.3.30)

- Input: A crossed module  $X$  with  $\pi_1 X, \pi_2 X$   $p$ -groups and an integer  $n \geq 0$ .
- Output: The matrix of persistent Betti numbers of  $X$  at degree  $n$ .  
(See Algorithm 7.2.2.)

## 8.3 GAP Code

### 8.3.1 BarResolutionEquivalence(R)

```
#####
#0
#F BarResolutionEquivalence
## Input:  A HAP resolution of group G
## Output: A ZG-equivariant chain homotopy equivalence between the bar
##         resolution B^G and the HAP resolution R^G
##
InstallGlobalFunction(BarResolutionEquivalence,function(R)
local
    nElts,Elts,e,nR,n,k,
    SearchPosition,AddElement,HapHomotopy,BarHomotopy,BarBoundary,
    PsiBasis,BoundBasis,TmpPsi,tmp1,tmp2,sign,base,g,Stmp,
    Phi,Psi,Equiv;

    Elts:=R!.elts;
    nElts:=Length(Elts);
    e:=Identity(R!.group);
    nR:=EvaluateProperty(R,"length");

#####
#1
#F SearchPosition
## Input:  An element g of G
## Output: The position of g in Elts
##
SearchPosition:=function(g)
local  n,i;

    n:=Length(Elts);
    for i in [1..n] do
        if Elts[i]=g then
            return i;
        fi;
    od;
    Add(Elts,g);          #These two lines added by Graham
    return n+1;          #
end;
##
##### end of SearchPosition #####

#####
#1
#F AddElement
## Input:  A list L=[[m_1,h_1,g_11,...,g_1n],...,[m_k,h_k,g_k1,
##         ...,g_kn]] and an element x=[m',h',g'_1,...,g'_n]
## Output: Add the element x into the list L
##
AddElement:=function(L,x)
local  sx,nx,nL,flag,i,j;

    sx:=StructuralCopy(x);
    nx:=Length(sx);
    nL:=Length(L);
    for i in [1..nL] do
```

```

    flag:=0;
    for j in [2..nx] do
      if L[i][j]<>sx[j] then
        flag:=1;
        break;
      fi;
    od;
    if flag=0 then
      L[i][1]:=L[i][1]+sx[1];
      if L[i][1]=0 then
        Remove(L,i);
      fi;
      return;
    fi;
  od;
  Add(L,sx);
end;
##
##### end of AddElement #####

#####
#1
#F BarBoundary
## Input:  A word w=[m_1,h_1,g_11,...,g_1n],..., [m_k,h_k,g_k1,...
##         ,g_kn] and an integer n>=0
## Output: The image of w under the boudary map d_n:B_n->B_{n-1}
##
BarBoundary:=function(n,w)
local i,j,tmp,x,Rew;

  Rew=[];
  for x in w do

    ##### Compute 0 #####
    tmp=[x[1],x[2]*x[3]];
    for j in [2..n] do
      Add(tmp,x[j+2]);
    od;
    AddElement(Rew,tmp);

    ##### Compute 1 -> n-1 #####
    for i in [1..n-1] do
      tmp=[(-1)^i*x[1],x[2]];
      for j in [1..i-1] do
        Add(tmp,x[j+2]);
      od;
      Add(tmp,x[i+2]*x[i+3]);
      for j in [i+2..n]do
        Add(tmp,x[j+2]);
      od;
      AddElement(Rew,tmp);
    od;

    ##### Compute n #####
    tmp=[(-1)^n*x[1],x[2]];
    for j in [1..n-1] do
      Add(tmp,x[j+2]);
    od;
    AddElement(Rew,tmp);
  end;
end;

```

```

    od;
return Rew;
end;
##
##### end of BarBoundary #####

#####
#1
#F HapHomotopy
## Input:  A word w=[[m_1,e_1,pos_1],...,[m_k,e_k,pos_k]] and
##         an integer n>=0
## Output: The image of w under the homotopy map h_n: R_n->R_{n+1}
##
HapHomotopy:=function(n,w)
local  Rew, x, Hw,iHw,m;

    Rew:=[];
    for x in w do
        m:=x[1];
        Hw:=R!.homotopy(n,[x[2],x[3]]);
        for iHw in Hw do
            if iHw[1]>0 then
                AddElement(Rew,[m,iHw[1],iHw[2]]);
            else
                AddElement(Rew,[-m,-iHw[1],iHw[2]]);
            fi;
        od;
    od;
    return Rew;
end;
##
##### end of HapHomotopy #####

#####
#1
#F BarHomotopy
## Input:  A word w=[[m_1,h_1,g_11,...,g_1n],...,[m_k,h_k,g_k1,
##         ...,g_kn]] and an intger number n>=0
## Output: The image of w under the homotopy map h_n: B_n->B_{n+1}
##
BarHomotopy:=function(n,w)
local  i,x,Rew,tmp;

    Rew:=[];
    for x in w do
        tmp:=[x[1],e,x[2]];
        for i in [1..n] do
            Add(tmp,x[i+2]);
        od;
        AddElement(Rew,tmp);
    od;
    return Rew;
end;
##
##### end of BarHomotopy #####

##### Compute the image of basis of R under the map psi:R->B
PsiBasis:=List([0..nR],x->[]);
PsiBasis[1][1]:=[[1,e]];
##### [0+1] [1]

```

```

for n in [1..nR] do
  for k in [1..R!.dimension(n)] do
    TmpPsi:=[];
    BoundBasis:=R!.boundary(n,k);      ##ex: [[2,3], [-3,5]]
    for tmp1 in BoundBasis do
      if tmp1[1]<0 then
        sign:=-1;
        base:=-tmp1[1];
      else
        sign:=1;
        base:=tmp1[1];
      fi;
      g:=Eelts[tmp1[2]];
      Stmp:=StructuralCopy(PsiBasis[n][base]);
      for tmp2 in Stmp do
        tmp2[1]:=sign*tmp2[1];
        tmp2[2]:=g*tmp2[2];
      od;
      Append(TmpPsi,Stmp);
    od;
    PsiBasis[n+1][k]:=BarHomotopy(n-1,TmpPsi);
  od;
od;

#####
#1
#F Psi
## Input: A word w:=[[m1,e1,pos1],...,[mk,ek,posk]] and n>=0
## Output: The image of w under the map psi_n: R_n->B_n
##
Psi:= function(n,w)
local Rew,m,h,x,u,Psix;

  Rew:=[];
  for x in w do
    m:=x[1];
    h:=Eelts[x[3]];
    Psix:=StructuralCopy(PsiBasis[n+1][x[2]]);
    for u in Psix do
      u[1]:=m*u[1];
      u[2]:=h*u[2];
      AddElement(Rew,u);
    od;
  od;
  return Rew;
end;
##
##### end of Psi #####

#####
#1
#F Phi
## Input: A word w =[[m1,h1,g11],...,[mk,hk,gk1],...,[gkn]]
## Output: The image of w under the map phi_n: B_n->R_n
##
Phi:=function(n,w)
local x,Rew,Rex,h,u,cw;

  cw:=StructuralCopy(w);

```

```

Rew:=[];
if n=0 then
  for x in cw do
    AddElement(Rew,[x[1],1,SearchPosition(x[2])]);
  od;
  return Rew;
fi;
for x in cw do
  h:=x[2];
  x[2]:=e;
  Rex:=HapHomotopy(n-1,Phi(n-1,
    BarBoundary(n,[x])));
  for u in Rex do
    u[3]:=SearchPosition(h*Elts[u[3]]);
    AddElement(Rew,u);
  od;
od;
return Rew;
end;
##
##### end of Phi #####

#####
#1
#F Equiv
## Input: A word w =[[m1,h1,g11,...,g1n],...,[mk,hk,gk1,...,gkn]]
## Output: The image of w under the homotopy map H_n: B_n->B_{n+1}
##
Equiv:=function(n,w)
local
  cw,h,x,
  PsiPhix,HBx,HLx,
  tmp,Rex,Rew,u;

cw:=StructuralCopy(w);
if n = 0 then
  return [];
fi;
Rew:=[];
for x in cw do
  h:=x[2];
  x[2]:=e;
  HBx:=Equiv(n-1,BarBoundary(n,[x]));
  AddElement(HBx,x);
  for tmp in HBx do
    tmp[1]:=-tmp[1];
  od;
  PsiPhix:= Psi(n,Phi(n,[x]));
  HLx:= Concatenation(PsiPhix,HBx);
  Rex:=BarHomotopy(n,HLx);
  for u in Rex do
    u[2]:=h*u[2];
    AddElement(Rew,u);
  od;
od;
return Rew;
end;
##
##### end of Equiv #####

```

```

    return rec(
        phi:=Phi,
        psi:=Psi,
        equiv:=Equiv
    );
end);
##
##### end of BarResolutionEquivalence #####

```

### 8.3.2 BarComplexEquivalence(R)

```

#####
#0
#F BarComplexEquivalence
## Input:  A HAP resolution of group G
## Output: A Z-equivariant chain homotopy equivalence between the bar
##         complex  $cB^G$  and the HAP complex  $cR^G$ 
##
InstallGlobalFunction(BarComplexEquivalence,function(R)
local
    e,dim,
    BarResEqui,Phi,Psi,Equiv,
    CPhi,CPsi,CEquiv;

    e:=Identity(R!.group);
    dim:=R!.dimension;
    BarResEqui:=BarResolutionEquivalence(R);
    Phi:=BarResEqui!.phi;
    Psi:=BarResEqui!.psi;
    Equiv:=BarResEqui!.equiv;

#####
#1
#F CPsi
## Input:  A word  $w = [m_1, e_1], \dots, [m_k, e_k]$  with  $k = \dim(R_n^G)$ 
## Output: The image of  $w$  under map  $cpsi: cR_n \rightarrow cB_n$ 
##
CPsi:=function(n,w)
local  Rew,x,cw;

    cw:=StructuralCopy(w);
    for x in cw do
        Add(x,1);
    od;
    Rew:=Psi(n,cw);
    for x in Rew do
        Remove(x,2);
    od;
    return Rew;
end;
##
##### end of CPsi #####

#####
#1
#F CPhi

```

```

## Input: A word w =[[m_1,g_11,...,g_1n],...[m_k,g_k1,...,g_kn]]
## Output: The image of w under map cphi: cB_n->cR_n
##
CPhi:=function(n,w)
local Zw,x,tmp,PhiZw,i,Rew;

  Zw:=[];
  for x in w do
    tmp=[x[1],e];
    for i in [2..n+1] do
      Add(tmp,x[i]);
    od;
    Add(Zw,tmp);
  od;
  PhiZw:=Phi(n,Zw);
  Rew:= List([ 1..dim(n)],x->0);
  for tmp in PhiZw do
    i:=tmp[2];
    Rew[i]:=Rew[i]+tmp[1];
  od;
return Rew;
end;
##
##### end of CPhi #####

#####

#1
#F CEquiv
## Input: A word w =[[m_1,g_11,...,g_1n],...,[m_k,g_k1,...,g_kn]]
## Output: The image of w under homotopy map cH_n: cB_n->cB_{n+1}
##
CEquiv:=function(n,w)
local Zw,x,i,tmp,Rew;

  Zw:=[];
  for x in w do
    tmp=[x[1],e];
    for i in [2..n+1] do
      Add(tmp,x[i]);
    od;
    Add(Zw,tmp);
  od;
  Rew:=Equiv(n,Zw);
  for tmp in Rew do
    Remove(tmp,2);
  od;
  return Rew;
end;
##
##### end of CEquiv #####

return rec(
  phi:=CPhi,
  psi:=CPsi,
  equiv:=CEquiv
);
end);
##
##### end of BarComplexEquivalence #####

```

### 8.3.3 ChainComplexOfSimplicialGroup(X)

```
#####
#0
#F ChainComplexOfSimplicialGroup
## Input:  A simplicial group, or a morphism of simplicial groups or
##         a sequence of morphisms of simplicial groups
## Output: The image of the input under the functor
##         (Simplicial groups)->(Chain complexes)
##
InstallGlobalFunction(ChainComplexOfSimplicialGroup, function(X)
local
  AddElement,
  ChainComplexOf_Objpre,
  ChainComplexOf_Obj,
  ChainComplexOf_Morpre,
  ChainComplexOf_Mor,
  ChainComplexOf_Seq;

#####
#1
#F AddElement
## Input:  A list L=[[m_1,h_1,g_11,...,g_1n],..., [m_k,h_k,g_k1,
##         ...,g_kn]] and an element x=[m',h',g'_1,...,g'_n]
## Output: Add the element x into the list L
##
AddElement:=function(L,x)
local  sx,nx,nL,flag,i,j;

  sx:=StructuralCopy(x);
  nx:=Length(sx);
  nL:=Length(L);
  for i in [1..nL] do
    flag:=0;
    for j in [2..nx] do
      if L[i][j]<>sx[j] then
        flag:=1;
        break;
      fi;
    od;
    if flag=0 then
      L[i][1]:=L[i][1]+sx[1];
      if L[i][1]=0 then
        Remove(L,i);
      fi;
      return;
    fi;
  od;
  Add(L,sx);
end;
##
##### end of AddElement #####

#####
#1
#F ChainComplexOf_Objpre
##
ChainComplexOf_Objpre:=function(N,Maps,Bar,Hap)
local
```

```

HapBoundary,Phi,Psi,Equiv,HapDimension,BarMap,
Dim,BoundChain,ImageBasis,
i,j,k,n,tmp,t,
M,m,ii,jj,
SearchPosition,Dimension,BelowDim,Boundary,
d0,dm;

#####
#2
HapBoundary:= function(i,j,k)
  return Hap[i+1]!.boundary(j,k);
end;
##
#####
#2
Psi:=function(i,j,w)
  return Bar[i+1]!.psi(j,w);
end;
##
#####
#2
Phi:=function(i,j,w)
  return Bar[i+1]!.phi(j,w);
end;#
##
#####
#2
Equiv:=function(i,j,w)
  return Bar[i+1]!.equiv(j,w);
end;
##
#####
#2
HapDimension:=function(i,j)
  return Hap[i+1]!.dimension(j);
end;
##
#####

#####
#2
#F BarMap
## Input: A position (i,j) and a word w
## Output: The image of w under del_n: B_{i}^j->B_{i-1}^j
##
BarMap:=function(i,j,w)
local Rew,sign,ii,jj,x,d,tmp;

  if j mod 2 = 0 then
    sign:=1;
  else
    sign:=-1;
  fi;
  Rew:=[];
  for ii in [0..i] do
    d:=Maps(i,ii);
    for x in w do
      tmp:=[sign*x[1]];
      for jj in [1..j] do

```

```

        Add(tmp,Image(d,x[jj+1]));
      od;
      AddElement(Rew,tmp);
    od;
    sign:=-sign;
  od;
  return Rew;
end;
##
##### end of BarMap #####

##### Compute the dimension of K_n #####
Dim:=[];
for i in [0..N] do
  k:=0;
  for j in [0..i] do;
    k:=k+HapDimension(j,i-j);
  od;
  Dim[i+1]:=k;
od;

#####
#2
Dimension:=function(n)
  return Dim[n+1];
end;
##
#####

#### Compute the sum of dimensions under the position(i,j)
BelowDim:=List([0..N],n->[]); ##i+1,j+1
for i in [0..N] do
  for j in [0..N-i] do
    n:=i+j;
    tmp:=0;
    for k in [0..j-1] do
      tmp:=tmp+HapDimension(n-k,k);
    od;
    BelowDim[i+1][j+1]:=tmp;
  od;
od;

#####
#2
#F SearchPosition
## Input: Two non-negative integers n and t
## Output: The position (i,j) and the position of e_t in R_ij
##
SearchPosition:=function(n,t)
local count,j,k;

  count:=0;
  for j in [0..n] do
    k:=t-count;
    count:=count+HapDimension(n-j,j);
    if t <= count then
      return [n-j,j,k];
      break;
    fi;
  od;
end;

```

```

        od;
    end;
    ##
    ##### end of SearchPosition #####

#####
#2
#F d0
## Input: A position (i,j) and a basis element e_k
## Output: The image of e_k under the map d_0
##
d0:=function(i,j,k)
local n,t,beg,Bound,Rew;

    n:=i+j;
    if n=0 then
        return [0];
    fi;
    Rew:=List([1..Dimension(n-1)],x->0);
    if j=0 then
        return Rew;
    fi;
    beg:=BelowDim[i+1][j];      ##below(i,j-1)
    Bound:=HapBoundary(i,j,k);
    for t in [1..HapDimension(i,j-1)] do
        Rew[beg+t]:=Bound[t];
    od;
return Rew;
end;
##
##### end of d0 #####

##### Compute d_m(e_k) at the postion (i,j) #####

ImageBasis:=[];
for i in [1..N] do
    ImageBasis[i]:=[];
    for j in [0..N-i]do
        ImageBasis[i][j+1]:=[];
        for m in [1..i] do
            ImageBasis[i][j+1][m]:=[];
        od;
    od;
od;

##### Compute ImageBasis[i][j+1][1][k] #####
for i in [1..N] do
    for j in [0..N-i] do
        for k in [1..HapDimension(i,j)] do
            ImageBasis[i][j+1][1][k]:=BarMap(i,j,Psi(i,j,[[1,k]]));
        od;
    od;
od;

##### Compute for m>1 #####
for i in [2..N]do
    for j in [0..N-i]do
        for m in [2..i] do
            for k in [1..HapDimension(i,j)] do

```

```

        tmp:=StructuralCopy(ImageBasis[i][j+1][m-1][k]);
        ImageBasis[i][j+1][m][k]:=BarMap(i-m+1,j+m-1,
            Equiv(i-m+1,j+m-2,tmp));
    od;
  od;
od;

#####
#2
#F dm
## Input: A position (i,j) and a basis element e_k
## Output: The image of e_k under the map d_m
##
dm:=function(i,j,m,k)
local n,t,beg,Rew,Phiw;

  n:=i+j;
  if n=0 then
    return [0];
  fi;
  Rew:=List([1..Dimension(n-1)],x->0);
  if m>i then
    return Rew;
  fi;
  Phiw:= Phi(i-m,j+(m-1),ImageBasis[i][j+1][m][k]);
  beg:=BelowDim[(i-m)+1][j+(m-1)+1];
  for t in [1..HapDimension(i-m,j+(m-1))] do
    Rew[beg+t]:=Phiw[t];
  od;
  return Rew;
end;
##
##### end of dm #####

BoundChain:=List([0..N],x->[]);
BoundChain[1][1]:=[0];
for n in [1..N] do
  for t in [1..Dimension(n)] do
    M:=SearchPosition(n,t);
    i:=M[1];
    j:=M[2];
    k:=M[3];
    tmp:=d0(i,j,k);
    for m in [1..i] do
      tmp:=tmp+dm(i,j,m,k);
    od;
    BoundChain[n+1][t]:=tmp;
  od;
od;
#####
#2
Boundary:=function(n,k)
  return BoundChain[n+1][k];
end;
##
#####

return Objectify( HapChainComplex, rec(

```

```

        boundary:=Boundary,
        dimension:=Dimension,
        properties:= [ [ "length",N],
                       [ "type", "chainComplex" ],
                       [ "characteristic",0 ] ]
    ) );

end;
##
##### end of ChainComplexOf_Objpre #####

#####
#1
#F ChainComplexOf_Obj
## Input: A simplicial group G
## Output: The chain complex of G
##
ChainComplexOf_Obj:=function(G)
local N,Maps,Grps,Bar,Hap,Res;

    N:=EvaluateProperty(G,"length");
    Maps:=G!.boundariesList;
    Grps:=G!.groupsList;
    Res:=List([0..N],i->ResolutionGenericGroup(Grps(i),N-i));
    Bar:=List([0..N],i->BarComplexEquivalence(Res[i+1]));
    Hap:=List([0..N],i->TensorWithIntegers(Res[i+1]));
    return ChainComplexOf_Objpre(N,Maps,Bar,Hap);
end;
##
##### end of ChainComplexOf_Obj #####

#####
#1
#F ChainComplexOf_Morpre
##
ChainComplexOf_Morpre:=function(N,map,MapsH,BarH,HapH,MapsG,BarG,HapG)
local
    PsiH,PhiH,EquivH,HapDimensionH,DimensionH,BarMapH,DimH,
    SearchPosH,ZeroVectorH,ImageBasisH,
    PsiG,PhiG,EquivG,HapDimensionG,DimensionG,BarMapG,DimG,
    SearchPosG,ZeroVectorG,ImgG,Imgs,
    i,j,k,m,n,t,tmp,Tmp,itmp,w,LM,FF,LowDimG,Mapping,
    BarMapHG;

##### All functions for H #####
#####
#2
PsiH:=function(i,j,w)
    return BarH[i+1]!.psi(j,w);
end;
##
#####
#2
PhiH:=function(i,j,w)
    return BarH[i+1]!.phi(j,w);
end;
##
#####
#2
EquivH:=function(i,j,w)

```

```

        return BarH[i+1]!.equiv(j,w);
    end;
    ##
    #####
    #2
    HapDimensionH:=function(i,j)
        return HapH[i+1]!.dimension(j);
    end;
    ##
    #####

    #####
    #2
    #F BarMapH
    ## Input:  A position (i,j) and a word w
    ## Output: The image of w under del_n: BH_{i}^j->BH_{i-1}^j
    ##
    BarMapH:=function(i,j,w)
    local Rew,sign,ii,jj,x,d,tmp;

        if j mod 2 = 0 then
            sign:=1;
        else
            sign:=-1;
        fi;
        Rew:=[];
        for ii in [0..i] do
            d:=MapsH(i,ii);
            for x in w do
                tmp:=[sign*x[1]];
                for jj in [1..j] do
                    Add(tmp,Image(d,x[jj+1]));
                od;
                AddElement(Rew,tmp);
            od;
            sign:=-sign;
        od;
        return Rew;
    end;
    ##
    ##### end of BarMapH #####

    ##### Compute the dimention of KH_n #####
    DimH:=[];
    for i in [0..N] do
        k:=0;
        for j in [0..i] do;
            k:=k+HapDimensionH(j,i-j);
        od;
        DimH[i+1]:=k;
    od;
    #####
    #2
    DimensionH:=function(n)
        return DimH[n+1];
    end;
    ##
    #####

```

```

#####
#2
#F SearchPosH
## Input: Two non-negative integers n and t
## Output: The position (i,j) and the order of e_t
SearchPosH:=function(n,t)
local count,j,k;

    if t>DimensionH(n) then
        return fail;
    fi;
    count:=0;
    for j in [0..n] do
        k:=t-count;
        count:=count+HapDimensionH(n-j,j);
        if t <= count then
            return [n-j,j,k];
            break;
        fi;
    od;
end;
##
##### end of SearchPosition #####

##### All functions for G #####
#####
#2
PsiG:=function(i,j,w)
    return BarG[i+1]!.psi(j,w);
end;
##
#####
#2
PhiG:=function(i,j,w)
    return BarG[i+1]!.phi(j,w);
end;
##
#####
#2
EquivG:=function(i,j,w)
    return BarG[i+1]!.equiv(j,w);
end;
##
#####
#2
HapDimensionG:=function(i,j)
    return HapG[i+1]!.dimension(j);
end;
##
#####

#####
#2
#F BarMapG
## Input: A position (i,j) and a word w
## Output: The image of w under del_n: BG_{i}^j->BG_{i-1}^j
##
BarMapG:=function(i,j,w)
local Rew,sign,ii,jj,x,d,tmp;

```

```

    if j mod 2 = 0 then
      sign:=1;
    else
      sign:=-1;
    fi;
    Rew:=[];
    for ii in [0..i] do
      d:=MapsG(i,ii);
      for x in w do
        tmp:=[sign*x[1]];
        for jj in [1..j] do
          Add(tmp,Image(d,x[jj+1]));
        od;
        AddElement(Rew,tmp);
      od;
      sign:=-sign;
    od;
    return Rew;
end;
##
##### end of BarMapG #####

##### Compute the dimention of KG_n #####
DimG:=[];
for i in [0..N] do
  k:=0;
  for j in [0..i] do;
    k:=k+HapDimensionG(j,i-j);
  od;
  DimG[i+1]:=k;
od;
#####
#2
DimensionG:=function(n)
  return DimG[n+1];
end;
##
#####

#####
#2
#F SearchPosG
## Input: Two non-negative integers n and t
## Output: The position (i,j) and the position of e_t in R_ij
##
SearchPosG:=function(n,t)
local count,j,k;

  if t>DimensionG(n) then
    return fail;
  fi;
  count:=0;
  for j in [0..n] do
    k:=t-count;
    count:=count+HapDimensionG(n-j,j);
    if t <= count then
      return [n-j,j,k];
    break;
  od;
end;

```

```

        fi;
    od;
end;
##
##### end of SearchPosition #####

#####
#2
#F BarMapHG
## Input: A position (i,j) and a word w
## Output: The image of w under the map_i: BH->BG
##
BarMapHG:=function(i,j,w)
local x,f,jj,tmp,Rew;

    f:=map(i);
    Rew=[];
    for x in w do
        tmp:=x[1];
        for jj in [2..j+1] do
            Add(tmp,Image(f,x[jj]));
        od;
        AddElement(Rew,tmp);
    od;
    return Rew;
end;
##
##### end of BarMapHG #####

##### Compute d_m(e_k) at the postion (i,j) in BH ####
ImageBasisH:=[];
for i in [0..N] do
    ImageBasisH[i+1]:=[];
    for j in [0..N-i]do
        ImageBasisH[i+1][j+1]:=[];
        for m in [0..i] do
            ImageBasisH[i+1][j+1][m+1]:=[];
        od;
    od;
od;

##### Compute for m =1 #####
for i in [0..N] do
    for j in [0..N-i] do
        for m in [0..i] do
            for k in [1..HapDimensionH(i,j)] do
                ImageBasisH[i+1][j+1][m+1][k]:=PsiH(i,j,[[1,k]]);
            od;
        od;
    od;
od;

##### Compute for m >1 #####
for i in [1..N]do
    for j in [0..N-i]do
        for m in [1..i] do
            for k in [1..HapDimensionH(i,j)] do
                tmp:=StructuralCopy(ImageBasisH[i+1][j+1][m][k]);
                ImageBasisH[i+1][j+1][m+1][k]:=EquivH(i-m,j+(m-1),

```



```

                Iimgs[i+1][j+1][m+1][k]:=PhiG(i-m,j+m,itmp);
            od;
        od;
    od;
    od;

##### Compute the sum of dimensions under position(i,j)
LowDimG:=List([0..N],n->[]);
for i in [0..N] do
    for j in [0..N-i] do
        n:=i+j;
        tmp:=0;
        for k in [0..j-1] do
            tmp:=tmp+HapDimensionG(n-k,k);
        od;
        LowDimG[i+1][j+1]:=tmp;
    od;
od;
#####
FF:=List([0..N],n->[]);
for n in [0..N] do
    for t in [1..DimensionH(n)] do
        LM:=SearchPosH(n,t);
        i:=LM[1];
        j:=LM[2];
        k:=LM[3];
        Tmp:=List([1..LowDimG[i+1][j+1]],m->0);
        for m in [0..i] do
            Append(Tmp,Iimgs[i+1][j+1][m+1][k]);
        od;
        FF[n+1][t]:=Tmp;
    od;
od;

#####
#2
#F Mapping
##
Mapping:=function(v,n)
local Rew,len,k;

    Rew:=List([1..DimensionG(n)],x->0);
    len:=Length(v);
    for k in [1..len] do;
        if v[k] <> 0 then
            Rew:=Rew+v[k]*FF[n+1][k];
        fi;
    od;
    return Rew;
end;
##
##### end of Mapping #####

return Mapping;
end;
##
##### end of ChainComplexOf_Morpre #####
#####

```

```

#1
#F ChainComplexOf_Mor
## Input:  A simplicial map Sf:G->G'
## Output: The chain map of chain complexes of Sf
##
ChainComplexOf_Mor:=function(Sf)
local
    map,N,
    H,GrpsH,MapsH,RH,HapH,BarH,
    G,GrpsG,MapsG,RG,HapG,BarG;

    H:=Sf!.source;
    G:=Sf!.target;
    map:=Sf!.mapping;
    N:=EvaluateProperty(Sf,"length");

    GrpsH:=H!.groupsList;
    MapsH:=H!.boundariesList;
    RH:=List([0..N],i->ResolutionGenericGroup(GrpsH(i),(N+1)-i));
    HapH:=List([0..N],i->TensorWithIntegers(RH[i+1]));
    BarH:=List([0..N],i->BarComplexEquivalence(RH[i+1]));

    GrpsG:=G!.groupsList;
    MapsG:=G!.boundariesList;
    RG:=List([0..N],i->ResolutionGenericGroup(GrpsG(i),(N+1)-i));
    HapG:=List([0..N],i->TensorWithIntegers(RG[i+1]));
    BarG:=List([0..N],i->BarComplexEquivalence(RG[i+1]));

    return Objectify( HapChainMap, rec(
        source:=ChainComplexOf_Objpre(N,MapsH,BarH,HapH),
        target:=ChainComplexOf_Objpre(N,MapsG,BarG,HapG),
        mapping:=ChainComplexOf_Morpre(N,map,MapsH,BarH,
            HapH,MapsG,BarG,HapG),
        properties:= [ [ "type", "chainMap" ],
            [ "characteristic",0 ] ]
    ));
end;
##
##### end of ChainComplexOf_Mor #####

#####
#1
#F ChainComplexOf_Seq
## Input:  A squence of simplicial maps L
## Output: The sequence of chain maps of chain complexes of Sf
##
ChainComplexOf_Seq:=function(L)
local
    nL,k,
    LSG,G,N,Grps,Maps,R,Hap,Bar,KG,RewL,map;

    nL:=Length(L);
    LSG=[];
    for k in [1..nL] do;
        LSG[k]:=L[k]!.source;
    od;
    LSG[nL+1]:=L[nL]!.target;
    Hap=[];
    Bar=[];

```

```

Maps:=[];
KG:=[];
for k in [1..nL+1] do
  G:=LSG[k];
  N:=EvaluateProperty(G,"length");
  Grps:=G!.groupsList;
  Maps[k]:=G!.boundariesList;
  R:=List([0..N],i->ResolutionGenericGroup(Grps(i),(N+1)-i));
  Hap[k]:=List([0..N],i->TensorWithIntegers(R[i+1]));
  Bar[k]:=List([0..N],i->BarComplexEquivalence(R[i+1]));
  KG[k]:=ChainComplexOf_Objpre(N,Maps[k],Bar[k],Hap[k]);
od;

RewL:=[];
for k in [1..nL] do
  N:=EvaluateProperty(L[k],"length");
  map:=L[k]!.mapping;
  RewL[k]:=Objectify( HapChainMap, rec(
    source := KG[k],
    target := KG[k+1],
    mapping := ChainComplexOf_Morpre(N,map,Maps[k],
      Bar[k],Hap[k],Maps[k+1],Bar[k+1],Hap[k+1]),
    properties:= [ [ "type", "chainMap" ],
      [ "characteristic",0 ] ]
  ) );
od;
return RewL;
end;
##
##### end of ChainComplexOf_Seq #####

if IsHapSimplicialGroup(X) then
  return ChainComplexOf_Obj(X);
fi;

if IsHapSimplicialGroupMorphism(X) then
  return ChainComplexOf_Mor(X);
fi;

if IsList(X) then
  return ChainComplexOf_Seq(X);
fi;
end);
##
##### end of ChainComplexOfSimplicialGroup #####

```

### 8.3.4 EilenbergMacLaneSimplicialGroup(X,n,l)

```

#####
#0
#F EilenbergMacLaneSimplicialGroup
## Input:  An abelian group A or a homomorphism of abelian groups, and
##         two positive integers n>=2,l>=1.
## Output: The Eilenberg-MacLane simplicial group K(A,n) of length l or
##         the morphism f_*: K(A,n) -> K(B,n) of simplicial groups of
##         length l by applying the functor
##         K(-,n): (Abelian groups)->(Simplicial abelian groups)

```

```

##
InstallGlobalFunction(EilenbergMacLaneSimplicialGroup, function(X,n,l)
local
  EilenbergMacLane_Obj,
  EilenbergMacLane_Map;

#####
#1
#F EilenbergMacLane_Obj
## Input:  An abelian group A and two non-negative integers n, l
## Output: The simplicial group K(A,n) of length l
##
EilenbergMacLane_Obj:= function(A,NN,nK)
local
  nn,zero,i,j,n,k,pos,tmp,x,y,
  GensA,ZeroGrp,CoF,CoD,MN,
  ListGensG,T,N,G,H,GensG,nG,Pro,Emb,nNumG,nNumH,
  NumberFace,NumberDegen,ImgGens,
  ZeroToZero,AToZero,ZeroToA,IdA,
  Surjection,NumOfSur,
  Faces,Degens,ListGroups,
  GroupsList,FaciesList,DegeneraciesList,
  AllSurjections,CompositeOfMaps,CoDegeneracies,CoFaces;

  nn:=NN-1;

#####
#2
#F AllSurjections
## Input:  Two non-negative integers m, n and m>=n
## Output: All surjections from [m] to [n]
AllSurjections:=function(m,n)
local CreatCouple,Res,y,M;

  if m<n then
    return fail;
  fi;
  if m=0 then
    return [[[0,0]]];
  fi;
#####
#3
#F CreatCouple
##
CreatCouple:=function(m,n)
local LA,LB,M,x,i;

  if n=0 then
    M:=[];
    for i in [0..m] do
      Add(M,[i,0]);
    od;
    return [M];
  fi;
  if m=n-1 then
    M:=[];
    for i in [0..m] do
      Add(M,[i,i]);
    od;

```

```

        return [M];
    fi;
    LA:=CreatCouple(m-1,n-1);
    for x in LA do
        Add(x,[m,n-1]);
    od;

    LB:=CreatCouple(m-1,n);
    for x in LB do
        Add(x,[m,n]);
    od;

    return Concatenation(LA,LB);
end;
##
##### end of CreatCouple #####

Res:=CreatCouple(m-1,n);
for y in Res do
    Add(y,[m,n]);
od;
return Res;
end;
##### end of AllSurjections #####

#####
#2
#F CoFaces
## Input:  A integer n>=0
## Output: All cofaces: [n]->[n+1]
##
CoFaces:=function(n)
local i,k,M,Res;

    Res=[];
    for i in [0..n+1] do
        M=[];
        for k in [0..i-1] do
            Add(M,[k,k]);
        od;
        for k in [i..n] do
            Add(M,[k,k+1]);
        od;
        Add(Res,M);
    od;
    return Res;
end;
##
##### end of CoFaces #####

#####
#2
#F CoDegeneracies
## Input:  An integer n>=0
## Output: All codegeneracies: [n]-->[n-1]
##
CoDegeneracies:=function(n)
local i,k,M,Res;

```

```

    Res:=[];
    for i in [0..n-1] do
      M:=[];
      for k in [0..i-1] do
        Add(M,[k,k]);
      od;
      Add(M,[i,i]);
      for k in [i+1..n] do
        Add(M,[k,k-1]);
      od;
      Add(Res,M);
    od;
    return Res;
end;
##
##### end of CoDegeneracies #####

#####
#2
#F CompositeOfMaps
## Input:  Two maps m]->[n] and [n]->>[k]
## Output: The map [m]->>[k] if it exists or 0 for otherwise
##
CompositeOfMaps:=function(M,N)
local Res,k,m,Temp,i,x,y;
  k:=nn;
  m:=Length(M)-1;
  x:=M[m+1][2];
  y:=N[x+1][2];
  if y<>k then
    return 0;
  fi;
  Res:=[];
  Temp:=[];
  for i in [0..m] do
    x:=M[i+1][2];
    y:=N[x+1][2];
    Add(Res,[i,y]);
    Add(Temp,y);
  od;
  if Length(Set(Temp))<k+1 then
    return 0;
  fi;
  return Res;
end;
##
##### end of CompositeOfMaps #####

Surjection:=[];
NumOfSur:=[];
for i in [nn..nK] do
  Surjection[i+1]:=AllSurjections(i,nn);      ##[i+1]
  NumOfSur[i+1]:=Length(Surjection[i+1]);    ##[i+1]
od;

zero:=Identity(A);
ZeroGrp:=Group(zero);
ListGroups:=[];
for i in [0..nn-1] do

```

```

    ListGroups[i+1]:=ZeroGrp;
od;
ListGroups[nn+1]:=A;
for i in [nn+1..nK] do
    ListGroups[i+1]:=DirectProduct(List([1..NumOfSur[i+1]],x->A));
od;

GensA:=GeneratorsOfGroup(A);
ZeroToZero:=GroupHomomorphismByImages(ZeroGrp,ZeroGrp,[],[]);
AToZero:=GroupHomomorphismByImages(A,ZeroGrp,GensA,
    List(GensA,x->zero));
ZeroToA:=GroupHomomorphismByImages(ZeroGrp,A,[],[]);
IdA:=GroupHomomorphismByImages(A,A,GensA,GensA);

##### Compute the faces map:  $K_n \rightarrow K_{n-1}$  #####

Faces:=List([1..nK],i->[]);

##### Compute the face maps  $d_k^i$  with  $k < n$  #####
for i in [1..nn-1] do
    for j in [0..i] do
        Faces[i][j+1]:=ZeroToZero;
    od;
od;

##### Compute the face maps  $d_{nn}^i$  #####
if nn>0 then
    for j in [0..nn] do
        Faces[nn][j+1]:=AToZero;
    od;
fi;

##### Compute the face map  $d_n^i$  #####
NumberFace:=[];
for n in [1..nK] do
    NumberFace[n]:=[];
    for i in [0..n] do
        NumberFace[n][i+1]:=[];
    od;
od;

for n in [nn+1..nK] do
    CoF:=CoFaces(n-1);
    MN:=Surjection[n];
    for i in [0..n] do
        for k in [1..NumOfSur[n+1]] do
            N:=Surjection[n+1][k];
            T:=CompositeOfMaps(CoF[i+1],N);
            if T=0 then
                NumberFace[n][i+1][k]:=0;
            else
                NumberFace[n][i+1][k]:=Position(MN,T);
            fi;
        od;
    od;
od;

for n in [nn+1..nK] do
    G:=ListGroups[n+1];

```

```

H:=ListGroupsWith[n];
nNumG:=NumOfSur[n+1];
nNumH:=NumOfSur[n];
GensG:=GeneratorsOfGroup(G);
nG:=Length(GensG);
Pro:=List([1..nNumG],k->Projection(G,k));
ListGensG:=[];
for i in [1..nG] do
  x:=GensG[i];
  tmp:=List([1..nNumG],k->Image(Pro[k],x));
  Add(ListGensG,tmp);
od;

if n=nn+1 then      ## at position nn, there is only A
  Emb:=[IdA];
else
  Emb:=List([1..nNumH],k->Embedding(H,k));
fi;

for i in [0..n] do
  ImgGens:=[];
  for j in [1..nG] do
    x:=Identity(H);
    for k in [1..nNumG] do
      pos:=NumberFace[n][i+1][k];
      if pos<>0 then
        x:=x*Image(Emb[pos],ListGensG[j][k]);
      fi;
    od;
    ImgGens[j]:=x;
  od;
  Faces[n][i+1]:=GroupHomomorphismByImages(G,H,GensG,ImgGens);
od;
od;

##### Compute the degeneracy map  $s_n^i$  #####
Degens:=List([0..nK-1],i->[]);

##### Compute the degeneracy maps  $s_k^i$  with  $k<n-1$  #####
for i in [0..nn-2] do
  for j in [0..i] do
    Degens[i+1][j+1]:=ZeroToZero;
  od;
od;

##### Compute the degeneracy maps at  $s_{n-1}^i$  #####
if nn>1 then
  for j in [0..nn-1] do
    Degens[nn][j+1]:=ZeroToA;
  od;
fi;

NumberDegen:=[]; ##### [n+1][i+1][k]
for n in [0..nK-1] do
  NumberDegen[n+1]:=[];
  for i in [0..n] do
    NumberDegen[n+1][i+1]:=[];
  od;
od;
od;

```

```

for n in [nn..nK-1] do
  CoD:=CoDegeneracies(n+1);
  MN:=Surjection[n+2];
  for i in [0..n] do
    for k in [1..NumOfSur[n+1]] do
      N:=Surjection[n+1][k];
      T:=CompositeOfMaps(CoD[i+1],N);
      if T=0 then
        NumberDegen[n+1][i+1][k]:=0;
      else
        NumberDegen[n+1][i+1][k]:=Position(MN,T); ##i+1
      fi;
    od;
  od;
od;
#####
for n in [nn..nK-1] do
  G:=ListGroups[n+1];
  H:=ListGroups[n+2];
  nNumG:=NumOfSur[n+1];
  nNumH:=NumOfSur[n+2];
  GensG:=GeneratorsOfGroup(G);
  nG:=Length(GensG);

  if n=nn then
    Pro:=[IdA];
  else
    Pro:=List([1..nNumG],k->Projection(G,k));
  fi;
  ListGensG:=[];
  for i in [1..nG] do
    x:=GensG[i];
    tmp:=List([1..nNumG],k->Image(Pro[k],x));
    Add(ListGensG,tmp);
  od;
  Emb:=List([1..nNumH],k->Embedding(H,k));

  for i in [0..n] do
    ImgGens:=[];
    for j in [1..nG] do
      x:=Identity(H);
      for k in [1..nNumG] do
        pos:=NumberDegen[n+1][i+1][k];
        if pos<>0 then
          x:=x*Image(Emb[pos],ListGensG[j][k]);
        fi;
      od;
      ImgGens[j]:=x;
    od;
    Degen[n+1][i+1]:=GroupHomomorphismByImages(G,H,
      GensG,ImgGens);
  od;
od;
#####
GroupsList:=function(i)
  return ListGroups[i+1];

```

```

end;
#####
FaciesList:=function(i,j)
    return Faces[i][j+1];
end;
#####
DegeneraciesList:=function(i,j)
    return Degens[i+1][j+1];
end;
#####
return Objectify(HapSimplicialGroup,
    rec(
        groupsList:=GroupsList,
        boundariesList:=FaciesList,
        degeneraciesList:=DegeneraciesList,
        properties:=[["length",nK]]
    ));
end;
##
##### end of EilenbergMacLane_Obj #####

#####
#1
#F EilenbergMacLane_Map
## Input: A homomorphism of abelian groups f:A->B and n, nK
## Output: The morphism K(f,n):K(A,n)->K(B,n) of simplicial groups
##         of length nK
##
EilenbergMacLane_Map:=function(f,n,nK)
local
    A,B,KA,KB,
    Maps,Mapping,GrpKA,GrpKB,
    Gens,Pro,Emb,ImgGens,
    i,j,k,t,nGens,
    h,g;

A:=f!.Source;
B:=f!.Range;
KA:=EilenbergMacLane_Obj(A,n,nK);
KB:=EilenbergMacLane_Obj(B,n,nK);
Maps:=[];
for i in [0..n-2] do
    Maps[i+1]:=GroupHomomorphismByImages(Group(Identity(A)),
        Group(Identity(B)), [], []);
od;
Maps[n]:=f;      ##n-1
for i in [n..nK] do
    GrpKA:=KA!.groupsList(i);
    GrpKB:=KB!.groupsList(i);
    Gens:=GeneratorsOfGroup(GrpKA);
    Pro:=[];
    Emb:=[];
    k:=Length(GrpKA!.DirectProductInfo!.groups);
    for j in [1..k] do
        Pro[j]:=Projection(GrpKA,j);
        Emb[j]:=Embedding(GrpKB,j);
    od;
    ImgGens:=[];
    nGens:=Length(Gens);

```

```

    for j in [1..nGens] do
      h:=Gens[j];
      g:=Identity(GrpKB);
      for t in [1..k] do
        g:=g*Image(Emb[t],Image(f,Image(Pro[t],h)));
      od;
      ImgGens[j]:=g;
    od;
    Maps[i+1]:=GroupHomomorphismByImages(GrpKA,GrpKB,Gens,ImgGens);
  od;
  #####
  Mapping:=function(i)
    return Maps[i+1];
  end;
  #####
  return Objectify(HapSimplicialGroupMorphism,
    rec(
      source:=KA,
      target:=KB,
      mapping:=Mapping,
      properties:=["length",nK]
    ));
end;
##
##### end of EilenbergMacLane_Map #####

if IsGroup(X) then
  return EilenbergMacLane_Obj(X,n,l);
fi;

if IsGroupHomomorphism(X) then
  return EilenbergMacLane_Map(X,n,l);
fi;
end);
##
##### end of EilenbergMacLaneSimplicialGroup #####

```

### 8.3.5 CrossedModuleByAutomorphismGroup(G)

```

#####
#0
#F CrossedModuleByAutomorphismGroup
## Input: A group G
## Output: The crossed module d:G->Aut(G)
##
InstallGlobalFunction(CrossedModuleByAutomorphismGroup, function(G)
local AutG,GensG,d,act;

  AutG:=AutomorphismGroup(G);
  GensG:=GeneratorsOfGroup(G);
  d:=GroupHomomorphismByImages(G,AutG,GensG,List(GensG,g->
    GroupHomomorphismByImages(G,G,GensG,List(GensG,x->g^(-1)*x*g))););
  act:=function(f,g)
    return Image(f,g);
  end;
  return Objectify(HapCrossedModule,rec(
    map:=d,

```

```

                action:=act
            ));
end);
##
##### end of CrossedModuleByAutomorphismGroup #####

```

### 8.3.6 CrossedModuleByNormalSubgroup(G,N)

```

#####
#0
#F CrossedModuleByNormalSubgroup
## Input: A group G with normal subgroup N
## Output: The inclusion crossed module i:N->G
##
InstallGlobalFunction(CrossedModuleByNormalSubgroup, function(G,N)
local d,act;

    if not IsNormal(G,N) then
        Print("Only apply for a normal subgroup of group");
        return fail;
    fi;
    d:=GroupHomomorphismByFunction(N,G,x->x);
    act:=function(g,h)
        return g^(-1)*h*g;
    end;
    return Objectify(HapCrossedModule,rec(
        map:=d,
        action:=act
    ));
end);
##
##### end of CrossedModuleByNormalSubgroup #####

```

### 8.3.7 Order(X)

```

#####
#0
#0 Order
## Input: A crossed module X
## Output: The order of X.
##
InstallOtherMethod(Order, "Order of crossed modules",
[IsHapCrossedModule], function(X)
    return Order(Source(X!.map))*Order(Range(X!.map));
end);
##
##### end of Order #####

```

### 8.3.8 HomotopyGroup(X,n)

```

#####
#0
#0 HomotopyGroup
## Input: A crossed module X and n=1,2

```

```

## Output: The nth homotopy groups of X
##
InstallOtherMethod(HomotopyGroup, "Homology group of crossed modules",
[IsHapCrossedModule, IsInt], function(X,n)
local d;

    d:=X!.map;
    if n = 1 then
        return Range(d)/Image(d);
    fi;
    if n =2 then
        return Kernel(d);
    fi;
    Print("Only apply for n=1,2");
    return fail;
end);
##
##### end of HomotopyGroup #####

```

### 8.3.9 CatOneGroupByCrossedModule(X)

```

#####
#0
#F CatOneGroupByCrossedModule
## Input: A crossed module, or a morphism of crossed modules, or
## a sequence of morphisms of crossed modules
## Output: The image of the input under the functor
## (Crossed modules)->(Cat-1-groups)
##
InstallGlobalFunction(CatOneGroupByCrossedModule, function(X)
local
    CMTtoCat1_Obj,
    CMTtoCat1_Morpre,
    CMTtoCat1_Mor,
    CMTtoCat1_Seq;

#####
#1
#F CMTtoCat1_Obj
## Input: A crossed module X
## Output: The cat-1-group corresponds to X
##
CMTtoCat1_Obj:=function(XC)
local
    d,act,p,m,M,P,AutM,GensM,GensP,alpha,
    G,GensG,pro,emb1,emb2,Elts,Eltt,g,pg,s,t;

    d:=XC!.map;
    act:=XC!.action;
    M:=Source(d);
    P:=Range(d);

    AutM:=AutomorphismGroup(M);
    GensM:=GeneratorsOfGroup(M);
    GensP:=GeneratorsOfGroup(P);
    alpha:=GroupHomomorphismByImages(P,AutM,GensP,List(GensP,p->
        GroupHomomorphismByImages(M,M,GensM,List(GensM,m->act(p,m)))));

```

```

G:=SemidirectProduct(P,alpha,M);
GensG:=GeneratorsOfGroup(G);
pro:=Projection(G);
emb1:=Embedding(G,1);
emb2:=Embedding(G,2);
Elts:=[];
Eltt:=[];
for g in GensG do
  p:=Image(pro,g);
  pg:=Image(emb1,p);
  Add(Elts,pg);
  m:=PreImagesRepresentative(emb2,pg^(-1)*g);
  Add(Eltt,Image(emb1,p*Image(d,m)));
od;
s:=GroupHomomorphismByImages(G,G,GensG,Elts);
t:=GroupHomomorphismByImages(G,G,GensG,Eltt);
return Objectify(HapCatOneGroup,
  rec(sourceMap:=s,
    targetMap:=t
  ));
end;
##
##### end of CMTtoCat1_Obj #####

#####
#1
#F CMTtoCat1_Morpre
##
CMTtoCat1_Morpre:=function(CC,CD,map)
local
  GC,GensGC,proC,emb1C,emb2C,g,
  GD,fM,fP,p,m,emb1D,emb2D,ImGensGC;

  GC:=Source(CC!.sourceMap);
  GensGC:=GeneratorsOfGroup(GC);
  GD:=Source(CD!.sourceMap);
  fM:=map(1);
  fP:=map(2);

  proC:=Projection(GC);
  emb1C:=Embedding(GC,1);
  emb2C:=Embedding(GC,2);
  emb1D:=Embedding(GD,1);
  emb2D:=Embedding(GD,2);
  ImGensGC:=[];
  for g in GensGC do
    p:=Image(proC,g);
    m:=PreImagesRepresentative(emb2C,(Image(emb1C,p))^(-1)*g);
    Add(ImGensGC,Image(emb1D,Image(fP,p))*
      Image(emb2D,Image(fM,m)));
  od;
  return Objectify(HapCatOneGroupMorphism,
    rec(
source:= CC,
target:= CD,
mapping:= GroupHomomorphismByImages(GC,
  GD,GensGC,ImGensGC)
));
end;

```

```

##
##### end of CMTtoCat1_Morpre #####

#####
#1
#F CMTtoCat1_Mor
## Input: A morphism fX of crossed modules
## Output: The morphism of cat-1-groups corresponds to fX
##
CMTtoCat1_Mor:=function(fX)
local CC,CD,map;

    CC:=CMTtoCat1_Obj(fX!.source);
    CD:=CMTtoCat1_Obj(fX!.target);
    map:=fX!.mapping;
    return CMTtoCat1_Morpre(CC,CD,map);
end;
##
##### end of CMTtoCat1_Mor #####

#####
#1
#F CMTtoCat1_Seq
## Input: A sequence LfX of morphisms of crossed modules
## Output: The sequence of morphisms of cat-1s corresponds to LfX
##
CMTtoCat1_Seq:=function(LfX)
local n,i,GC,Res;

    n:=Length(LfX);
    GC:=[];
    for i in [1..n] do
        GC[i]:=CMTtoCat1_Obj(LfX[i]!.source);
    od;
    GC[n+1]:=CMTtoCat1_Obj(LfX[i]!.target);
    Res:=[];
    for i in [1..n] do
        Res[i]:=CMTtoCat1_Morpre(GC[i],GC[i+1],LfX[i]!.mapping);
    od;
    return Res;
end;
##
##### end of CMTtoCat1_Seq #####

if IsHapCrossedModule(X) then
    return CMTtoCat1_Obj(X);
fi;
if IsHapCrossedModuleMorphism(X) then
    return CMTtoCat1_Mor(X);
fi;
if IsList(X) then
    return CMTtoCat1_Seq(X);
fi;
end);
##
##### end of CatOneGroupByCrossedModule #####

```

### 8.3.10 CrossedModuleByCatOneGroup(X)

```
#####
#0
#F CrossedModuleByCatOneGroup
## Input:  A cat-1-group, or a morphism of cat-1-groups, or
##         a sequence of morphisms of cat-1-groups
## Output: The image of the input under the functor
##         (Cat-1-groups)->(Crossed modules)
##
InstallGlobalFunction(CrossedModuleByCatOneGroup, function(X)
local
    Cat1ToCM_Obj,
    Cat1ToCM_Morpre,
    Cat1ToCM_Mor,
    Cat1ToCM_Seq;

#####
#1
#F Cat1ToCM_Obj
## Input:  A cat-1-group C
## Output: The crossed module corresponds to C
##
Cat1ToCM_Obj:=function(C)
local  s, t,M,P,GensM,d,act;

    s:=C!.sourceMap;
    t:=C!.targetMap;
    M:=Kernel(s);
    P:=Image(s);
    GensM:=GeneratorsOfGroup(M);
    d:=GroupHomomorphismByImages(M,P,GensM,List(GensM,m->Image(t,m)));
    act:=function(p,m)
        return p^(-1)*m*p;
    end;
    return Objectify(HapCrossedModule,
                    rec(map:=d,
                       action:=act)
                    );
end;
##
##### end of Cat1ToCM_Obj #####

#####
#1
#F Cat1ToCM_Morpre
##
Cat1ToCM_Morpre:=function(XC,XD,f)
local
    phiC,MC,PC,
    GensM,GensP,mapM,mapP,Map,
    phiD,MD,PD;

    phiC:=XC!.map;
    phiD:=XD!.map;
    MC:=Source(phiC);
    PC:=Range(phiC);
    MD:=Source(phiD);
    PD:=Range(phiD);
```

```

GensM:=GeneratorsOfGroup(MC);
GensP:=GeneratorsOfGroup(PC);
mapM:=GroupHomomorphismByImages(MC,MD,GensM,List(GensM,
  m->Image(f,m)));
mapP:=GroupHomomorphismByImages(PC,PD,GensP,List(GensP,
  p->Image(f,p)));

#####
#2
Map:=function(n)
  if n=1 then
    return mapM;
  fi;
  if n=2 then
    return mapP;
  fi;
  Print("Only apply for n =1,2");
  return fail;
end;
##
#####

return Objectify(HapCrossedModuleMorphism,
  rec(source:=XC,
    target:=XD,
    mapping:=Map
  ));

end;
##
##### end of Cat1ToCM_Morpre #####

#####
#1
#F Cat1ToCM_Mor
## Input: A morphism fC of cat-1-groups
## Output: The morphism of crossed modules corresponds to fC
##
Cat1ToCM_Mor:=function(fC)
local XC,XD,f;

  XC:=Cat1ToCM_Obj(fC!.source);
  XD:=Cat1ToCM_Obj(fC!.target);
  f:=fC!.mapping;
  return Cat1ToCM_Morpre(XC,XD,f);
end;
##
##### end of Cat1ToCM_Mor #####

#####
#1
#F Cat1ToCM_Seq
## Input: A sequence LfC of morphisms of cat-1-groups
## Output: The sequence of morphisms of crossed modules
## corresponds to LfC
##
Cat1ToCM_Seq:=function(LfC)
local n,i,XC,Res;

  n:=Length(LfC);

```

```

XC:=[];
for i in [1..n] do
  XC[i]:=Cat1ToCM_Obj(LfC[i]!.source);
od;
XC[n+1]:=Cat1ToCM_Obj(LfC[i]!.target);
Res:=[];
for i in [1..n] do
  Res[i]:=Cat1ToCM_Morpre(XC[i],XC[i+1],LfC[i]!.mapping);
od;
return Res;
end;
##
##### end of Cat1ToCM_Seq #####

if IsHapCatOneGroup(X) then
  return Cat1ToCM_Obj(X);
fi;
if IsHapCatOneGroupMorphism(X) then
  return Cat1ToCM_Mor(X);
fi;
if IsList(X) then
  return Cat1ToCM_Seq(X);
fi;
end);
##
##### end of CrossedModuleByCatOneGroup #####

```

### 8.3.11 NerveOfCatOneGroup(X,n)

```

#####
#0
#F NerveOfCatOneGroup
## Input: A cat-1-group, or a morphism of cat-1-group or a sequence of
## morphisms of cat-1-groups
## Output: The image of the input under the functor
## Nerve: (cat-1-groups)->(simplicial groups)
##
InstallGlobalFunction(NerveOfCatOneGroup, function(X,n)
local
  NerveOfCatOneGroup_Morpre,
  C,NerveOfCatOneGroup_Obj,
  NerveOfCatOneGroup_Mor,
  NerveOfCatOneGroup_Seq;

#####
#1
#F NerveOfCatOneGroup_Obj
## Input: A cat-1-group C and an integer number >=0
## Output: The nerve of C
##
NerveOfCatOneGroup_Obj:=function(C,number)
local
  LGs,LBs, LDs,
  s,t,e,
  N,M,AutM,phi,
  g,pg,m,ConjTmp,Tmp,TmpBs,TmpDs,
  Gens,GensToLists,ImgGens,

```

```

LTmpB,LTmpD,
i,j,k,n,len,
EmbOnes,EmbTwos,Pros,
ListToOne,BoundariesOfToList,DegeneraciesOfToList,
GroupsList,BoundariesList,DegeneraciesList;

if not IsHapCatOneGroup(C) then
  Print("This function must be applied to a cat-1-group.\n");
  return fail;
fi;
s:=C!.sourceMap;
t:=C!.targetMap;
N:=Image(s);
M:=Kernel(s);
AutM:=AutomorphismGroup(M);
e:=One(M);
LGs:=[];
LBs:=[];
LDs:=[];
EmbOnes:=[];
EmbTwos:=[];
Pros:=[];
Gens:=[];
GensToLists:=[];

##### Compute the list of group G_i for i=1..n #####
LGs[1]:=Source(s);
Gens[1]:=GeneratorsOfGroup(LGs[1]);
GensToLists[1]:=List(Gens[1],g->[g]);
for n in [2..number] do
  ConjTmp:=[];
  len:=Length(Gens[n-1]);
  for i in [1..len] do
    m:=GensToLists[n-1][i][1];
    for j in [2..n-1] do
      m:=m*GensToLists[n-1][i][j];
    od;
    Add(ConjTmp,ConjugatorAutomorphismNC(M,Image(t,m)));
  od;
  phi:=GroupHomomorphismByImagesNC(LGs[n-1],AutM,
    Gens[n-1],ConjTmp);
  LGs[n]:=SemidirectProduct(LGs[n-1],phi,M);
  EmbOnes[n]:=Embedding(LGs[n],1);
  EmbTwos[n]:=Embedding(LGs[n],2);
  Pros[n]:=Projection(LGs[n]);
  Gens[n]:=GeneratorsOfGroup(LGs[n]);
  len:=Length(Gens[n]);
  GensToLists[n]:=List([1..len],x->[]);
  for i in [1..len] do
    g:=Gens[n][i];
    Tmp:=[];
    for j in [1..n-1] do
      pg:=Image(Pros[n-j+1],g);
      m:=PreImagesRepresentative(EmbTwos[n-j+1],
        (Image(EmbOnes[n-j+1],pg))(-1)*g);
      Tmp[n-j+1]:=m;
      g:=pg;
    od;
    Tmp[1]:=g;
  od;

```

```

        GensToLists[n][i]:=Tmp;
    od;
od;

#####
#2
#F BoundariesOfToList
## Input: List Lm:=[g_1,m_2,...,m_n]
## Output: List of the image of d_i(Lm) with i:=0..n
##
BoundariesOfToList:=function(Lm,n)
local i,j,TmpB,LB;

    if n=2 then
        LB:=[[Image(t,Lm[1])*Lm[2]],[Lm[1]*Lm[2]],[Lm[1]]];
    fi;

    if n>2 then
        LB:=[];

        ##### Compute d_0 #####
        TmpB:=[Image(t,Lm[1])*Lm[2]];
        for i in [2..n-1] do
            TmpB[i]:=Lm[i+1];
        od;
        Add(LB,TmpB);

        ##### Compute d_1-->d_{n-1} #####
        for i in [2..n] do
            TmpB:=[];
            for j in [1..i-2] do
                TmpB[j]:=Lm[j];
            od;
            TmpB[i-1]:=Lm[i-1]*Lm[i];
            for j in [i..n-1] do
                TmpB[j]:=Lm[j+1];
            od;
            Add(LB,TmpB);
        od;

        ##### Compute d_n #####
        TmpB:=[];
        for i in [1..n-1] do
            TmpB[i]:=Lm[i];
        od;
        Add(LB,TmpB);
    fi;
    return LB;
end;
##
##### end of BoundariesOfToList #####

#####
#2
#F DegeneraciesOfToList
## Input: List Lm:=[g_1,m_2,...,m_n]
## Output: List of the image of s_i(Lm) with i:=0..n
##
DegeneraciesOfToList:=function(Lm,n)

```

```

local i,j,TmpD,LD,g;

g:=Lm[1];
if n=1 then
  LD:=[[Image(s,g),Image(s,g^(-1))*g],[g,e]];
fi;
if n>1 then
  LD:=[];

##### Compute s_0 #####
TmpD:= [Image(s,g),Image(s,g^(-1))*g];
for i in [3..n+1] do
  TmpD[i]:=Lm[i-1];
od;
Add(LD,TmpD);

##### Compute s_1 -> s_n #####
for i in [2..n+1] do
  TmpD:=[];
  for j in [1..i-1] do
    TmpD[j]:=Lm[j];
  od;
  TmpD[i]:=e;
  for j in [i+1..n+1] do
    TmpD[j]:=Lm[j-1];
  od;
  Add(LD,TmpD);
od;
fi;
return LD;
end;
##
##### end of DegeneraciesOfToList #####

#####
#2
#F ListToOne
## Input: List Lm:=[g_1,m_2,m_3,...,m_n]
## Output: The semi-product g_1 x| m_2 x| m_3 x| ... x| m_n
##
ListToOne:=function(Lm,n)
local i,m;

if n=1 then
  m:=Lm[1];
fi;
if n>1 then
  m:=Lm[1];
  for i in [2..n] do
    m:=Image(EmbOnes[i],m)*Image(EmbTwos[i],Lm[i]);
  od;
fi;
return m;
end;
##
##### end of ListToOne #####

##### Compute boundary maps #####
LBs:=[[t,s]];

```

```

for n in [2..number] do
  len:=Length(Gens[n]);
  Tmp:=[];
  TmpBs:=[];
  for i in [1..len] do
    Tmp[i]:=BoundariesOfToList(GensToLists[n][i],n);
    TmpBs[i]:=List(Tmp[i],Lm->ListToOne(Lm,n-1));
  od;
  ImgGens:=[];
  for k in [1..n+1] do
    ImgGens[k]:=List([1..len],i->TmpBs[i][k]);
  od;
  LTmpB:=[];
  for k in [1..n+1] do
    LTmpB[k]:=GroupHomomorphismByImagesNC(LGs[n],LGs[n-1],
      Gens[n],ImgGens[k]);
  od;
  LBs[n]:=LTmpB;
od;

##### Compute degeneracy maps #####
for n in [1..number-1] do
  len:=Length(Gens[n]);
  Tmp:=[];
  TmpDs:=[];
  for i in [1..len] do
    Tmp[i]:=DegeneraciesOfToList(GensToLists[n][i],n);
    TmpDs[i]:=List(Tmp[i],Lm->ListToOne(Lm,n+1));
  od;
  ImgGens:=[];
  for k in [1..n+1] do
    ImgGens[k]:=List([1..len],i->TmpDs[i][k]);
  od;
  LTmpD:=[];
  for k in [1..n+1] do
    LTmpD[k]:=GroupHomomorphismByImagesNC(LGs[n],LGs[n+1],
      Gens[n],ImgGens[k]);
  od;
  LDs[n]:=LTmpD;
od;

#####
#2
GroupsList:=function(n)
  if n=0 then
    return N;
  fi;
  return LGs[n];
end;
##
#####

#####
#2
BoundariesList:=function(n,k)
  return LBs[n][k+1];
end;
##
#####

```

```

#####
#2
DegeneraciesList:=function(n,k)
  if n=0 and k = 0 then
    return GroupHomomorphismByFunction(N,LGs[1],
      function(x) return x; end);
  fi;
  return LDs[n][k+1];
end;
##
#####

return Objectify(HapSimplicialGroup,
  rec(
    groupsList:=GroupsList,
    boundariesList:=BoundariesList,
    degeneraciesList:=DegeneraciesList,
    properties:=[["length",number]]
  ));
end;
##
##### end of NerveOfCatOneGroup_Obj #####

#####
#1
#F NerveOfCatOneGroup_Morpre
## Input: Nerve of G, nerve of H and map f:G-->H
## Output: The morphism between nerve of G and nerve of H
##
NerveOfCatOneGroup_Morpre:=function(NG,NH,f,number)
local
  GLs,GEmbOnes,GEmbTwos,GPros,HLs,HEmbOnes,HEmbTwos,HPros,
  Gens,GensToLists,
  i,j,n,m,g,pg,len,
  Tmp,ImgGens,Maps,
  HListToOne,Mapping;

  GLs:=[];
  GEmbOnes:=[];
  GEmbTwos:=[];
  GPros:=[];
  HLs:=[];
  HEmbOnes:=[];
  HEmbTwos:=[];
  HPros:=[];
  Gens:=[];
  GensToLists:=[];
  for n in [2..number] do
    GLs[n]:=NG!.groupsList(n);
    GEmbOnes[n]:=Embedding(GLs[n],1);
    GEmbTwos[n]:=Embedding(GLs[n],2);
    GPros[n]:=Projection(GLs[n]);
    HLs[n]:=NH!.groupsList(n);
    HEmbOnes[n]:=Embedding(HLs[n],1);
    HEmbTwos[n]:=Embedding(HLs[n],2);
    HPros[n]:=Projection(HLs[n]);
    Gens[n]:=GeneratorsOfGroup(GLs[n]);
    len:=Length(Gens[n]);

```

```

GensToLists[n]:=List([1..len],x->[]);
for i in [1..len] do
  g:=Gens[n][i];
  Tmp:=[];
  for j in [1..n-1] do
    pg:=Image(GPros[n-j+1],g);
    m:=PreImagesRepresentative(GEmbTwos[n-j+1],(Image(
      GEmbOnes[n-j+1],pg))^(n-j)*g);
    Tmp[n-j+1]:=m;
    g:=pg;
  od;
  Tmp[1]:=g;
  GensToLists[n][i]:=Tmp;
od;
od;

#####
#2
#F HListToOne
## Input: List [h_1,m_2,m_3,...,m_n]
## Output: The semi-product h_1 x| m_2 x| m_3 x| ... x| m_n
##
HListToOne:=function(Lm,n)
local i,m;

  m:=Lm[1];
  for i in [2..n] do
    m:=Image(HEmbOnes[i],m)*Image(HEmbTwos[i],Lm[i]);
  od;
  return m;
end;
##
##### end of HListToOne #####

Maps:=[];
for n in [2..number] do
  len:=Length(Gens[n]);
  ImgGens:=[];
  for i in [1..len] do
    ImgGens[i]:=HListToOne(List(GensToLists[n][i],
  m->Image(f,m)),n);
  od;
  Maps[n]:=GroupHomomorphismByImages(GLs[n],HLs[n],
  Gens[n],ImgGens);
od;

#####
#2
Mapping:=function(n)
  if n=0 then
    return GroupHomomorphismByFunction(NG!.groupsList(0),
      NH!.groupsList(0),function(x) return Image(f,x); end);
  fi;
  if n=1 then
    return f;
  fi;
  return Maps[n];
end;
##

```

```

#####

return Objectify(HapSimplicialGroupMorphism,
  rec(
    source:=NG,
    target:=NH,
    mapping:=Mapping,
    properties:=["length",number])
  ));
end;
##
##### end of NerveOfCatOneGroup_Morpre #####

#####
#1
#F NerveOfCatOneGroup_Mor
## Input: A morphism of cat-1-groups
## Output: The morphism of their nerves
##
NerveOfCatOneGroup_Mor:=function(Cf,n)
local NG,NH,f;

  NG:=NerveOfCatOneGroup_Obj(Cf!.source,n);
  NH:=NerveOfCatOneGroup_Obj(Cf!.target,n);
  f:=Cf!.mapping;
  return NerveOfCatOneGroup_Morpre(NG,NH,f,n);
end;
##
##### end of NerveOfCatOneGroup_Mor #####

#####
#1
#F NerveOfCatOneGroup_Seq
## Input: A sequence of morphisms of cat-1-groups
## Output: The sequence of morphisms of their nerves
##
NerveOfCatOneGroup_Seq:=function(Lf,n)
local len,i,NC,Res;

  len:=Length(Lf);
  NC:=[];
  for i in [1..len] do
    NC[i]:=NerveOfCatOneGroup_Obj(Lf[i]!.source,n);
  od;
  NC[len+1]:=NerveOfCatOneGroup_Obj(Lf[len]!.target,n);
  Res:=[];
  for i in [1..len] do
    Res[i]:=NerveOfCatOneGroup_Morpre(NC[i],NC[i+1],
    Lf[i]!.mapping,n);
  od;
  return Res;
end;
##
##### end of NerveOfCatOneGroup_Seq #####

if IsHapCatOneGroup(X) then
  return NerveOfCatOneGroup_Obj(X,n);
fi;

```

```

if IsHapCatOneGroupMorphism(X) then
  return NerveOfCatOneGroup_Mor(X,n);
fi;
if IsList(X) then
  if IsEmpty(X) then
    return [];
  fi;
  return NerveOfCatOneGroup_Seq(X,n);
fi;
end);
##
##### end of NerveOfCatOneGroup #####

```

### 8.3.12 CatOneGroupsByGroup(G)

```

#####
#0
#F CatOneGroupsByGroup
## Input: A finite group G
## Output: The list of all non-isomorphic cat-1-group structures on G
##
InstallGlobalFunction(CatOneGroupsByGroup, function(G)
local
  nk,n,k,Lst,S,p,x,tmp,i,Imgs,s,h,hinv,Gens,C,ResCats,
  ClassifyPairsByOrbit,CreatePairsByAbelianGroup,
  CreatePairsByNonAbelianGroup,CreatePairsByGroup;

#####
#1
#F ClassifyPairsByOrbit
## Input: The automorphism group of group G and
## a list Lst of pairs of group homomorphisms [s,t]:G->G
## Output: A list of non-isomorphic pairs of Lst
##
ClassifyPairsByOrbit:=function(A,Lst)
local
  CL,TmpCL,Lx,Res,
  ActToMap,ActToPair,
  RefineClassesUnderGroup,
  processDuplicates;

if Length(Lst)<=1 then
  return Lst;
fi;

#####
#2
ActToMap:=function(s,f)
  return InverseGeneralMapping(f)*s*f;
end;
##
#####
#2
ActToPair:=function(p,f)
  local h;
  h:=InverseGeneralMapping(f);
  return [h*p[1]*f,h*p[2]*f];

```

```

end;
##
#####

#####
#2
#F RefineClassesUnderGroup
##
RefineClassesUnderGroup:=function(A,Lst,Indx,attr,actAttr)
local
  ValAttr,i,NC,LC,Sel,Orb,T,Dict,
  S,Gens,op,qs,g,h,img,p,x,cnt;

  ValAttr:=[];
  for i in [1..Length(Indx)] do
    ValAttr[i]:=attr(Lst[Indx[i]]);
  od;
  NC:=[];
  Gens:=SmallGeneratingSet(A);
  Sel:=[1..Length(Indx)];
  while Length(Sel)>0 do
    # orbit algorithm on attributes, regular transversal
    LC:=[Indx[Sel[1]]];
    Orb:=[ValAttr[Sel[1]]];
    Unbind(Sel[1]);
    T:=[One(A)];
    Dict:=NewDictionary(Orb[1],true);
    AddDictionary(Dict,Orb[1],1);
    S:=TrivialSubgroup(A);
    op:=1;
    qs:=Size(A);
    while op<=Length(Orb) and Size(S)<qs do
      for g in Gens do
        img:=actAttr(Orb[op],g);
        p:=LookupDictionary(Dict,img);
        if p=fail then
          Add(Orb,img);
          AddDictionary(Dict,img,Length(Orb));
          Add(T,T[op]*g);
          qs:=Size(A)/Length(Orb);
        elif Size(S)<=qs/2 then
          x:=T[op]*g/T[p];
          S:=ClosureSubgroup(S,x);
        fi;
      od;
      op:=op+1;
    od;

    # which other values are in the orbit
    for i in [2..Length(Sel)] do
      p:=LookupDictionary(Dict,ValAttr[Sel[i]]);
      if p<>fail then

        x:=Indx[Sel[i]];
        AddSet(LC,x);
        Unbind(Sel[i]);
        if p>1 then # not identity
          h:=T[p]^-1;
          Lst[x]:=ActToPair(Lst[x],h);

```

```

                fi;
            fi;
        od;
        Add(NC,[S,LC]);
        Sel:=Set(Sel);
    od;
    return NC;
end;
##
##### end of RefineClassesUnderGroup #####

#####
#2
#F processDuplicates #remove duplicates
##
processDuplicates:=function()
local Lx,i,p,Sel;

    TmpCL:=[];
    for Lx in CL do
        Sel:=[];
        for i in Lx[2] do
            p:=First(Sel,x->Lst[i]=Lst[x]);
            if p=fail then
                Add(Sel,i);
            fi;
        od;
        Add(TmpCL,[Lx[1],Sel]);
    od;
    CL:=TmpCL;
end;
##
##### end of processDuplicates #####

##### Images of first component #####
CL:=RefineClassesUnderGroup(A,Lst,[1..Length(Lst)],
x->Image(x[1]),function(s,a) return Image(a,s);end);
processDuplicates();

Res:=[];
##### Kernels of first component #####
TmpCL:=[];
for Lx in CL do
    if Length(Lx[2])= 1 then
        Add(Res,Lst[Lx[2][1]]);
    else
        Append(TmpCL,RefineClassesUnderGroup(Lx[1],Lst,Lx[2],
            x->Kernel(x[1]),function(s,a) return Image(a,s);end));
    fi;
od;
CL:=TmpCL;
processDuplicates();

##### Kernels of second component #####
TmpCL:=[];
for Lx in CL do
    if Length(Lx[2])= 1 then
        Add(Res,Lst[Lx[2][1]]);
    else

```

```

        Append(TmpCL,RefineClassesUnderGroup(Lx[1],Lst,Lx[2],
        x->Kernel(x[2]),function(s,a) return Image(a,s);end));
    fi;
od;
CL:=TmpCL;
processDuplicates();

##### First component #####
TmpCL:=[];
for Lx in CL do
    if Length(Lx[2])= 1 then
        Add(Res,Lst[Lx[2][1]]);
    else
        Append(TmpCL,RefineClassesUnderGroup(Lx[1],Lst,Lx[2],
        x->x[1],ActToMap));
    fi;
od;
CL:=TmpCL;
processDuplicates();

##### Second component #####
TmpCL:=[];
for Lx in CL do
    if Length(Lx[2])= 1 then
        Add(Res,Lst[Lx[2][1]]);
    else
        Append(TmpCL,RefineClassesUnderGroup(Lx[1],Lst,Lx[2],
        x->x[2],ActToMap));
    fi;
od;
CL:=TmpCL;
processDuplicates();

if IsEmpty(CL) then
    return Res;
fi;

if ForAny(CL,x->Length(x[2])>1) then
    Error("Uniqueness failure");
fi;
return Concatenation(Res,List(CL,x->Lst[x[2][1]]));
end;
##
##### end of ClassifyPairsByOrbit #####

#####
#1
#F CreatePairsByAbelianGroup
## Input: An abelian group G
## Output: A list of all non-isomorphic pairs [s,t]:G->G
## such that (G,s,t) is a cat-1-group
##
CreatePairsByAbelianGroup:=function(G)
local
    AbIn,NumX,SizeX,nX,GroupX,GensX,
    i,LSX,e,Gens,sum,
    GensLK,LK,sLK,LComX,tmp,GensK,Imgs,xK,xG,
    M,S,f,finv,nLK,Aut,n,LfK,K,CompK,N,GensN,t,
    ResPairs,FCombination;

```

```

AbIn:=AbelianInvariants(G);
if IsEmpty(AbIn) then
  return [IdentityMapping(G),IdentityMapping(G)];
fi;

NumX:=Set(AbIn);
SizeX:=List(NumX,x->Length(Filtered(AbIn,i->i=x)));

#####
#2
#F FCombination
##
FCombination:=function(n)
local T,ST,tmp,Res,x;

  if n=1 then
    return List([0..SizeX[n]],x->[x]);
  fi;
  if n>1 then
    Res:=[];
    T:=FCombination(n-1);
    for x in [0..SizeX[n]] do
      ST:=StructuralCopy(T);
      for tmp in ST do
        Add(tmp,x);
      od;
      Append(Res,ST);
    od;
    return Res;
  fi;
end;
##
#####

nX:=Length(NumX);
GroupX:=[];
GensX:=[];
for i in [1..nX] do
  GroupX[i]:=CyclicGroup(NumX[i]);
  GensX[i]:=First(GroupX[i],g->Order(g)=NumX[i]);
od;

LSX:=[];
for i in [1..nX] do
  Append(LSX,List([1..SizeX[i]],m->GroupX[i]));
od;
S:=DirectProduct(LSX);
e:=One(S);
Gens:=[];
sum:=0;
for i in [1..nX] do
  Append(Gens,List([1..SizeX[i]],m->Image(
    Embedding(S,sum+m),GensX[i])));
  sum:=sum+SizeX[i];
od;
GensLK:=[];
LK:=[];
sLK:=[];

```

```

LComX:=FCombination(Length(NumX));
for tmp in LComX do
  GensK:=[];
  Imgs:=[];
  sum:=0;
  for i in [1..Length(tmp)] do
    xK:=List([1..tmp[i]],m->Gens[sum+m]);
    xG:=Concatenation(xK,List([tmp[i]+1..SizeX[i]],m->e));
    Append(GensK,xK);
    Append(Imgs,xG);
    sum:=sum+SizeX[i];
  od;
  Add(GensLK,GensK);
  if IsEmpty(GensK) then
    Add(LK,Group(e));
  else
    Add(LK,Group(GensK));
  fi;
  Add(sLK,GroupHomomorphismByImages(S,S,Gens,Imgs));
od;

f:=IsomorphismGroups(S,G);
finv:=InverseGeneralMapping(f);
LK:=List(LK,K->Image(f,K));
sLK:=List(sLK,s->finv*s*f);
GensLK:=List(LK,K->GeneratorsOfGroup(K));
nLK:=Length(LK);

Aut:=AutomorphismGroup(G);
e:=One(G);
ResPairs:=[];
for n in [1..nLK] do
  LfK:=[];
  K:=LK[n];
  CompK:=ComplementClasses(G,K);
  for N in CompK do
    GensN:=SmallGeneratingSet(N);
    Gens:=Concatenation(GensLK[n],GensN);
    Imgs:=Concatenation(GensLK[n],List(GensN,g->e));
    t:=GroupHomomorphismByImages(G,G,Gens,Imgs);
    Add(LfK,[sLK[n],t]);
  od;
  Append(ResPairs,ClassifyPairsByOrbit(Aut,LfK));
od;
return ResPairs;
end;
##
##### end of CreatePairsByAbelianGroup #####

#####
#1
#F CreatePairsByNonAbelianGroup
## Input: An non-abelian group G
## Output: A list of all non-isomorphic pairs [s,t]:G->G such that
## (G,s,t) is a cat-1-group
##
CreatePairsByNonAbelianGroup:=function(G)
local

```

```

Aut:=AutomorphismGroup(G);
GensG:=GeneratorsOfGroup(G);
LN:=NormalSubgroups(G);
nLN:=Length(LN);
IdLN:=List(LN,x->IdGroup(x));
SizeLN:=List(LN,x->Size(x));
SetSizeLN:=Set(SizeLN);
CLN:=List([1..Length(SetSizeLN)],x->[]);
for i in [1..nLN] do
  Add(CLN[Position(SetSizeLN,SizeLN[i])],i);
od;
nCLN:=Length(CLN);
LS:=LatticeSubgroups(G)!.conjugacyClassesSubgroups;
if not IsMutable(LS) then
  LS:= ShallowCopy(LS);
fi;
LS:=List(LS,x->x[1]);
IdLS:=List(LS,x->IdGroup(x));
nLS:=Length(LS);
PairSKer:=[];
PairNotSKer:=[];
for n in [1..nCLN] do
  L:=CLN[n];
  nL:=Length(L);
  LfN:=List([1..nLN],x->[]);
  for i in L do
    N:=LN[i];
    nat:=NaturalHomomorphismByNormalSubgroup(G,N);
    GoN:=Range(nat);
    SizeGoN:=Size(GoN);
    IdGoN:=IdGroup(GoN);
    for k in [1..nLS] do
      if IdLS[k]= IdGoN then
        K:=LS[k];
        if Size(Image(nat,K))=SizeGoN then
          GensK:=GeneratorsOfGroup(K);
          f:=GroupHomomorphismByImages(K,GoN,GensK,
            List(GensK,g->Image(nat,g)));
          h:=InverseGeneralMapping(f);
          s:=GroupHomomorphismByImages(G,G,GensG,
            List(GensG,g->Image(h,Image(nat,g))));
          Add(LfN[i],[k,s]);
        fi;
      fi;
    od;
  od;
od;

SKer:=[];
for a in [1..nL] do
  i:=L[a];
  Li:=[];
  if Size(CommutatorSubgroup(LN[i],LN[i]))=1 then
    Li:=List(LfN[i],x->[x[2],x[2]]);
  fi;
od;

```

```

        fi;
        Append(SKer,ClassifyPairsByOrbit(Aut,Li));
    od;
    Append(PairSKer,ClassifyPairsByOrbit(Aut,SKer));

    NotSKer:=[];
    for a in [2..nL] do
        i:=L[a];
        Li:=[];
        for b in [1..a-1] do
            j:=L[b];
            if Size(CommutatorSubgroup(LN[i],LN[j]))=1 then
                for x in LfN[i] do
                    for y in LfN[j] do
                        if x[1]=y[1] then
                            Add(Li,[x[2],y[2]]);
                        fi;
                    od;od;
                fi;
            od;
            Append(NotSKer,ClassifyPairsByOrbit(Aut,Li));
        od;
        NotSKer:=ClassifyPairsByOrbit(Aut,NotSKer);
        T:=ShallowCopy(NotSKer);
        for x in NotSKer do
            Add(T,[x[2],x[1]]);
        od;
        Append(PairNotSKer,ClassifyPairsByOrbit(Aut,T));
    od;
    ResPairs:=Concatenation(PairSKer,PairNotSKer);
    return ResPairs;
end;
##
##### end CreatePairsByNonAbelianGroup #####

#####
#1
CreatePairsByGroup:=function(G)
    if IsAbelian(G) then
        return CreatePairsByAbelianGroup(G);
    else
        return CreatePairsByNonAbelianGroup(G);
    fi;
end;
##
##### CreatePairsByGroup #####

n:=Size(G);
if n<=HAP_CAT_SIZE then
    nk:=IdGroup(G);
    n:=nk[1];
    k:=nk[2];
    Gens:=GeneratorsOfGroup(G);
    S:=SmallGroup(n,k);
    h:=IsomorphismGroups(G,S);
    hinverse:=InverseGeneralMapping(h);
    Lst:=[];
    if nk in HAP_CAT_PERM then
        for x in HAP_CAT[n][k] do

```

```

        tmp:=[];
        for i in [1..2] do
            s:=GroupHomomorphismByImages(S,S,x[i][1],x[i][2]);
            Iimgs:=List(Gens,g->Image(hinv,Image(s,(Image(h,g)))))
            tmp[i]:=GroupHomomorphismByImages(G,G,Gens,Iimgs);
        od;
        Add(Lst,tmp);
    od;
else
    p:=Pcgs(S);
    for x in HAP_CAT[n][k] do
        tmp:=[];
        for i in [1..2] do
            Iimgs:=List(x[i],m->PcElementByExponents(p,m));
            s:=GroupHomomorphismByImages(S,S,p,Iimgs);
            Iimgs:=List(Gens,g->Image(hinv,Image(s,(Image(h,g)))))
            tmp[i]:=GroupHomomorphismByImages(G,G,Gens,Iimgs);
        od;
        Add(Lst,tmp);
    od;
fi;
else
    Lst:=CreatePairsByGroup(G);
fi;

ResCats:=[];
for x in Lst do
    C:=Objectify(HapCatOneGroup,
                rec(sourceMap:=x[1],
                   targetMap:=x[2]
                ));
    Add(ResCats,C);
od;
return ResCats;
end);
##
##### end of CatOneGroupsByGroup #####

```

### 8.3.13 NumberSmallCatOneGroups(arg)

```

#####
#0
#F NumberSmallCatOneGroups
## Input: A positive integer m or two positive integers m,k
## Output: The number of cat-1-groups of order m or the number of
##          non-isomorphic cat-1-group structures on the kth group of
##          order m from the database of small groups
##
InstallGlobalFunction(NumberSmallCatOneGroups, function(arg)
local m,k;

    m:=arg[1];
    if m >HAP_CAT_SIZE then
        Print("This function only apply for order less than or equal to ",
              HAP_CAT_SIZE, ".\m");
        return fail;
    fi;

```

```

if Length(arg)=1 then
  return Sum(List([1..NumberSmallGroups(m)],
    k->Length(HAP_CAT[m][k])));
fi;
if Length(arg)=2 then
  k:=arg[2];
  if k>NumberSmallGroups(m) then
    Print("There are only ",NumberSmallGroups(m),
  " groups of order ",m,"\m");
    return fail;
  fi;
  return Length(HAP_CAT[m][k]);
fi;

return fail;
end);
##
##### end of NumberSmallCatOneGroups #####

```

### 8.3.14 SmallCatOneGroup(m,k,i)

```

#####
#0
#F SmallCatOneGroup
## Input: Three positive integers m,k,i with m <=255
## Output: The ith cat-1-group structure on SmallGroup(m,k)
##
InstallGlobalFunction(SmallCatOneGroup, function(m,k,i)
local S,sm,x,s,t,p;

sm:=Length(HAP_CAT[m][k]);
if i>sm then
Print("There are only ",sm," cat-1-groups of SmallGroup(",
m,",",k,")\m");
return fail;
fi;

S:=SmallGroup(m,k);
x:=HAP_CAT[m][k][i];
if [m,k] in HAP_CAT_PERM then
s:=GroupHomomorphismByImages(S,S,x[1][1],x[1][2]);
t:=GroupHomomorphismByImages(S,S,x[2][1],x[2][2]);
else
p:=Pcgs(S);
s:=GroupHomomorphismByImages(S,S,p,List(x[1],
m->PcElementByExponents(p,m)));
t:=GroupHomomorphismByImages(S,S,p,List(x[2],
m->PcElementByExponents(p,m)));
fi;
return Objectify(HapCatOneGroup,
rec(sourceMap:=s,
targetMap:=t
));
end);
##
##### end of SmallCatOneGroup #####

```

### 8.3.15 IsomorphismCatOneGroups(C,D)

```
#####
#0
#F IsomorphismCatOneGroups
## Input: Two finite cat-1-groups C and D
## Output: An isomorphism between C and D if they are isomorphic
##         and fail otherwise
##
InstallGlobalFunction(IsomorphismCatOneGroups, function(C,D)
local
    sC,tC,G,sD,tD,GD,
    xC,xD,A,attr,actAttr,M,p,f,h,
    ActToMap,ActToPair,ActToSubgroup,FindOrbit,processOrbit,
    Map,map;

#####
#1
ActToMap:=function(s,f)
    return InverseGeneralMapping(f)*s*f;
end;
##
#####
#1
ActToPair:=function(p,f)
    local h;
    h:=InverseGeneralMapping(f);
    return [h*p[1]*f,h*p[2]*f];
end;
##
#####
#1
ActToSubgroup:=function(K,f)
    return Image(f,K);
end;
##
#####
#1
#F FindOrbit
##
FindOrbit:=function(A,attr,actAttr)
local
    Gens,Orb,T,Dict,S,op,qs,g,p,img,h;

    Gens:=SmallGeneratingSet(A);
    Orb:=[attr(xC)];
    T:=[One(A)];
    Dict:=NewDictionary(Orb[1],true);
    AddDictionary(Dict,Orb[1],1);
    S:=TrivialSubgroup(A);
    op:=1;
    qs:=Size(A);
    while op<=Length(Orb) and Size(S)<qs do
        for g in Gens do
            img:=actAttr(Orb[op],g);
            p:=LookupDictionary(Dict,img);
            if p=fail then
                Add(Orb,img);
                AddDictionary(Dict,img,Length(Orb));
            end;
        end;
        op:=op+1;
    end;
end;
#####
```

```

        Add(T,T[op]*g);
        qs:=Size(A)/Length(Orb);
    elif Size(S)<=qs/2 then # otherwise stabilizer cant grow
        h:=T[op]*g/T[p];
        S:=ClosureSubgroup(S,h);
    fi;
od;
op:=op+1;
od;
return [Dict,S,T];
end;
##
#####
##
processOrbit:=function()

    M:=FindOrbit(A,attr,actAttr);
    p:=LookupDictionary(M[1],attr(xD));
    if p=fail then
        return fail;
    fi;
    A:=M[2];
    if p<>1 then
        h:=M[3][p]^-1;
        f:=f*h;
        xD:=ActToPair(xD,h);
    fi;
end;
##
#####

#####
sC:=C!.sourceMap;
tC:=C!.targetMap;
G:=Source(sC);
sD:=D!.sourceMap;
tD:=D!.targetMap;
GD:=Source(sD);

f:=IsomorphismGroups(GD,G);
if f = fail then
    return fail;
fi;
if IsomorphismGroups(HomotopyGroup(C,1),HomotopyGroup(D,1))=fail then
    return fail;
fi;
if IsomorphismGroups(HomotopyGroup(C,2),HomotopyGroup(D,2))=fail then
    return fail;
fi;
if IsomorphismGroups(Kernel(tC),Kernel(tD))=fail then
    return fail;
fi;
if IsomorphismGroups(Kernel(sC),Kernel(sD))=fail then
    return fail;
fi;
if IsomorphismGroups(Image(tC),Image(tD))=fail then
    return fail;
fi;
if IsomorphismGroups(Image(sC),Image(sD))=fail then

```

```

    return fail;
fi;

#####
##
Map:=function()
  xC:=[sC,tC];
  xD:=ActToPair([sD,tD],f);
  if xC=xD then
    return InverseGeneralMapping(f);
  fi;

  ##### Image of first component #####
  A:=AutomorphismGroup(G);
  attr:=s->Image(s[1]);
  actAttr:=ActToSubgroup;
  processOrbit();

  ##### Kernel of first component #####
  attr:=s->Kernel(s[1]);
  actAttr:=ActToSubgroup;
  processOrbit();
  if xC=xD then
    return InverseGeneralMapping(f);
  fi;

  ##### Image of second component #####
  attr:=s->Image(s[2]);
  actAttr:=ActToSubgroup;
  processOrbit();
  if xC=xD then
    return InverseGeneralMapping(f);
  fi;

  ##### Kernel of second component #####
  attr:=s->Kernel(s[1]);
  actAttr:=ActToSubgroup;
  processOrbit();
  if xC=xD then
    return InverseGeneralMapping(f);
  fi;

  ##### First component #####
  attr:=s->s[1];
  actAttr:=ActToMap;
  processOrbit();
  if xC=xD then
    return InverseGeneralMapping(f);
  fi;

  ##### Second component #####
  attr:=s->s[2];
  actAttr:=ActToMap;
  processOrbit();
  if xC=xD then
    return InverseGeneralMapping(f);
  fi;
  return fail;
end;
##
#####

```

```

map:=Map();
if map=fail then
  return fail;
fi;
return Objectify(HapCatOneGroupMorphism,
  rec(
    source:= C,
    target:= D,
    mapping:= map
  ));
end);
##
##### IsomorphismCatOneGroups #####

```

### 8.3.16 IdCatOneGroup(C)

```

#####
#0
#F IdCatOneGroup
## Input: A cat-1-group C of order <= 255
## Output: A triple [m,k,i] where C is isomorphic to the ith cat-1-group
## structure on SmallGroup(m,k).
##
InstallGlobalFunction(IdCatOneGroup, function(C)

local
  ActToMap,ActToPair,ActToSubgroup,FindOrbit,processOrbit,
  s,t,G,nk,n,k,S,f,Lst,x,tmp,i,p,Imgs,A,xC,Ln,CLn,attr,actAttr,M;

#####
#1
ActToMap:=function(s,f)
  return InverseGeneralMapping(f)*s*f;
end;
##
#####
#1
ActToPair:=function(p,f)
  local h;
  h:=InverseGeneralMapping(f);
  return [h*p[1]*f,h*p[2]*f];
end;
##
#####
#1
ActToSubgroup:=function(K,f)
  return Image(f,K);
end;
##
#####

#####
#1
#F FindOrbit
##
FindOrbit:=function(A,attr,actAttr)
local Gens,Orb,T,Dict,S,op,qs,g,p,img,h;

```

```

Gens:=SmallGeneratingSet(A);
Orb:=[attr(xC)];
T:=[One(A)];
Dict:=NewDictionary(Orb[1],true);
AddDictionary(Dict,Orb[1],1);
S:=TrivialSubgroup(A);
op:=1;
qs:=Size(A);
while op<=Length(Orb) and Size(S)<qs do
  for g in Gens do
    img:=actAttr(Orb[op],g);
    p:=LookupDictionary(Dict,img);
    if p=fail then
      Add(Orb,img);
      AddDictionary(Dict,img,Length(Orb));
      Add(T,T[op]*g);
      qs:=Size(A)/Length(Orb);
    elif Size(S)<=qs/2 then # otherwise stabilizer cant grow
      h:=T[op]*g/T[p];
      S:=ClosureSubgroup(S,h);
    fi;
  od;
  op:=op+1;
od;
return [Dict,S,T];
end;
##
#####
##
processOrbit:=function()
  M:=FindOrbit(A,attr,actAttr);
  for i in Ln do
    p:=LookupDictionary(M[1],attr(Lst[i]));
    if p<>fail then
      Add(CLn,i);
      Lst[i]:=ActToPair(Lst[i],M[3][p]^-1);
    fi;
  od;
end;
##
#####
s:= C!.sourceMap;
t:= C!.targetMap;
G:=Source(s);
n:=Size(G);
if n>HAP_CAT_SIZE then
  Print("This function only apply for cat-1-groups of order less than ",
        HAP_CAT_SIZE+1,"\n");
  return fail;
fi;
nk:=IdGroup(G);
k:=nk[2];
S:=SmallGroup(n,k);
f:=IsomorphismGroups(G,S);
Lst:=[];
if nk in HAP_CAT_PERM then
  for x in HAP_CAT[n][k] do
    tmp:=[];

```

```

        for i in [1..2] do
            tmp[i]:=GroupHomomorphismByImages(S,S,x[i][1],x[i][2]);
        od;
        Add(Lst,tmp);
    od;
else
    p:=Pcgs(S);
    for x in HAP_CAT[n][k] do
        tmp:=[];
        for i in [1..2] do
            Imgs:=List(x[i],m->PcElementByExponents(p,m));
            tmp[i]:=GroupHomomorphismByImages(S,S,p,Imgs);
        od;
        Add(Lst,tmp);
    od;
fi;

A:=AutomorphismGroup(S);
xC:=ActToPair([s,t],f);

##### Image of first component #####
A:=AutomorphismGroup(S);
Ln:=[1..Length(Lst)];
CLn:=[];
attr:=s->Image(s[1]);
actAttr:=ActToSubgroup;
processOrbit();
if Length(CLn) =1 then
    return [n,k,CLn[1]];
fi;
A:=M[2];
Ln:=CLn;
CLn:=[];

##### Kernel of first component #####
attr:=s->Kernel(s[1]);
actAttr:=ActToSubgroup;
processOrbit();
if Length(CLn) =1 then
    return [n,k,CLn[1]];
fi;
A:=M[2];
Ln:=CLn;
CLn:=[];

##### Image of second component #####
attr:=s->Image(s[2]);
actAttr:=ActToSubgroup;
processOrbit();
if Length(CLn) =1 then
    return [n,k,CLn[1]];
fi;
A:=M[2];
Ln:=CLn;
CLn:=[];

##### Kernel of second component #####
attr:=s->Kernel(s[1]);
actAttr:=ActToSubgroup;

```

```

processOrbit();
if Length(CLn) =1 then
    return [n,k,CLn[1]];
fi;
A:=M[2];
Ln:=CLn;
CLn:=[];

##### First component #####
attr:=s->s[1];
actAttr:=ActToMap;
processOrbit();
if Length(CLn) =1 then
    return [n,k,CLn[1]];
fi;
A:=M[2];
Ln:=CLn;
CLn:=[];

##### Second component #####
attr:=s->s[2];
actAttr:=ActToMap;
processOrbit();
if Length(CLn) =1 then
    return [n,k,CLn[1]];
fi;
return fail;
end);
##
##### end of IdCatOneGroup #####

```

### 8.3.17 NumberSmallCrossedModules(m)

```

#####
#0
#F NumberSmallCrossedModules
## Input: A positive integer m<=255
## Output: The number of crossed modules of order m
##
InstallGlobalFunction(NumberSmallCrossedModules, function(m)

    if m >HAP_CAT_SIZE then
        Print("This function only apply for order less than or equal to ",
            HAP_CAT_SIZE, ".\m");
        return fail;
    fi;
    return Sum(HAP_CAT[m],x->Length(x));
end);
##
##### NumberSmallCrossedModules #####

```

### 8.3.18 SmallCrossedModule(m,k)

```

#####
#0

```

```

#F SmallCrossedModule
## Input: Two positive integers m,k with m<=255
## Output: The kth crossed module of order m in the database of
##         small crossed module
##
InstallGlobalFunction(SmallCrossedModule, function(m,k)
local sum,t,i;

  if m >HAP_CAT_SIZE then
    Print("This function only apply for order less than or equal to ",
          HAP_CAT_SIZE, ".\m");
    return fail;
  fi;
  if k >NumberSmallCrossedModules(m) then
    Print("There are only ",NumberSmallCrossedModules(m),
          " crossed modules of order ",m, "\m");
    return fail;
  fi;
  sum:=0;
  t:=0;
  while sum<k do
    i:=k-sum;
    t:=t+1;
    sum:=sum+Length(HAP_CAT[m][t]);
  od;
  return CrossedModuleByCatOneGroup(SmallCatOneGroup(m,t,i));
end);
##
##### SmallCrossedModule #####

```

### 8.3.19 IsomorphismCrossedModules(XC,XD)

```

#####
#0
#F IsomorphismCrossedModules
## Input: Two finite crossed modules XC, XD
## Output: An isomorphism between XC and XD if they are isomorphic and
##         fail otherwise
##
InstallGlobalFunction(IsomorphismCrossedModules, function(XC,XD)
local
  C,D,Iso,f,GC,GD,MC,MD,
  proD,emb1C,emb2C,emb1D,emb2D,
  Gens,Imgs,m,x,px,Map;

  C:=CatOneGroupByCrossedModule(XC);
  D:=CatOneGroupByCrossedModule(XD);
  Iso:=IsomorphismCatOneGroups(C,D);
  if Iso=fail then
    return fail;
  fi;
  f:=Iso!.mapping;
  GC:=Source(f);
  GD:=Range(f);

  MC:=Source(XC!.map);
  MD:=Source(XD!.map);

```

```

proD:=Projection(GD);
emb1C:=Embedding(GC,1);
emb2C:=Embedding(GC,2);
emb1D:=Embedding(GD,1);
emb2D:=Embedding(GD,2);

Gens:=GeneratorsOfGroup(MC);
Imgs:=[];
for m in Gens do
  x:=Image(f,Image(emb2C,m));
  px:=Image(emb1D,Image(proD,x));
  Add(Imgs,PreImagesRepresentative(emb2D,px^(-1)*x));
od;

#####
##
Map:=function(n)
  if n=1 then
    return GroupHomomorphismByImages(MC,MD,Gens,Imgs);
  fi;
  if n=2 then
    return emb1C*f*proD;
  fi;
  Print("Only apply for n =1,2");
  return fail;
end;
##
#####
return Objectify(HapCrossedModuleMorphism,
  rec(source:=XC,
    target:=XD,
    mapping:=Map
  ));
end);
##
##### IsomorphismCrossedModules #####

```

### 8.3.20 IdCrossedModule(X)

```

#####
#0
#F IdCrossedModule
## Input: A crossed module X of order <=255
## Output: A pair [n,k] where X is isomorphic to the kth crossed module
## of order n in the database of small crossed module
##
InstallGlobalFunction(IdCrossedModule, function(X)
local T;

  if Order(X) > HAP_CAT_SIZE then
    Print("This function only apply for crossed module of order <=",
      HAP_CAT_SIZE, ".\n");
    return fail;
  fi;
  T:=IdCatOneGroup(CatOneGroupByCrossedModule(X));
  return [T[1],Sum(List([1..T[2]-1],m->

```

```

                Length(HAP_CAT[T[1]][m])))+T[3]];
end);
##
##### IdCrossedModule #####

```

### 8.3.21 SubQuasiIsomorph(C)

```

#####
#0
#F SubQuasiIsomorph
## Input: A finite cat-1-group C
## Output: A quasi-isomorphic sub cat-1-group of C
##
InstallGlobalFunction(SubQuasiIsomorph,function(C)
local
    s,t,G,H,Kers,Kert,Kersnt,tKers,OrdPiOne,OrdPiTwo,OrdPi,
    LS,Lx,x,sx,Ordsx,flag,
    newGens,news,newt;

    s:= C!.sourceMap;
    t:= C!.targetMap;
    G:=Source(s);
    Kers:=Kernel(s);
    Kert:=Kernel(t);
    Kersnt:=Intersection(Kers,Kert);
    tKers:=Image(t,Kers);
    OrdPiOne:=Order(HomotopyGroup(C,1));
    OrdPiTwo:=Order(HomotopyGroup(C,2));
    OrdPi:=OrdPiOne*OrdPiTwo;
    LS:=ConjugacyClassesSubgroups(LatticeSubgroups(G));
    if not IsMutable(LS) then
        LS:= ShallowCopy(LS);
    fi;
    Sort(LS,function(x,y) return Size(x[1])<Size(y[1]); end);
    flag:=0;
    for Lx in LS do
        x:=Lx[1];
        if Order(x)>= OrdPi then
            if IsSubgroup(x,Kersnt) then
                for x in Lx do
                    if IsSubgroup(x,Image(s,x)) then
                        if IsSubgroup(x,Image(t,x)) then
                            sx:=Image(s,x);
                            Ordsx:=Order(sx);
                            if Ordsx=Order(Image(t,Intersection(Kers,x)))
*OrdPiOne then
                                if Ordsx=Order(Intersection(sx,tKers))*OrdPiOne then
                                    H:=x;
                                    flag:=1;
                                    break;
                                fi;
                                fi;
                            fi;
                            fi;
                        od;
                    fi;
                    fi;
                fi;
            fi;
        fi;
    fi;

```

```

        if flag =1 then
            break;
        fi;
    od;
    if H=G then
        return C;
    fi;
    newGens:=GeneratorsOfGroup(H);
    news:=GroupHomomorphismByImagesNC(H,H,newGens,
        List(newGens,x->Image(s,x)));
    newt:=GroupHomomorphismByImagesNC(H,H,newGens,
        List(newGens,x->Image(t,x)));
    return Objectify(HapCatOneGroup,rec(
        sourceMap:=news,
        targetMap:=newt));
end);
##
##### end of SubQuasiIsomorph #####

```

### 8.3.22 QuotientQuasiIsomorph(C)

```

#####
#0
#F QuotientQuasiIsomorph
## Input: A finite cat-1-group C
## Output: A quasi-isomorphic quotient cat-1-group of C
##
InstallGlobalFunction(QuotientQuasiIsomorph, function (C)
local
    s,t,G,H,Kers,Kert,Kersnt,Ims,OrdIms,Imt,OrdPiOne,OrdPiTwo,Ord,
    LN,x,n,i,
    OrderPiOneGx,OrderPiTwoGx,
    epi,newG,newGens,news,newt;

    s:=C!.sourceMap;
    t:=C!.targetMap;
    G:=Source(s);
    Kers:=Kernel(s);
    Ims:=Image(s);
    OrdIms:=Order(Ims);
    Imt:=Image(t);
    Kert:=Kernel(t);
    Kersnt:=Intersection(Kers,Kert);
    OrdPiOne:= Order(HomotopyGroup(C,1));
    OrdPiTwo:= Order(HomotopyGroup(C,2));
    Ord:=Order(G)/(OrdPiOne*OrdPiTwo);

#####
#1
OrderPiOneGx:=function(x)
local tsx;

    tsx:=Image(t,PreImages(s,Intersection(Ims,x)));
    return (OrdIms*Order(Intersection(tsx,x)))/
        (Order(Intersection(Ims,x))*Order(tsx));
end;
##

```

```

#####
#1
OrderPiTwoGx:=function(x)
local f;

    f:=NaturalHomomorphismByNormalSubgroup(G,x);
    return Order(Intersection(Image(f,PreImages(s,Intersection
      (Ims,x))),Image(f,PreImages(t,Intersection(Imt,x))))));
end;
##
#####

LN:=NormalSubgroups(G);
if not IsMutable(LN) then
  LN:= ShallowCopy(LN);
fi;
Sort(LN,function(x,y) return Size(x)>Size(y); end);
n:=Length(LN);
for i in [1..n] do
  x:=LN[i];
  if Order(x) <= Ord then
    if IsSubgroup(x,Image(s,x)) then
      if IsSubgroup(x,Image(t,x)) then
        if IsSubgroup(x,CommutatorSubgroup(PreImages(s,Intersection
          (Ims,x)),PreImages(t,Intersection(Imt,x)))) then
          if Order(Kersnt) = Order(Intersection(Kersnt,x))*OrdPiTwo then
            if OrderPiTwoGx(x) = OrdPiTwo then
              if OrderPiOneGx(x) = OrdPiOne then
                H:=x;
                break;
              fi;
            fi;
          fi;
        fi;
      fi;
    fi;
  fi;
od;
if Order(H)=1 then
  return C;
fi;

epi:=NaturalHomomorphismByNormalSubgroup(G,H);
newG :=Image(epi);
newGens:=GeneratorsOfGroup(newG);
news:=GroupHomomorphismByImagesNC(newG,newG,newGens,List(newGens,
  x->Image(epi,Image(s,PreImagesRepresentative(epi,x)))));
newt:=GroupHomomorphismByImagesNC(newG,newG,newGens,List(newGens,
  x->Image(epi,Image(t,PreImagesRepresentative(epi,x)))));
return Objectify(HapCatOneGroup,rec(
  sourceMap:=news,
  targetMap:=newt));
end);
##
##### end of QuotientQuasiIsomorph #####

```

### 8.3.23 QuasiIsomorph(X)

```
#####
#0
#F QuasiIsomorph
## Input: A finite cat-1-group or a finite crossed module X
## Output: A quasi-isomorphism of X
##
InstallGlobalFunction(QuasiIsomorph,function (X)
local QuasiIsomorphOfCat, QuasiIsomorphOfCross;

#####
#1
#F QuasiIsomorphOfCat
## Input: A finite cat-1-group C
## Output: A quasi-isomorphism of C
##
QuasiIsomorphOfCat:=function(C)
local D;

    D:=QuotientQuasiIsomorph(C);
    D:=SubQuasiIsomorph(D);
    while Size(D) < Size(C) do
        C:=D;
        D:=QuotientQuasiIsomorph(C);
        if Size(D) < Size(C) then
            D:=SubQuasiIsomorph(D);
        fi;
    od;
    return D;
end;
##
##### end of QuasiIsomorphOfCat #####

#####
#1
#F QuasiIsomorphOfCross
## Input: A finite crossed module XC
## Output: A quasi-isomorphism of XC
##
QuasiIsomorphOfCross:=function(XC)
local C,D;

    C:=CatOneGroupByCrossedModule(XC);
    D:=QuasiIsomorphOfCat(C);
    return CrossedModuleByCatOneGroup(D);
end;
##
##### end of QuasiIsomorphOfCross #####

if IsHapCatOneGroup(X) then
    return QuasiIsomorphOfCat(X);
fi;
if IsHapCrossedModule(X) then
    return QuasiIsomorphOfCross(X);
fi;
end);
##
##### end of QuasiIsomorph #####
```

### 8.3.24 Homology(X,n)

```
#####
#0
#0 Homology
## Input: A crossed module X and an integer n>=0
## Output: The integral homology H_n(X,Z)
##
InstallOtherMethod(Homology, "Homology of crossed modules",
[IsHapCrossedModule,IsInt], function(X,n)
local C,D,N,K;

    C:=CatOneGroupByCrossedModule(X);
    D:=QuasiIsomorph(C);
    N:=NerveOfCatOneGroup(D,n+1);
    K:=ChainComplexOfSimplicialGroup(N);
    return Homology(K,n);
end);
##### end of Homology #####
```

### 8.3.25 HomotopyCrossedModule(X)

```
#####
#0
#F HomotopyCrossedModule
## Input: A crossed module X
## Output: The homotopy crossed module 0:pi_2(X)->pi_1(X) of X
##
InstallGlobalFunction(HomotopyCrossedModule, function(X)
local phi,act,P,A,nat,G,Gens,alpha;

    phi:=X!.map;
    act:=X!.action;
    P:=Range(phi);
    A:=Kernel(phi);
    nat:=NaturalHomomorphismByNormalSubgroup(P,Image(phi));
    G:=Range(nat);
    #####
    #1
    alpha:=function(g,a)
    local x;

        x:=PreImagesRepresentative(nat,g);
        return act(x,a);
    end;
    ##
    #####
    Gens:=GeneratorsOfGroup(A);
    return Objectify(HapCrossedModule,rec(
        map:=GroupHomomorphismByImages(A,G,Gens,List(Gens,x->One(G))),
        action:=alpha
    ));
end);
##
##### end of HomotopyCrossedModule #####
```

### 8.3.26 NumberSmallQuasiCrossedModules(m)

```
#####
#0
#F NumberSmallQuasiCrossedModules
## Input: A positive integer m<=255
## Output: The number of quasi-isomorphism classes of order m.
##
InstallGlobalFunction(NumberSmallQuasiCrossedModules, function(m)

  if (m > HAP_QCAT_SIZE) or (m in HAP_QCAT_NOT) then
    Print("This function only apply for order < ",HAP_QCAT_SIZE+1);
    Print(" and not in ",HAP_QCAT_NOT,"\n");
    return fail;
  fi;
  return Length(HAP_SMALL_QCAT[m]);
end);
##
##### end of NumberSmallQuasiCrossedModules #####
```

### 8.3.27 SmallQuasiCrossedModule(m,k)

```
#####
#0
#F SmallQuasiCrossedModule
## Input: Two positive integers m,k with m<=255
## Output: The smallest representative of the kth quasi-isomorphism
##         classes of order m.
##
InstallGlobalFunction(SmallQuasiCrossedModule, function(m,k)
local t,x;

  if (m > HAP_QCAT_SIZE) or (m in HAP_QCAT_NOT) then
    Print("This function only apply for order < ",HAP_QCAT_SIZE+1);
    Print(" and not in ",HAP_QCAT_NOT,"\m");
    return fail;
  fi;
  t:=Length(HAP_SMALL_QCAT[m]);
  if k> t then
    Print("There are only ",t," quasi-isomorphism classes of order ",m,"\m");
    return fail;
  fi;
  x:=HAP_SMALL_QCAT[m][k];
  return CrossedModuleByCatOneGroup(SmallCatOneGroup(m,x[1],x[2]));
end);
##
##### end of SmallQuasiCrossedModule #####
```

### 8.3.28 IdQuasiCrossedModule(X)

```
#####
#0
#F IdQuasiCrossedModule
## Input: A finite crossed module X
## Output: A pair of integers [m,k] where X is quasi-isomorphic to
```

```

##          SmallQuasiCrossedModule(m,k)
##
InstallGlobalFunction(IdQuasiCrossedModule, function(X)
local C,x;

    C:=QuasiIsomorph(CatOneGroupByCrossedModule(X));
    x:=IdCatOneGroup(C);
    if (x[1] > HAP_QCAT_SIZE) or (x[1] in HAP_QCAT_NOT) then
        Print("This function only apply for order < ",HAP_QCAT_SIZE+1);
        Print(" and not in ",HAP_QCAT_NOT,"\n");
        return fail;
    fi;
    return HAP_ID_QCAT[x[1]][x[2]][x[3]];
end);
##
##### end of IdQuasiCrossedModule #####

```

### 8.3.29 HomotopyLowerCentralSeriesOfCrossedModule(X)

```

#####
#O
#F HomotopyLowerCentralSeriesOfCrossedModule
## Input: A crossed module X with  $\pi_1(X)$ ,  $\pi_2(X)$  p -groups
## Output: The homotopy lower central series of X
##
InstallGlobalFunction(HomotopyLowerCentralSeriesOfCrossedModule,
function(X)
local
    del,act,M,P,GensM,ImgGensM,A,G,nat,Gs,Ps,
    nOne,XOne,i,phi,MorphismOne,
    Gens,As,nTwo,a,g,natMs,Ms,XTwo,GenMs,
    PreImGenMs,MorphismTwo,
    ActOne,MapOne,MapTwo;

    del:=X!.map;
    act:=X!.action;
    M:=Source(del);
    P:=Range(del);
    GensM:=GeneratorsOfGroup(M);
    ImgGensM:=List(GensM,m->Image(del,m));
    nat:=NaturalHomomorphismByNormalSubgroup(P,Image(del));
    A:=Kernel(del);
    G:=Range(nat);
    Gs:=[G];
    Ps:=[P];
    nOne:=1;
    while not IsTrivial(Gs[nOne]) do
        nOne:=nOne+1;
        Gs[nOne]:=CommutatorSubgroup(Gs[nOne-1],G);
        Ps[nOne]:=PreImage(nat,Gs[nOne]);
    od;
    MorphismOne:=[];
    if nOne>1 then
        Ps:=Reversed(Ps);
        XOne:=[];
        for i in [1..nOne-1] do
            phi:=GroupHomomorphismByImages(M,Ps[i],GensM,ImgGensM);

```

```

    XOne[i]:=Objectify(HapCrossedModule,
                      rec(map:=phi,
                          action:=act
                          ));
od;
XOne[nOne]:=X;
#####
#1
MapOne:= function(i)
  return function(n)
    local Gens;
    if n = 1 then
      return IdentityMapping(M);
    fi;
    if n =2 then
      Gens:=GeneratorsOfGroup(Ps[i]);
      return GroupHomomorphismByImages(Ps[i],
                                         Ps[i+1],Gens,Gens);
    fi;
  end;
end;
##
#####

for i in [1..nOne-1] do
  MorphismOne[i]:=Objectify(HapCrossedModuleMorphism,
                             rec(source:=XOne[i],
                                 target:=XOne[i+1],
                                 mapping:=MapOne(i)
                                 ));
od;
fi; ##### end of nOne>1

G:=List(G,g->PreImagesRepresentative(nat,g));
As:=[A];
nTwo:=1;
while not IsTrivial(As[nTwo]) do
  Gens:=[];
  for a in As[nTwo] do
    for g in G do
      Add(Gens,a*act(g,a^(-1)));
    od;
  od;
  nTwo:=nTwo+1;
  As[nTwo]:=Group(Gens);
od;
MorphismTwo:=[];
if nTwo>1 then
  As:=Reversed(As);
  natMs:=[IdentityMapping(M)];
  Ms:=[M];
  XTwo:=[X];
  GenMs:=[GensM];
  PreImGenMs:=[GensM];

#####
#1
ActOne:=function(i)
  return function(p,mA)

```

```

        return Image(natMs[i],act(p,
            PreImagesRepresentative(natMs[i],mA)));
    end;
end;
##
#####

for i in [2..nTwo] do
    natMs[i]:=NaturalHomomorphismByNormalSubgroup(M,As[i]);
    Ms[i]:=Range(natMs[i]);
    GenMs[i]:=GeneratorsOfGroup(Ms[i]);
    PreImGenMs[i]:=List(GenMs[i],
        m->PreImagesRepresentative(natMs[i],m));
    phi:=GroupHomomorphismByImages(Ms[i],P,GenMs[i],
        List(PreImGenMs[i],m->Image(del,m)));
    XTwo[i]:=Objectify(HapCrossedModule,
        rec(map:=phi,
            action:=ActOne(i)
        ));
od;

#####
#1
MapTwo:=function(i)
    return function(n)
        if n = 1 then
            return GroupHomomorphismByImages(Ms[i],Ms[i+1],
                GenMs[i],List(PreImGenMs[i],
                    m->Image(natMs[i+1],m)));
        fi;
        if n =2 then
            return IdentityMapping(P);
        fi;
    end;
end;
##
#####

for i in [1..nTwo-1] do
    MorphismTwo[i]:=Objectify(HapCrossedModuleMorphism,
        rec(source:=XTwo[i],
            target:=XTwo[i+1],
            mapping:=MapTwo(i)
        ));
od;
fi; ##end of nTwo>1
return Concatenation(MorphismOne,MorphismTwo);
end);
##
##### end of HomotopyLowerCentralSeriesOfCrossedModule #####

```

### 8.3.30 PersistentHomologyOfCrossedModule(X,n)

```

#####
#0
#0 PersistentHomologyOfCrossedModule
## Input: A crossed module X with  $\pi_1(X)$ ,  $\pi_2(X)$  p-groups and an

```

```
##          integer n>=0
## Output: The matrix of persistent Betti numbers of X at degree n
##
InstallGlobalFunction(PersistentHomologyOfCrossedModule, function(X,n)
local
  p,Maps,
  PrimeOne,PrimeTwo,PrimeOneTwo;

  PrimeOne:=PrimeDivisors(Size(HomotopyGroup(X,1)));
  PrimeTwo:=PrimeDivisors(Size(HomotopyGroup(X,2)));
  PrimeOneTwo:=Set(Concatenation(PrimeOne,PrimeTwo));
  if Length(PrimeOneTwo) <>1 then
    return fail;
  fi;

  p:=PrimeOneTwo[1];
  Maps:=HomotopyLowerCentralSeriesOfCrossedModule(X);
  Maps:=CatOneGroupByCrossedModule(Maps);
  Maps:=NerveOfCatOneGroup(Maps,n+1);
  Maps:=ChainComplexOfSimplicialGroup(Maps);
  Maps:=List(Maps,f->TensorWithIntegersModP(f,p));
  Maps:=List(Maps,f->HomologyVectorSpace(f,n));
  return LinearHomomorphismsPersistenceMat(Maps);
end);
##
##### end of PersistentHomologyOfCrossedModule #####
```