Discrete Vector Fields

and the

Cohomology of Certain Arithmetic and Crystallographic Groups

PhD thesis

by

Bui Anh Tuan

Supervisor: Professor Graham Ellis

School of Mathematics, Statistics and Applied Mathematics

National University of Ireland, Galway

May 2015

Contents

Declaration

I, Bui Anh Tuan, 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 express my deepest gratitude to my supervisor, Professor Graham Ellis, whose expertise, understanding, and patience, added considerably to my graduate experience. I appreciate his vast knowledge and skill in many areas, and his assistance in writing reports (i.e., grant proposals, scholarship applications and this thesis). Without his guidance, I would never have been able to finish this thesis.

I place on record, my sincere thank to College of Science and the Irish Research Council for financial supporting me to do something I really enjoyed. I take this opportunity to express gratitude to the staff and other postgraduates in the Department of Mathematics for their help and support.

I am also grateful to Dr. Alexander Rahm, lecturer, in the School of Maths. I am extremely thankful and indebted to him for sharing expertise, and sincere and valuable guidance and encouragement extended to me.

I would also like to thank my family for the support they provided me through my entire life.

Galway, May, 2015

Bui Anh Tuan

List of symbols

$\mathbb{Z}$	integer numbers $\{\ldots, -2, -1, 0, 1, 2, \ldots\}$
$\mathbb{Z}G$	integral group ring
C_n	cyclic group of order n
$\oplus$	direct sum
$\otimes$	tensor product
$\rtimes$	semidirect product
1_A	identity map from A to A
$\langle S \rangle$	group generated by the set S
$\mathbf{1}$	trivial group
$\Gamma_0^m(p)$	congruence subgroup of $SL_2(\mathbb{Z}[1/m])$ of level p
$\mathrm{Gram}(P)$	Gramian matrix
R_*^G	free resolution for group G
$C_*(X)$	cellular chain complex of space X
V	discrete vector field
$H^*(G, \mathbb{Z})$	cohomology ring of group G

Summary

This thesis makes the following contributions to the area of Computational Algebraic Topology:

1. All algorithms written in this thesis are implemented and are publicly available as documented functions for the GAP [16] computer algebra system, and are distributed with the system as part of its HAP [11] package.

2. We devise and implement an algorithm for computing a finite $\mathbb{Z}G$-equivariant CW-space with nice cell stabilizer groups and a contracting discrete vector field, where $G = SL_2(\mathbb{Z}[1/m])$ for any positive integer m. (See Algorithm 3.2.1.)

3. We implement a function which inputs a non-free $\mathbb{Z}G$-resolution and outputs finitely many terms of a free $\mathbb{Z}G$-resolution R^G_* of $\mathbb{Z}$, where $G = SL_2(\mathbb{Z}[1/m])$ for any positive integer m. (See Algorithm 2.1.1.)

4. We devise and implement an algorithm for computing finitely many terms of a free $\mathbb{Z}H$-resolution R^H_* of $\mathbb{Z}$ for H a finite index subgroup of $G = SL_2(\mathbb{Z}[1/m])$. (See Algorithm 3.2.2.)

5. We devise and implement an algorithm that attempts to find a cubical fundamental cell for a cubical crystallographic group G. (See Algorithm 4.3.1).

6. We devise and implement an algorithm that inputs a crystallographic group G together with a cubical fundamental cell and outputs a finite $\mathbb{Z}G$-equivariant CW-space with nice cell stabilizer groups and a contracting discrete vector field. (See Algorithm 4.4.1).

7. We implement a function for calculating finitely many terms of a free $\mathbb{Z}G$-resolution R_*^G of $\mathbb{Z}$ with contracting homotopy, where G is an n-dimensional cubical crystallographic group. This resolution can be used to compute the cohomology ring structure of G. (See Algorithm 4.5.1.)

8. We give the complete list of 3-dimensional cubical Bieberbach groups with their cohomology ring structures.(See Section 4.7.)

Chapter 1

Introduction

1.1 Outline of the thesis

This thesis has four chapters.

Chapter 1 includes two sections. In the first section we present the main goal of the thesis, namely methods for calculating the integral homology of $SL(2,\mathbb{Z}[1/m])$ and the cohomology ring structure of certain crystallographic groups. The second section recalls standard material that will be used in the thesis.

Chapter 2 introduces discrete vector fields in the context of group cohomology and explains how they can be used to find a contracting homotopy on a classifying space. We also recall a homological perturbation lemma that can be used to construct a free resolution from a non-free resolution.

Chapter 3 is devoted to constructing a free resolution for $SL(2,\mathbb{Z}[1/m])$ where m is any positive integer, and calculating the homology of such groups. We describe an algorithm for computing finitely many terms of a free $\mathbb{Z}G$-resolution of $\mathbb{Z}$ for G a finite index subgroup of $SL(2,\mathbb{Z}[1/m])$. An implementation of the algorithm is used to determine the integral homology groups $H_n(SL(2,\mathbb{Z}[1/m]),\mathbb{Z})$ for all integers $m \leq 50$ $(m \neq 36,\ 42)$, and $n \geq 0$.

Chapter 4 provides a method for calculating the cohomology ring structure for Euclidean crystallographic groups with cubical fundamental region. We describe algorithms for attempting to decide if a given crystallogrpahic group admits a cubical fundamental region; and calculating the resulting cellular chain complex as a $\mathbb{Z}G$-resolution. We also give a method for computing the contracting homotopy on the chain complex. Finally we provide an algorithm for calculating the cohomology cup product.

Appendix includes data types which are used in the implementation of our algorithms and the full list of implemented functions in GAP programing language.

1.2 Main goals of the thesis

This thesis has two main topics: Arithmetic groups and Crystallographic groups.

The work on arithmetic groups $SL(2, \mathbb{Z}[1/m])$ was motivated by a question of Kevin Hutchinson. In his paper [20], Hutchinson stated that he is interested in the precise structure of $H_3(SL_2(\mathbb{Q}), \mathbb{Z})$ and the answer is still unknown. This chapter is written not to solve Hutchinson's question but to use it as motivation since the class of integral rings $\mathbb{Z}[1/m]$ is related to the field of rational numbers $\mathbb{Q}$ via the inclusions

$$\mathbb{Z}[1/2] \subset \mathbb{Z}[1/2.3] \subset \mathbb{Z}[1/2.3.5] \subset \cdots \subset \mathbb{Z}[1/2.3.5...] = \mathbb{Q}.$$

Hutchinson uses our calculation of $H_n(\mathbb{Z}[1/6])$ in his recent preprint [21] in order to prove some of his results. The work on arithmetic groups is also a generalization of the paper "On the cohomology of $SL(2, \mathbb{Z}[1/p])$" by A. Adem and N. Naffah [1] in which the authors compute integral cohomology for primes p only.
Our main goal in relation to arithmetic groups is to present a new method for calculating the group homology and cohomology of the arithmetic groups $SL(2, \mathbb{Z}[1/m])$.

Crystallographic groups have been studied for many years and are still an active topic of research. Some recently published works in the area are [2, 7, 9]. The computational algebra system GAP [16] provides two methods for calculating the group homology of crystallographic groups.
One of those two methods is developed and implemented by Graham Ellis and presented as a function in HAP [11]. His method is based on a technique of C. T. C Wall [31] involving group extensions $T \to G \to P$. The technique can be applied to a crystallographic group G with translation subgroup T given some methods for computing the free resolutions R^T_* and R^P_*. A free $\mathbb{Z}G$-resolution can be constructed by combining those resolutions. This method works for all crystallographic groups and yields cohomology rings for such groups. However, for some cases, it produces big $\mathbb{Z}G$-resolutions or doesn't stop after one hour of running and in some cases give an error of exceeding permitted memory.
Another method is provided by Marc Röder [26]. His algorithm uses convex hull computations to construct a fundamental domain for a Bieberbach group and pro-

duces a finite regular CW-space for such a group. The disadvantages are that Röder considers only Biebebach groups and uses convex hull computations that can be expensive. Moreover, for some groups, the fundamental domain produced by this method may have a complicated shape for which we have no method for finding a contracting homotopy on the corresponding cellular chain complexes; consequently the method does not compute the cohomology ring structure for such groups.
In the light of the disadvantages of the above methods, we are interested in finding a new method which could be faster than Ellis and Wall's method and more general than Röder's method. We limit our goal to groups that admit a cubical fundamental region. The reasons are that: we observe that a high proportion of low-dimensional crystallographic groups admit such a cubical fundamental region; and we can easily give explicit formulas for a contracting homotopy on the resulting cellular chain complex. Furthermore, by subdividing the cubical fundamental region, we could obtain the Bredon homology, though it is not included in the thesis.
Our main goals for crystallographic groups are to introduce: (i) new algorithms which attempt to calculate the group (co)homology of certain crystallographic groups; (ii) a method for computing the cohomology ring structures for those groups where the algorithms of (i) are successful.

1.3 Review of necessary background material

This section recalls basic definitions and concepts needed in the thesis. The material is standard and taken from [19, 25, 18] with little or no modification.

1.3.1 CW-spaces

This section gives a brief review of CW-spaces. We use the following notation for the *closed unit n-ball* , the *open unit n-ball* and the *unit* $(n-1)$*-sphere*

$$D^n = \{x \in \mathbb{R}^n : \quad \| x \| \leq 1\},$$

$$int(D^n) = \{x \in \mathbb{R}^n : \quad \| x \| < 1\},$$

$$S^{n-1} = \{x \in \mathbb{R}^n : \quad \| x \| = 1\},$$

where $\| \, . \, \|$ is the *standard norm*, $\| (x_1, x_2, \ldots, x_n) \| = \sqrt{x_1^2 + x_2^2 + \ldots + x_n^2}$.

Definition 1.3.1. [18] An *n-cell* is a topological space homeomorphic to the open n-disk $int(D^n)$. A *cell* is a space which is an n-cell for some $n \geq 0$. We say the *dimension* of such a cell is n.

Definition 1.3.2. [18] A *cell-decomposition* (or *cell-structure*) of a topological space X is a family $\mathcal{E} = \{e_\alpha | \alpha \in I\}$ of subspaces of X such that each e_α is a cell and

$$X = \coprod_{\alpha \in I} e_\alpha$$

(disjoint union of sets). The *n-skeleton* of X is the subspace

$$X^n = \coprod_{\alpha \in I, dim(e_\alpha) \leq n} e_\alpha$$

Definition 1.3.3. [18] A pair $(X, \mathcal{E})$ consisting of a Hausdorff space X and a cell-decomposition $\mathcal{E}$ of X is called a *CW-space* if the following three axioms are satisfied:

- **Axiom 1:** (*Characteristic Maps*) For each n-cell $e \in \mathcal{E}$ there is a map $\varphi_e : D^n \to X$ restricting to a homeomorphism $\varphi_e|_{int(D^n)} : int(D^n) \to e$ and taking S^{n-1} into X^{n-1} .
- **Axiom 2:** (*Closure Finiteness*) For any cell $e \in \mathcal{E}$ the closure $\bar{e}$ intersects only a finite number of other cells in $\mathcal{E}$.
- **Axiom 3:** (*Weak Topology*) A subset $A \subseteq X$ is closed iff $A \cap \bar{e}$ is closed in X for each $e \in \mathcal{E}$.

The restrictions $\varphi_e|_{\partial D_n}$ are called the *attaching maps*.

Notice that we can recover X (up to homeomorphism) from a knowledge of X^0 and the attaching maps. The recovery is described in [19] as follows:

(1) Start with a discrete set X^0, whose points are regarded as 0-cells.

(2) Inductively, form the n-skeleton X^n from X^{n-1} by attaching n-cells e^n_α via maps $\varphi_\alpha : S^{n-1} \to X^{n-1}$. This means that X^n is the quotient space of the disjoint union $X^{n-1} \coprod_\alpha D^n_\alpha$ of X^{n-1} with a collection of n-disks D^n_α under the identifications $x \sim \varphi_\alpha(x)$ for $x \in \partial D^n_\alpha$. Thus as a set $X^n = X^{n-1} \coprod_\alpha e^n_\alpha$ where each e^n_α is an open n-cell.

(3) One can either stop this inductive process at a finite stage, setting $X = X^n$ for some $n < \infty$, or one can continue indefinitely, setting $X = \bigcup_n X^n$. In the latter case X is given the *weak topology*: A set $A \subset X$ is open (or closed) iff $A \cap X^n$ is open (or closed) in X^n for each n.

So we have,

$$X^0 \subset X^1 \subset X^2 \subset ... \subset X^n \subset ...$$

If there exist an integer n such that $X^n = X$ then X is called *finite dimensional*.

Definition 1.3.4. [19] A CW-space is said to be *regular* if all its attaching maps are homeomorphisms.

Definition 1.3.5. [17] A map $f : X \to Y$ between CW-spaces is *cellular* if it satisfies $f(X^n) \subseteq Y^n$ for all n.

Definition 1.3.6. [18] A *CW-subspace* of a CW-space X is a union Y of cells in X such that the closure of each cell is also contained in Y; therefore, it is also a CW-space.

In other words, given a CW-space $X = \bigcup_{\alpha \in I} e_\alpha$ and subset $I' \subset I$, the union $Y = \bigcup_{\alpha \in I'} e_\alpha$ is a CW-subspace of X if $\bar{e}_\alpha \subset Y$ for all $\alpha \in I'$.

Thus X^n is a CW-subspace of X, for every n.

Let X be a CW-space and Y be a CW-subspace of X. We say that (X, Y) is a *CW-pair*. For an arbitrary CW-space X, we have CW-pairs (X, X^n) for all n and similarly (X^n, X^m) for $n \geq m$.

Proposition 1.3.1. [17] *Let (X, Y) be a CW-pair. Then the quotient space X/Y can be turned in a CW-space such that the quotient map $X \to X/Y$ is cellular.*

Definition 1.3.7. [19] A *G-space* is a CW-space X together with an action of G on X which permutes cells. That means, for each $g \in G$ a homeomorphism $x \to gx$ of X such that the image $g\sigma$ of any cell σ is again a cell.

1.3.2 Cellular chain complexes

Let R be a ring. We are mainly interested in the integers $R = \mathbb{Z}$ and the group ring $R = \mathbb{Z}G$, where G is a group.

Definition 1.3.8. [32] A *chain complex* $C_* = (C_n, d_n)_{n\in\mathbb{N}}$ of R-modules is a sequence of homomorphisms of R-modules

$$C_* : \quad \cdots \xrightarrow{d_{n+2}} C_{n+1} \xrightarrow{d_{n+1}} C_n \xrightarrow{d_n} C_{n-1} \xrightarrow{d_{n-1}} \cdots \xrightarrow{d_2} C_1 \xrightarrow{d_1} C_0 \xrightarrow{d_0} 0$$

such that $d_n d_{n+1} = 0$ for all n. The chain complex C_* is called an *exact sequence* if $\operatorname{Im} d_{n+1} = \operatorname{Ker} d_n$ for all n.

We say that the chain complex is of *length* n if $C_k = 0$ for $k > n$ and $C_n \neq 0$. We refer to the homomorphism d_n as a *boundary homomorphism*. The module C_k is said to be of *degree* k.

We apply the *Hom* funtor to a chain complex C_* by replacing C_n by $C^n = \operatorname{Hom}(C_n, R)$ and replacing d_n by $d^n = \operatorname{Hom}(d_n, R) : C^n \to C^{n+1}$ to obtain the *induced cochain complex*

$$\operatorname{Hom}(C_*, R) : \quad 0 \longrightarrow C^0 \xrightarrow{d^0} C^1 \xrightarrow{d^1} \cdots \xrightarrow{d^{n-2}} C^{n-1} \xrightarrow{d^{n-1}} \cdots$$

Elements in $\operatorname{Ker} d^n$, $\operatorname{Im} d^n$ are called *cocycles* and *coboundaries*.

Definition 1.3.9. [32] The k-dimensional homology of a chain complex C_* over R is the $R-$module

$$H_k(C_*) = \operatorname{Ker} d_k / \operatorname{Im} d_{k+1}.$$

And the k-dimensional cohomology of C_* is

$$H^k(C_*) = H_k(\operatorname{Hom}_R(C_*, R))$$

Suppose that X is a CW-space.

Definition 1.3.10. [19] The *cellular chain complex* $C_*(X)$ is the chain complex with $C_k(X)$ the free abelian group generated by all k-cells in X

$$C_k(X) = \bigoplus_{i=1}^{b_k} \mathbb{Z} e_i^k$$

where b_k is the number of k-cells; and where the boundary map is defined by using singular homology. The definition of the boundary is slightly complicated and we will not described it here. See [19] for the standard definition.

For the purpose of this thesis, we are mainly interested in regular CW-spaces. In this case, we use a practical construction for the cellular chain complex based on the following proposition.

Proposition 1.3.2. [24] *Let X be a regular CW-space and let C_k be the free abelian group formally generated by the k-cells of X. Let $d'_k : C_k \to C_{k-1}$ be any sequence of homomorphisms satisfying*

(i) $d'_k d'_{k+1} = 0$ *for any* $k > 0$;

(ii) $d'_k(e^k_\lambda) = \sum_{e^{k-1}_\mu \in X} \epsilon^k_{\lambda,\mu} e^{k-1}_\mu$ *where* e^{k-1}_μ *ranges over all '$(k-1)$-cells of X and*

$$\epsilon^k_{\lambda,\mu} = \begin{cases} \pm 1 & \text{if } e^{k-1}_\mu \text{ lies in the boundary of } e^k_\lambda, \\ 0 & \text{otherwise;} \end{cases}$$

$\epsilon^k_{\lambda,\mu} = [e^k_\lambda, e^{k-1}_\mu]$ *is called incident number.*

(iii) $\epsilon^1_{\lambda,\mu}\epsilon^1_{\lambda,\mu'} = -1$ *whenever vertices e^0_μ, $e^0_{\mu'}$ are incident with a common 1-cell e^1_λ.*

Then the chain complex (C_*, d'_*) and the chain complex $C_*(X)$ of Definition 1.3.10 have isomorphic homology $H_n(C_*) \cong H_n(C'_*)$, for $n \geq 0$.

Definition 1.3.11. [23] The homology groups of a CW-space X are defined to be the homology groups of the cellular chain complex $C_*(X)$.

1.3.3 (Co)homology of groups and ring structure

This section recalls details on homology and cohomology of group via topology. The main reference of this section is K. S. Brown's book [6].

Definition 1.3.12. [6] Let R be a ring and M a (left) R-module. A *resolution* of M is an exact sequence of R-modules

$$\cdots \longrightarrow F_2 \xrightarrow{\partial_2} F_1 \xrightarrow{\partial_1} F_0 \xrightarrow{\epsilon} M \longrightarrow 0$$

If each F_i is projective (free), then this is called a *projective (free) resolution.*

Let G be a group. Then $\mathbb{Z}G$ is the integral group ring of G and $\mathbb{Z}$ can be considered as a $\mathbb{Z}G$-module with trivial G-action.

Definition 1.3.13. [11] A *non-free $\mathbb{Z}G$-resolution* is an exact sequence of $\mathbb{Z}G$-modules

$$\cdots \to A_n \to A_{n-1} \to \cdots \to A_1 \to A_0 \to \mathbb{Z} \to 0$$

in which the modules are not necessarily free. We can represent such a sequence (non-uniquely) by first choosing free modules M_k mapping onto the A_k and then lifting the homomorphisms to a sequence

$$\cdots \to M_n \to M_{n-1} \to \cdots \to M_1 \to M_0 \to \mathbb{Z}$$

of homomorphisms of free $\mathbb{Z}G$-modules. Note that in the lifted sequence homomorphisms will not in general square to zero.

In HAP [11], a *non-free ZG-resolution*

$$\cdots \to M_n \to M_{n-1} \to \cdots \to M_1 \to M_0 \to \mathbb{Z}$$

is represented by a component object R with the following components.

- R!.dimension(k) is a function which returns the $\mathbb{Z}G$-rank of the module M_k.
- R!.boundary(k,j) is a function which returns the image in M_{k-1} of the j-th free generator of M_k.

- R!.homotopy(k,[i,g]) is a function which returns the image in M_{k+1}, under a contracting homotopy $M_k \to M_{k+1}$, of the element [[i,g]] in M_k. (The elements [[i,g]] freely generate M_k as an abelian group.) For most non-free resolutions a contracting homotopy is not constructed, in which case R!.homotopy is set equal to "fail".

- R!.elts is a (partial) list of (possibly duplicate) elements in G. In some cases R!.elts is a pseudo list and not a list.

- R!.group is the group in question (and could be a permutation group, matrix group, finitely presented group etc.).

- R!.stabilizer(k,j) returns the subgroup of G consisting of those elements that fix, up to sign, the j-th free generator of M_k.

- R!.action(k,j,g) is a function which returns $+1$ or -1 according to how the group element Elts[g] acts on the "orientation" of the j-th free generator in dimension k.

- R!.properties is a list of pairs ["name", value] where "name" is a string and value is a numerical or boolean value. Example pairs are: ["length", n] which records that there are n terms in the resolution; "characteristic", p] which records the characteristic of $\mathbb{Z}$; ["reduced", true] which record that M_0 is isomorphic to $\mathbb{Z}G$; ["type","resolution"] which records the type of the object R.

The operation IsHapNonFreeResoluton(R) returns "true" for a non-free resolution.

Definition 1.3.14. [6] Let G be a group and $\varepsilon : F \to \mathbb{Z}$ a projective resolution of $\mathbb{Z}$ over $\mathbb{Z}G$. We define the *n-th integral homology groups* of G by

$$H_n(G, \mathbb{Z}) = H_n(F \otimes_{\mathbb{Z}G} \mathbb{Z})$$

and the *n-th integral cohomology groups* of G by

$$H^n(G, \mathbb{Z}) = H_n(Hom_{\mathbb{Z}G}(F, \mathbb{Z})).$$

If X is a G-space then the action of G on X induces an action of G on the cellular chain complex $C_*(X)$ which thereby becomes a chain complex of $\mathbb{Z}G$-modules. Moreover, the canonical augmentation $\epsilon : C_0(X) \to \mathbb{Z}$ (defined by $\epsilon(v) = 1$ for every 0-cells v of X) is a map of $\mathbb{Z}G$-modules. In this case, we say that X is a *free G-space* if the action of G freely permutes the cells of X; and then each chain module $C_n(X)$ is a free $\mathbb{Z}G$-module with one basis element for every G-orbit of n-cells. We let X/G denote the set whose elements are the orbits $\bar{x} = Gx$ of G on X. Let $\pi : X \to X/G$ denote the natural map taking x into its orbit $\bar{x} = Gx$. Then X/G endowed with the quotient topology is called the *orbit space* of X (with respect to G).

Proposition 1.3.3. [6] *Let X be a contractible free G-space. Then the augmented cellular chain complex of X is a free resolution of $\mathbb{Z}$ over $\mathbb{Z}G$.*

Proposition 1.3.4. [6] *Let X be a free G-space. Then*

$$C_*(X/G) \approx C_*(X) \otimes_{\mathbb{Z}G} \mathbb{Z}$$

Theorem 1.3.5. *Let G be a group. If there is a contractible free G-space X then*

$$H_n(G, \mathbb{Z}) = H_n(C_*(X) \otimes_{\mathbb{Z}G} \mathbb{Z}) = H_n(C_*(X/G))$$

and

$$H^n(G, \mathbb{Z}) = H_n(Hom_{\mathbb{Z}G}(C_*(X), \mathbb{Z})) = H^n(C_*(X/G), \mathbb{Z})$$

Theorem 1.3.5 follows directly from the Proposition 1.3.3 and Proposition 1.3.4.

The cup product in cohomology is an operation which turns the cohomology groups of a group G into a graded ring: $H^*(G, R) = \bigoplus_{k \geq 0} H^k(G, R)$. Details on cup products can be found in [6]. In this section, we introduce a standard construction of the cup product which is implemented in HAP [11].

Definition 1.3.15. [6] Let R^G_* be a $\mathbb{Z}G$-resolution of $\mathbb{Z}$. A *contracting homotopy* on R^G_* is a family of $\mathbb{Z}$-linear homomorphisms $h_n : R^G_n \to R^G_{n+1}$ satisfying

$$h_{n-1}\delta_n + \delta_{n+1}h_n = 1_{R^G_n}$$

for all $n \geq 0$ with $h_{-1} = 0$ and $\delta_0 = 0$.

For any group G there is a bilinear mapping:

$$H^p(G,\mathbb{Z}) \oplus H^q(G,\mathbb{Z}) \to H^{p+q}(G,\mathbb{Z}), (u,v) \mapsto uv$$

called the *cup product*. The product is associative, and $uv = (-1)^{pq}vu$. The cup product gives a multiplication on the direct sum of the cohomology groups $H^*(G,\mathbb{Z}) = \oplus_{k\in\mathbb{N}} H^k(G,\mathbb{Z})$. This multiplication turns $H^*(G;\mathbb{Z})$ into a ring and now is called *integral cohomology ring* of G. The construction of the cup product is as follows. The cohomology class $u \in H^p(G,\mathbb{Z})$ is represented by a cocycle $\underline{u} : R_p^G \to \mathbb{Z}$ which induces a chain mapping $\underline{u}_n : R_n^G \to R_{n-p}^G$ (for $n > p-1$) by recursion using the contracting homotopy as the diagram below:

$$\begin{array}{ccccccccccccc}
\cdots\cdots & R^G_{n-p} & \overset{h_{n-p-1}}{\longleftarrow\!\!\!\longrightarrow} & R^G_{n-p-1} & \overset{h_{n-p-2}}{\longleftarrow\!\!\!\longrightarrow} & R^G_{n-p-2} & \overset{h_1}{\longleftarrow\!\!\!\cdots\!\!\!\longrightarrow} & R^G_1 & \overset{h_0}{\longleftarrow\!\!\!\longrightarrow} & R^G_0 & \overset{h_{-1}}{\longleftarrow\!\!\!\longrightarrow} & \mathbb{Z} \\
 & \uparrow{\scriptstyle \underline{u}_n} & & \uparrow{\scriptstyle \underline{u}_{n-1}} & & \uparrow{\scriptstyle \underline{u}_{n-2}} & & \uparrow{\scriptstyle \underline{u}_{p+1}} & & \uparrow{\scriptstyle \underline{u}_p} & & \uparrow{\scriptstyle 1_{\mathbb{Z}}} \\
\cdots\cdots\to & R^G_n & \xrightarrow[\delta_n]{} & R^G_{n-1} & \xrightarrow[\delta_{n-1}]{} & R^G_{n-2} & \cdots\to & R^G_{p+1} & \xrightarrow[\delta_{p+1}]{} & R^G_p & \xrightarrow[\underline{u}]{} & \mathbb{Z}
\end{array}$$

The composition of $\underline{u}_{p+q}$ with the cocycle $\underline{v} : R_q^G \to \mathbb{Z}$ is a cocycle representing a cohomology class uv in $H_{p+q}(G,\mathbb{Z})$.

Chapter 2

Vector fields and perturbations

In this chapter, we prove the following useful computational result

Theorem 2.0.6. *Let $G = SL_2(\mathbb{Z})$ be the group of 2×2 integer matrices of determinant 1. There is a free $\mathbb{Z}G$-resolution R_*^G of $\mathbb{Z}$ with precisely two free generators in each degree greater than 1 and with boundary homomorphism and contracting homotopy explicitly described below.*

To prove this theorem we need

(i) a perturbation technique of Wall [31] and

(ii) a discrete vector field on a contractible CW-space X with (non-free) action of $SL_2(\mathbb{Z})$.

We use Theorem 2.0.6 to obtain an algorithm (Algorithm 2.3.2) that returns the boundary homomorphism $d : R_*^{SL_2(\mathbb{Z})} \to R_*^{SL_2(\mathbb{Z})}$ and contracting homotopy $h : R_*^{SL_2(\mathbb{Z})} \to R_*^{SL_2(\mathbb{Z})}$

2.1 Wall's technique

Let G be a discrete group acting on a contractible CW-space X in such a way that the action permutes cells. Then X is a G-space. Each cell e in X has stabilizer group $G_e = \{g \in G : ge = e\}$ whose elements need not stabilize e point-wise. By G_e stabilizes e point-wisely, we mean $gx = x$ for all $x \in e$ and $g \in G_e$. Let $[e]$ denote the equivalence class of cells in the orbit of e under the action of G, and let $Orb(n)$ denote the set of equivalence classes of n-dimensional cells. The cellular chain complex $C_*(X)$ of X is an exact sequence of $\mathbb{Z}G$-modules with the integral homology group $H_0(C_*(X), \mathbb{Z}) = \mathbb{Z}$ and with

$$C_p(X) = \bigoplus_{[e] \in Orb(p)} \mathbb{Z}G \otimes_{\mathbb{Z}G_e} \mathbb{Z}^e$$

where each $\mathbb{Z}^e$ is a copy of the integers endowed with an "orientation" action of G_e. The chain complex $C_*(X)$ is a $\mathbb{Z}G$-resolution of $\mathbb{Z}$ but generally not free. The

method below gives a construction that inputs the non-free $\mathbb{Z}G$-resolution $C_*(X)$ together with free $\mathbb{Z}G_e$-resolutions $R_*^{G_e}$, and outputs a free $\mathbb{Z}G$-resolution R_*^G of $\mathbb{Z}$.

Suppose that for each class $[e]$ we are given a free $\mathbb{Z}G_e$-resolution $R_*^{G_e}$ of $\mathbb{Z}$. By defining

$$F_{p,q} := \bigoplus_{[e]\in Orb(p)} \mathbb{Z}G \otimes_{\mathbb{Z}G_e} R_q^{G_e}$$

we obtain a free $\mathbb{Z}G$-resolution

$$F_{p,*} : \cdots \to F_{p,q} \to \cdots \to F_{p,2} \to F_{p,1} \to F_{p,0}$$

of the module $C_p(X)$.

Let

$$R_n^G = \oplus_{p+q=n} F_{p,q}. \tag{2.1}$$

Then R_*^G is a free $\mathbb{Z}G$-resolution with respects to a differential constructed in the following propositions:

Proposition 2.1.1. [31] *Let A_{pq} $p, q \geq 0$ be a bigraded family of free R-modules for some ring R. Suppose that there are R-homomorphisms $d_0 : A_{p,q} \to A_{p,q-1}$ such that $(A_{p,*}, d_0)$ is an acyclic chain complex for each p. Set $C_p = H_0(A_{p,*}, d_0)$ and suppose further that there are R-homomorphisms $\delta : C_p \to C_{p-1}$ for which (C_*, δ) is an acyclic chain complex. Then there exist R-homomorphisms $d_k : A_{p,q} \to A_{p-k,q+k-1}$ $(k \geq 1,\ p > k)$ such that*
(i) d_i induces δ;
(ii) $\sum_{i=0}^{k} d_i d_{k-i} = 0$ for each k (where d_k is interpreted as zero if $q = k = 0$ or if $p < k$). Hence $d = d_0 + d_1 + \cdots : \oplus_{p+q=n} A_{pq} \to \oplus_{p+q=n-1} A_{pq}$ is a differential.

Proposition 2.1.2. [31, 12] *Let the family of modules $A_{*,*}$ and module homomorphisms $d_0 : A_{p,q} \to A_{p,q-1}$ and $\delta : C_p \to C_{p-1}$ be as in Proposition 2.1.1. Suppose that there exist abelian group homomorphisms $h_0 : A_{p,q} \to A_{p,q+1}$ such that $d_0 h_0 d_0(x) = d_0(x)$ for all $x \in A_{p,q+1}$. Then we can construct the module homomorphisms $d_k : A_{p,q} \to A_{p-k,q+k-1}$ and the differential d of Proposition 2.1.1 by first lifting δ to $d_1 : A_{p,0} \to A_{p-1,0}$ and then recursively defining $d_k = -h_0(\sum_{i=1}^{k} d_i d_{k-i})$ on free generators of the module $A_{p,q}$. Furthermore, if $H_0(C_*) \cong \mathbb{Z}$ and each C_p is free abelian,*

we can construct abelian group homomorphisms $h : \oplus_{p+q=n} A_{p,q} \to \oplus_{p+q=n+1} A_{p,q}$ *satisfying* $dhd(x) = d(x)$ *by setting* $h(a_{p,q}) = h_0(a_{p,q}) - hd^+h_0(a_{p,q}) + \varepsilon(a_{p,q})$ *for free generators* $a_{p,q}$ *of the abelian group* $A_{p,q}$. *Here* $d^+ = \sum_{i=1}^{p} d_i$ *and, for* $q \geq 1$, $\varepsilon = 0$. *For* $q = 0$ *we define* $\varepsilon = h_1 - h_0d^+h_1 + hd^+h_1 + hd^+h_0d^+h_1$ *where* $h_1 : A_{p,0} \to A_{p+1,0}$ *is an abelian group homomorphism induced by a contracting homotopy on* C_*.

The above construction can be implemented in the form of the following algorithm. (See Appendix 5.1 for the specification of the input/output data types.)

Algorithm 2.1.1 Converting a non-free $\mathbb{Z}G$-resolution to a free $\mathbb{Z}G$-resolution

Input:

- a non-free $\mathbb{Z}G$-resolution C_* of $\mathbb{Z}$ together with free $\mathbb{Z}G_e$-resolutions of $\mathbb{Z}$ with contracting homotopy for all stabilizers G_e, and
- an integer $n \geq 0$.

Output: The first $n+1$ terms of a free $\mathbb{Z}G$-resolution R_*^G of $\mathbb{Z}$. Moreover, if the input C_* is endowed with a contracting homotopy then the output is returned together with a contracting homotopy.

Procedure:

1: The *nth*-term R_n^G is computed by the formula 2.1;
2: The boundary is constructed by using Proposition 2.1.1;
3: If C_* has a contracting homotopy then the contracting homotopy on R_*^G is constructed by using Proposition 2.1.2.

Algorithm 2.1.1 has already been implemented in the HAP package but without a contracting homotopy. The contracting homotopy implementation is new and has been implemented as part of this thesis work.

2.2 Discrete vector fields

In this section we recall the concept of *discrete vector field* and introduce theoretical and algorithmic methods to construct a *contracting homotopy* on a CW-space as an application of a discrete vector field. Material is mainly taken from [15, 14, 27, 22].

Let X be a CW-space. Whenever a cell $\tau \in X$ is attached to a cell σ, we call σ

is a *face* of τ; a face of codimension 1 is called a *facet*. If the attaching map φ_τ of τ restricts to a homeomorphism on the preimage $\varphi_\tau^{-1}(\sigma)$ and the closure of $\varphi_\tau^{-1}(\sigma)$ is a closed ball, then σ is a *regular facet* of τ. If all facets are regular then X is a regular CW-space.

Definition 2.2.1. [15, 27] Let $(C_*(X), d, \beta)$ be the cellular chain complex of a CW-space X where $d_n : C_n(X) \to C_{n-1}(X)$ is the boundary maps and β_n is the cell basis for $C_n(X)$. If σ^k and τ^{k+1} are respectively a k-cell and $(k+1)$-cell, then the *incidence number* $\varepsilon(\sigma^k, \tau^{k+1})$ is the coefficient of σ^k in the boundary $d\tau$. This incidence number is non-null if and only if σ^k is a face of τ^{k+1}; it is ± 1 if and only if σ^k is a regular face of τ

Definition 2.2.2. [15, 22] A discrete vector field V on a regular CW-space X is a collection of arrows $s \to t$ where

- s, t are cells and any cell is involved in at most one arrow,
- $dim(t) = dim(s) + 1$,
- s lies in the boundary of t.

An arrow $s \to t$ involves two cells s and t, where s is called the *source* and t the *target* of such arrow. We also write $t = V(s)$ and $s = V^{-1}(t)$. A cell which is not involved in any arrow is called a *critical cell*. A cell basis β_n is canonically divided by the vector field V into three components $\beta_n = \beta_n^t \cup \beta_n^s \cup \beta_n^c$ where β_n^t (respectively β_n^s, β_n^c) is made of the target (respectively source, critical) n-cells.

Definition 2.2.3. [15, 27] Let V be a discrete vector field on a regular CW-space X. Then a V-path γ of length r from a k-dimensional cell σ^k to a k-dimensional cell τ^k is a sequence of arrows $((s_0 \to t_0), (s_1 \to t_1), \ldots, (s_{r-1} \to t_{r-1})) \subset V$ satisfying:

- $\sigma^k = s_0$ and τ^k is in the boundary of t_{r-1},
- s_{i+1} lies in the boundary of t_i for all $0 \leq i < r-1$.

A V-path γ is said to be a cycle if s_0 lies in the boundary of t_{r-1}.

Definition 2.2.4. [15] A discrete vector field is *admissible* if there is no cycle and for every source cell s, the length of any path starting from s is bounded by a fixed integer $\lambda(s)$, i.e. there is no path of infinite length.

Theorem 2.2.1. [15, 33] *If X is a regular CW-space with admissible discrete vector field then there is a homotopy equivalence*

$$X \simeq Y$$

where Y is a CW-space whose cells are in one-one correspondence with the critical cells of X .

Proof of Theorem 2.2.1 can be found in [4] .

We now consider a very special case when the CW-space X is contractible, meaning that it can be retracted to a single point. If there is a discrete vector field describing such a retraction, we shall call it a *contracting vector field*.

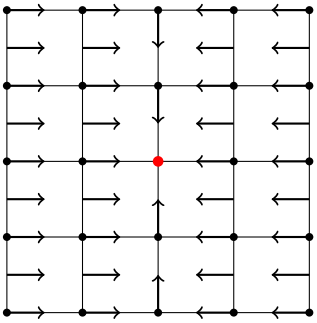


Figure 2.1: DVF on cubical CW-space

Figure 2.1 is an example of a contracting discrete vector field on a 2-dimensional CW-space with cubical cell structure.

The idea of a formula for a contracting homotopy based on discrete vector field was introduced in [22] and [15]. The practical version of the formula is given in [13] as follows

Theorem 2.2.2. [13] *Let $C(X) = (C_*(X), d_*, V)$ be the cellular chain complex of a contractible CW-space X together with a contracting discrete vector field V. Then the contracting homotopy on $C_*(X)$ can be defined as the sum of all maximal V-paths*

starting from any n-cell σ^n,

$$h_n(\sigma^n) = \begin{cases} 0 & \text{if } \sigma^n \text{ is not a source} \\ \sum \sigma_i^{n+1} & \partial_{n+1}(\sum \sigma_i^{n+1}) \text{ contains just the one} \\ & \text{source } \sigma^n \text{ of dimension } n \end{cases}$$

One can implement this contraction h_* by using the equivalent recursive formula as follows:

$$h_n(\sigma^n) = \varepsilon(\sigma^n, V(\sigma^n)) \left(V(\sigma^n) - \sum_{\sigma' \in \beta_n^s - \sigma^n} \varepsilon(\sigma', V(\sigma^n)) h_n(\sigma') \right)$$

2.3 Proof of theorem 2.0.6 and related algorithm

Consider the matrices

$$S = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}, \; T = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}.$$

Note that $S^4 = (ST)^6 = 1$ and that the group $SL_2(\mathbb{Z})$ is generated by S and T. We let M denote the group with presentation $\langle s, u \mid s^2 = u^3 = 1 \rangle$. Following [8] we construct the cubic tree $\mathcal{T}$ by taking the left cosets of $U = \langle u \rangle$ in M as vertices, and joining cosets xU and yU by an edge if, and only if, $x^{-1}y \in UsU$. Thus the vertex U is joined to sU, usU and u^2sU. The vertices of this tree are in one-to-one correspondence with all reduced words in s, u and u^2 that, apart from the identity, end in s. We say that a sequence of vertices x_0U, x_1U, ..., x_nU is a *rooted path* if $x_0 = 1$ and there is an edge between x_iU and $x_{i+1}U$ for $0 \leq i \leq n-1$. A rooted path is described by a reduced word in s, u and u^2 ending in s.

The group M can be realized as the modular group of transformations $z \mapsto (mz + n)/(pz + q)$, m, n, p, q integers with $mq - np = 1$, of the upper half complex plane $\mathbb{H} = \{x + \mathbf{i}y \in \mathbb{C} : y > 0\}$ which is generated by the two transformations $a(z) =$

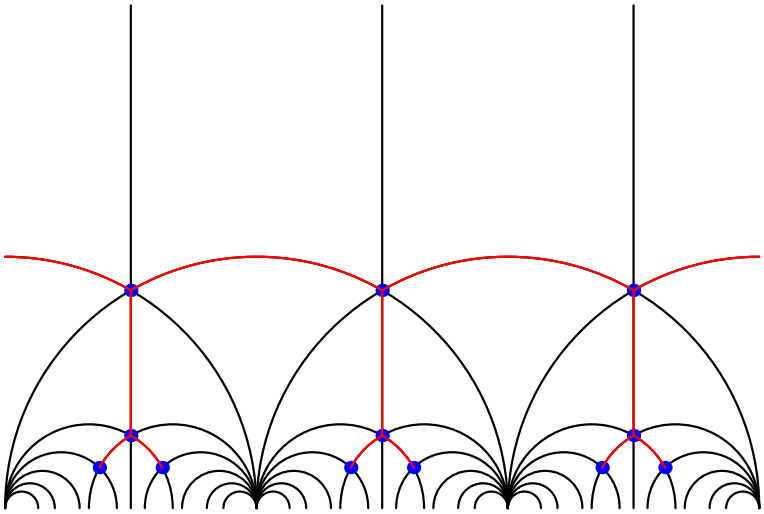


Figure 2.2: Portion of the cubic tree indicated in red

$-1/z$ and $u(z) = z + 1$. The cubic tree can thus be embedded in $\mathbb{H}$. Figure 2.3 illustrates this embedding by showing a portion of the cubic tree in red against the tessellation of $\mathbb{H}$ by triangular fundamental domains for the modular group. The surjection $SL_2(\mathbb{Z}) \to M, S \mapsto s, T \mapsto s^{-1}u$ and left multiplication in M yield an action of $SL_2(\mathbb{Z})$ on the cubic tree $\mathcal{T}$. Under this action there is one orbit of vertices and one orbit of edges. The stabilizer group of any vertex is conjugate to $C_6 = \langle ST \rangle$. The stabilizer group of any edge is conjugate to $C_4 = \langle S \rangle$. The cellular chain complex $C_*(\mathcal{T})$ thus has the form

$$C_*(\mathcal{T}): \quad \mathbb{Z}[SL_2(\mathbb{Z})] \otimes_{\mathbb{Z}C_4} \mathbb{Z}^e \xrightarrow{\delta} \mathbb{Z}[SL_2(\mathbb{Z})] \otimes_{\mathbb{Z}C_6} \mathbb{Z} \tag{2.2}$$

where C_6 acts trivially on $\mathbb{Z}$ and where $\mathbb{Z}^e$ denotes the integers with non-trivial action of C_4. The boundary homomorphism δ is induced by the equivariant homomorphism of free modules

$$\mathbb{Z}[SL_2(\mathbb{Z})] \longrightarrow \mathbb{Z}[SL_2(\mathbb{Z})], 1 \mapsto T - 1 \ .$$

Let $R_*^{C_m}$ denote the periodic free $\mathbb{Z}C_m$ -resolution of $\mathbb{Z}$ with one free generator in each degree. We define the free $\mathbb{Z}G$-modules

$$A_{0,q} = R_q^{C_6} \otimes_{\mathbb{Z}C_6} \mathbb{Z}G, \qquad A_{1,q} = (R_q^{C_4} \otimes_{\mathbb{Z}} \mathbb{Z}^e) \otimes_{\mathbb{Z}C_4} \mathbb{Z}G$$

for $q \geq 0$. In this example the first assertion of Proposition 2.1.2 is simply that δ

induces a chain map

$$\begin{array}{ccc} \vdots & & \vdots \\ \downarrow & & \downarrow \\ A_{1,2} & \xrightarrow{d_2} & A_{0,2} \\ \downarrow{\scriptstyle d_2} & & \downarrow{\scriptstyle d_1} \\ A_{1,1} & \xrightarrow{d_1} & A_{0,1} \\ \downarrow{\scriptstyle d_1} & & \downarrow{\scriptstyle d_0} \\ A_{1,0} & \xrightarrow{d_0} & A_{0,0} \end{array}$$

and that there is a chain complex

$$\cdots \xrightarrow{d_4} A_{1,2} \otimes A_{0,3} \xrightarrow{d_3} A_{1,1} \otimes A_{0,2} \xrightarrow{d_2} A_{1,0} \otimes A_{0,1} \xrightarrow{d_1} A_{0,0} \tag{2.3}$$

of $\mathbb{Z}G$-modules. Any contracting homotopy on the resolution $R_*^{C_6}$ extends to a contracting homotopy on the complex $\cdots \to A_{0,2} \to A_{0,1} \to A_{0,0}$. Similary any contracting homotopy on $R_*^{C_4}$ extends to one on $\cdots \to A_{1,2} \to A_{1,1} \to A_{1,0}$. Furthermore, the chain complex $C_*(\mathcal{T})$ is acylic and thus admits a contracting homotopy. The second assertion of Proposition 2.1.2 is a formula for a contracting homotopy on the chain complex 2.3. The existence of this latter contracting homotopy implies that 2.3 is exact and thus a free $\mathbb{Z}G$-resolution of $\mathbb{Z}$.

In order to implement a contracting homotopy on 2.3 we need to provide a contracting homotopy on $C_*(\mathcal{T})$. We do this by specifying a discrete vector field on the cubic tree with just one critical vertex and no critical edge. Above, we have identified the vertices of $\mathcal{T}$ with the left cosets of $U \leq M$. But since we are constructing a $\mathbb{Z}G$-resolution for $G = SL(2,\mathbb{Z})$ it is more practical to identify the vertices of $\mathcal{T}$ with the left cosets of the group $\langle ST \rangle$ in G. Each left coset consists of six matrices. We use the following two rules to construct a discrete vector field on $\mathcal{T}$. Suppose that a vertex $A\langle ST \rangle$ of $\mathcal{T}$ is represented by a matrix

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}.$$

1. If $A \in \langle ST \rangle$ then the vertex $A\langle ST \rangle$ is critical.

2. Otherwise, the vector field has an arrow from vertex $A\langle ST \rangle$ to the edge con-

taining this vertex and the vertex $B\langle ST\rangle$ with matrix B defined by

$$B = \begin{cases} SA & \text{if } |a| < |c| \text{ or } a = c, \\ T^{-1}A & \text{if } Sign(a) = Sign(c) \text{ and } |a| = |c|, \\ TA & \text{if } Sign(a) = -Sign(c) \text{ and } |a| \geq |c|. \end{cases}$$

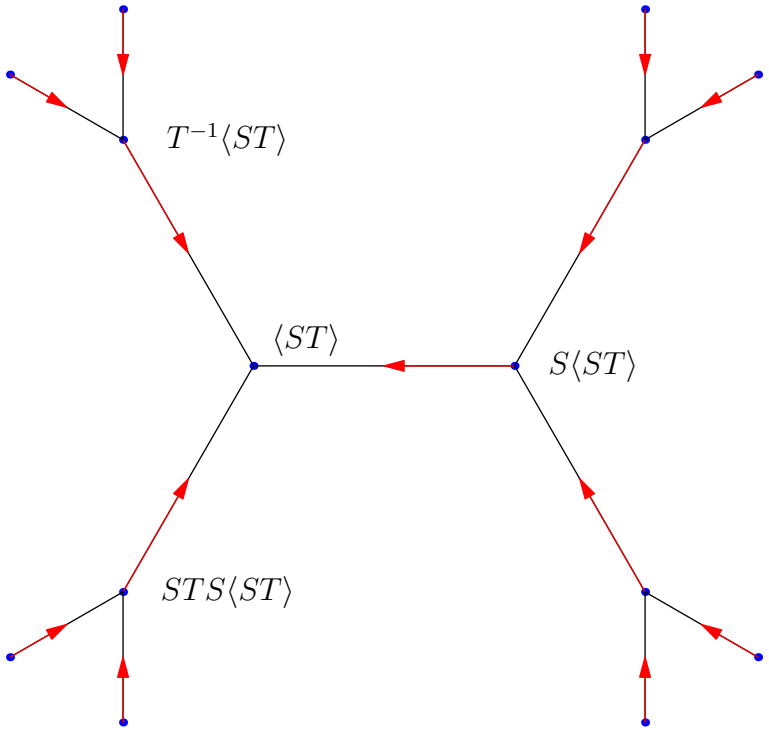


Figure 2.3: Discrete vector field based on Rule 1 and 2.

Rules 1 and 2 provide a recursive algorithm for expressing an arbitrary matrix $A \in G$ as a product of S, T, T^{-1} and $-I$ where I is the identity matrix. For instance, the matrix

$$A = \begin{pmatrix} 3 & 2 \\ -2 & -1 \end{pmatrix}$$

can be expressed as

$$A = T^{-1}\begin{pmatrix} 1 & 1 \\ -2 & -1 \end{pmatrix} = T^{-1}S\begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix} = T^{-1}ST\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$$

$$= T^{-1}STS \begin{pmatrix} -1 & -1 \\ 1 & 0 \end{pmatrix} = T^{-1}STST^{-1} \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} = T^{-1}STST^{-1}S$$

The above construction can be implemented on computer in the form of algorithms as follows:

Algorithm 2.3.1 Decompose elements of $SL_2(\mathbb{Z})$

Input: An element $g \in SL_2(\mathbb{Z})$.

Output: A product of $S,\ T,\ T^{-1}$ and $-I$ equal to g.

Procedure:

1: Follow the method described in rules 1 and 2; and the above example..

Algorithm 2.3.2 Cellular chain complex $C_*(\mathcal{T})$ with $SL_2(\mathbb{Z})$-action

Input: Void.

Output: A cellular chain complex $C_*(\mathcal{T})$ as a non-free resolution, where $\mathcal{T}$ is the cubic tree that $SL_2(\mathbb{Z})$ acts on. The chain complex is computed together with a contracting homotopy.

Procedure:

1: Compute the chain complex $C_*(\mathcal{T})$ by using 2.2.
2: The contracting homotopy on $C_*(\mathcal{T})$ is calculated using Algorithm 2.3.1.

One method of computing the cohomology ring is described in Section 1.3.3 and now is used for finding $H^*(SL_2(\mathbb{Z}), \mathbb{Z})$.

The following GAP session constructs a free $\mathbb{Z}G$-resolution R^G_* of $\mathbb{Z}$ for $G = SL_2(\mathbb{Z})$, up to degree $d = 3$, and uses it to compute a minimal set X that generates $H^*(G, \mathbb{Z})$ up to degree 2. The set X consists of two homogeneous cohomology classes of degrees 0 and 2 respectively.

GAP session

```
gap> R:=ResolutionSL2Z(1,4);
Resolution of length 4 in characteristic 0 for SL(2,Integers) .

gap> Cohomology(HomToIntegers(R),1);
[  ]
gap> Cohomology(HomToIntegers(R),2);
[ 12 ]
gap> Cohomology(HomToIntegers(R),3);
[  ]
gap> IntegralCohomologyGenerators(R,0);
[ [ 1 ] ]
gap> IntegralCohomologyGenerators(R,1);
[ [ 0 ] ]
gap> IntegralCohomologyGenerators(R,2);
[ [ 1 ] ]
```

We can check that $R_4 = R_2$ and $d_4 = d_2$. Hence there is a periodic resolution of period 2 for $SL_2(\mathbb{Z})$ that yields

$$H^k(SL_2(\mathbb{Z}), \mathbb{Z}) = \begin{cases} 0, & \text{odd } k \geq 1 \\ \mathbb{Z}_{12}, & \text{even } k \geq 2 \end{cases}$$

Let 1 denote a generator of $H^0(G, \mathbb{Z})$ and hence the identity element of the cohomology ring. Let x^2 be a generator of $H^2(G, \mathbb{Z})$. Since $SL_2(\mathbb{Z})$ is periodic of period 2 then the isomorphism $H^2(G, \mathbb{Z}) \cong H^4(G, \mathbb{Z})$ is induced from cup product by a generator of $H^2(G, \mathbb{Z})$. Hence x^4 is a generator of $H^4(G, \mathbb{Z})$. By induction $x^{2n} = x^2 \cup x^{2n-2}$ is a generator of $H^{2n}(G, \mathbb{Z})$. Since for all $n \in \mathbb{N}$, $H^{2n}(G, \mathbb{Z})$ is generated by a single generator then we have a graded ring isomorphism

$$H^*(SL_2(\mathbb{Z}), \mathbb{Z}) = \mathbb{Z}_{12}[x^2]$$

where x^2 is the generator of $H^2(G, \mathbb{Z})$.

Chapter 3

Homology of $SL_2(\mathbb{Z}[1/m])$

In this chapter, we describe an algorithm for computing finitely many terms of a free $\mathbb{Z}G$-resolution R^G_* of $\mathbb{Z}$ for G a finite index subgroup of $SL_2(\mathbb{Z}[1/m])$. The algorithm is valid, in principle, for any positive integer m and the modules R^G_n are finitely generated in each degree n. An implementation of the algorithm is available in the HAP [11] package for the GAP [16] computational algebra system and has been used to compute $H_n(SL_2(\mathbb{Z}[1/m]), \mathbb{Z})$ for all $m \leq 50, m \neq 30, 42$ and all $n \geq 0$. When $m = 30$ or $m = 42$ the implementation is practical only for $n = 1, 2$. Table 3.1 summarizes these computations.

The homology of $H_n(SL_2(\mathbb{Z}[1/m]), \mathbb{Z})$ is known for all primes $m = p$ by work of Adem and Naffah [1] and these prime cases are thus omitted from Table 3.1. The table also omits cases where m is divisible by the square of a prime as these coincide with square free cases. It is known by work of Williams and Wisner [34] that the homology is a finite group with only 2-torsion and 3-torsion for $n > k + 1$ when $m = p_1 p_2 \dots p_k$ is a product of k primes. In fact, from our algorithm we can prove that the homology is periodic of period 2 in degrees $> k + 1$. (See Proposition 3.2.1 for a proof.)

Each completed row of the table thus describes the integral homology of a group for all degrees n.

The algorithm works inductively by expressing a square free integer m in the form $m = pm'$ with p a prime, and using the decomposition

$$SL_2(\mathbb{Z}[1/m]) \cong SL_2(\mathbb{Z}[1/m']) *_{\Gamma_0(p)} SL_2(\mathbb{Z}[1/m'])$$

as an amalgamated free product of two distinct copies of $SL_2(\mathbb{Z}[1/m'])$ in $SL_2(\mathbb{Z}[1/m])$ over the congruence subgroup $\Gamma_0(p) \subset SL_2(\mathbb{Z}[1/m])$. The definition of $\Gamma_0(p)$ is given below. A special case of a homological perturbation technique given in [12] (which, in turn, is based on a result of Wall [31]) is used to construct a resolution for $SL_2(\mathbb{Z}[1/m])$ from resolutions for $SL_2(\mathbb{Z}[1/m'])$ and $\Gamma_0(p)$. Although our primary interest is for integral coefficients, our implementation of the algorithm in HAP [11] can be used for arbitrary finitely generated coefficient modules.

$m=$	$n=$				
	1	2	3	4	5
6	0	$\mathbb{Z}_2 \oplus \mathbb{Z}$	$\mathbb{Z}_4 \oplus \mathbb{Z}_8 \oplus (\mathbb{Z}_3)^2$	$\mathbb{Z}_2$	$\mathbb{Z}_4 \oplus \mathbb{Z}_8 \oplus (\mathbb{Z}_3)^2$
10	$\mathbb{Z}_3$	$\mathbb{Z}_4 \oplus \mathbb{Z}$	$\mathbb{Z}_2 \oplus \mathbb{Z}_8 \oplus (\mathbb{Z}_3)^4$	$\mathbb{Z}_4$	$\mathbb{Z}_2 \oplus \mathbb{Z}_8 \oplus (\mathbb{Z}_3)^4$
14	$\mathbb{Z}_3$	$\mathbb{Z}_2 \oplus \mathbb{Z}_3 \oplus \mathbb{Z}$	$\mathbb{Z}_4 \oplus \mathbb{Z}_8 \oplus (\mathbb{Z}_3)^2 \oplus \mathbb{Z}^2$	$(\mathbb{Z}_2)^3 \oplus (\mathbb{Z}_3)^2$	$\mathbb{Z}_4 \oplus \mathbb{Z}_8 \oplus (\mathbb{Z}_3)^2$
15	$\mathbb{Z}_4$	$\mathbb{Z}_4 \oplus \mathbb{Z}$	$\mathbb{Z}_2 \oplus (\mathbb{Z}_4)^2 \oplus (\mathbb{Z}_3)^2 \oplus \mathbb{Z}^2$	$\mathbb{Z}_2 \oplus (\mathbb{Z}_4)^2$	$\mathbb{Z}_2 \oplus (\mathbb{Z}_4)^2 \oplus (\mathbb{Z}_3)^2$
21	$\mathbb{Z}_4$	$\mathbb{Z}_2 \oplus \mathbb{Z}_3 \oplus \mathbb{Z}$	$\mathbb{Z}_2 \oplus (\mathbb{Z}_4)^3 \oplus \mathbb{Z}_3 \oplus \mathbb{Z}^2$	$(\mathbb{Z}_2)^3 \oplus \mathbb{Z}_3$	$\mathbb{Z}_2 \oplus (\mathbb{Z}_4)^3 \oplus \mathbb{Z}_3$
22	$\mathbb{Z}_3$	$\mathbb{Z}_2 \oplus \mathbb{Z}_5 \oplus \mathbb{Z}$	$\mathbb{Z}_4 \oplus \mathbb{Z}_8 \oplus (\mathbb{Z}_3)^4$	$\mathbb{Z}_2$	$\mathbb{Z}_4 \oplus \mathbb{Z}_8 \oplus (\mathbb{Z}_3)^4$
26	$\mathbb{Z}_3$	$\mathbb{Z}_4 \oplus \mathbb{Z}_3 \oplus \mathbb{Z}$	$\mathbb{Z}_2 \oplus \mathbb{Z}_8 \oplus (Z_3)^2 \oplus \mathbb{Z}^4$	$(\mathbb{Z}_2)^4 \oplus \mathbb{Z}_4 \oplus (\mathbb{Z}_3)^2$	$\mathbb{Z}_2 \oplus \mathbb{Z}_8 \oplus (\mathbb{Z}_3)^2$
30	0	$\mathbb{Z}_2 \oplus \mathbb{Z}_4 \oplus \mathbb{Z}$	$--$	$--$	$--$
33	$\mathbb{Z}_4$	$\mathbb{Z}_2 \oplus \mathbb{Z}_5 \oplus \mathbb{Z}$	$\mathbb{Z}_2 \oplus (\mathbb{Z}_4)^3 \oplus (\mathbb{Z}_3)^2 \oplus \mathbb{Z}^2$	$(\mathbb{Z}_2)^3$	$\mathbb{Z}_2 \oplus (\mathbb{Z}_4)^3 \oplus (\mathbb{Z}_3)^2$
34	$\mathbb{Z}_3$	$\mathbb{Z}_{16} \oplus \mathbb{Z}$	$\mathbb{Z}_2 \oplus \mathbb{Z}_8 \oplus (\mathbb{Z}_3)^4 \oplus \mathbb{Z}^2$	$(\mathbb{Z}_2)^2 \oplus \mathbb{Z}_4$	$\mathbb{Z}_2 \oplus \mathbb{Z}_8 \oplus (\mathbb{Z}_3)^4$
35	$\mathbb{Z}_4 \oplus \mathbb{Z}_3$	$\mathbb{Z}_4 \oplus \mathbb{Z}_3 \oplus \mathbb{Z}$	$\mathbb{Z}_2 \oplus (\mathbb{Z}_4)^2 \oplus (\mathbb{Z}_3)^2 \oplus \mathbb{Z}^6$	$(\mathbb{Z}_2)^5 \oplus (\mathbb{Z}_4)^2 \oplus (\mathbb{Z}_3)^2$	$\mathbb{Z}_2 \oplus (\mathbb{Z}_4)^2 \oplus (\mathbb{Z}_3)^2$
38	$\mathbb{Z}_3$	$\mathbb{Z}_2 \oplus \mathbb{Z}_9 \oplus \mathbb{Z}$	$\mathbb{Z}_4 \oplus \mathbb{Z}_8 \oplus (\mathbb{Z}_3)^2 \oplus \mathbb{Z}^4$	$(\mathbb{Z}_2)^5 \oplus (\mathbb{Z}_3)^2$	$\mathbb{Z}_4 \oplus \mathbb{Z}_8 \oplus (\mathbb{Z}_3)^2$
39	$\mathbb{Z}_4$	$\mathbb{Z}_4 \oplus \mathbb{Z}_3 \oplus \mathbb{Z}$	$\mathbb{Z}_2 \oplus (\mathbb{Z}_4)^2 \oplus \mathbb{Z}_3 \oplus \mathbb{Z}^6$	$(\mathbb{Z}_2)^5 \oplus (\mathbb{Z}_4)^2 \oplus \mathbb{Z}_3$	$\mathbb{Z}_2 \oplus (\mathbb{Z}_4)^2 \oplus \mathbb{Z}_3$
42	0	$(\mathbb{Z}_2)^2 \oplus \mathbb{Z}_3 \oplus \mathbb{Z}$	$--$	$--$	$--$
46	$\mathbb{Z}_3$	$\mathbb{Z}_2 \oplus \mathbb{Z}_{11} \oplus \mathbb{Z}$	$\mathbb{Z}_4 \oplus \mathbb{Z}_8 \oplus (\mathbb{Z}_3)^4 \oplus \mathbb{Z}^2$	$(\mathbb{Z}_2)^3$	$\mathbb{Z}_4 \oplus \mathbb{Z}_8 \oplus (\mathbb{Z}_3)^4$

Table 3.1: Homology groups $H_n(SL_2(\mathbb{Z}[1/m]), \mathbb{Z})$

3.1 Resolutions for groups acting on trees

In this section, we will give a construction of a free resolution R_*^G with contracting homotopy for any amalgamated free product $G = G_1 *_A G_2$. More precisely, we give a proof for the following theorem.

Theorem 3.1.1. *A particular free resolution R_*^G, with contracting homotopy, for an amalgamated free product $G = G_1 *_A G_2$ is precisely and algorithmically determined by*

1. *free resolutions $R_*^{G_1}$, $R_*^{G_2}$, R_*^A with contracting homotopies, and*
2. *a contracting homotopy on the cellular chain complex of the tree associated to the amalgamated product.*

The tools for the construction are a discrete vector field and a practical version of Wall's lemma which was mentioned in Chapter 2. Before going into the construction, we recall some necessary basic concepts.

A *graph* consists of:

(i) a non-empty set V (called vertices),

(ii) a set E (called oriented edges),

(iii) a map $E \to V \times V$ which sends any $e \in E$ to the pair $(s(e); t(e))$ of its *source vertex* $s(e)$ and the *terminal vertex* $t(e)$ and

(iv) a map from E to itself which sends each edge e to its inverse edge $\bar{e}$ which is different from e and has its source and terminus switched and so that $\bar{\bar{e}} = e$.

A *segment* is a graph (V, E) where V contains two distinct vertices and E contains only a single edge joining those vertices. A *path* of length n in a graph is a concatenation $e_1 \dots e_n$ of edges where e_i starts at the vertex where e_{i-1} ends for $i = 2, \dots, n$. We say that the path is from $s(e_1)$ to $t(e_n)$. A *circuit* is a path as above where e_n ends at the initial vertex of e_1. A circuit of length 1 is called a *loop*. A graph is said to be *connected* if each pair of vertices is contained in some path.

A *tree* is a connected non-empty graph without circuits. A group G is said to act on a graph $\Gamma = (V; E)$ if G acts on the set V in such a way that G takes edges to edges. In particular, G preserves an orientation of Γ if, and only if, it acts without inversion i.e., $ge \neq \bar{e}$ for any edge e and any $g \in G$. A group G acts freely on Γ if it acts without inversion and a vertex can be fixed only by the identity element.

In this thesis, we will use the equivalent definition below.

Definition 3.1.1. A *tree* is a contractible 1-dimensional CW-space. Then a tree T on which a group G acts is a G-space and we call T a G-tree.

One example of a G-tree is the modular tree with the action of $SL_2(\mathbb{Z})$ as described in Section 2.3.

If G acts without inversion on a graph Γ, one can define the quotient graph of Γ by G in a natural manner. It is defined to be the graph whose vertex set is the set of G-orbits of vertices of Γ and the edges are G-orbits of edges of Γ under the G-action.

From now on in this chapter, we make the convention that each group acting on a graph acts without inversion.

Recall that, in graph theory, an *isomorphism* of graphs Γ and Γ' is a bijection between the vertex sets of Γ and Γ' such that any two vertices u and v of Γ are adjacent in Γ if and only if their images are adjacent in Γ'.

Definition 3.1.2. [28] Let G be a group acting on a graph Γ. A *fundamental domain* for the action of G is a subgraph $F \subset \Gamma$ such that $F \simeq \Gamma/G$, the isomorphism being induced from the quotient map from $\Gamma \to \Gamma/G$.

Proposition 3.1.2. [28] *Let G be a group acting on a tree T. A fundamental domain for the action of G exists if and only if T/G is a tree.*

Theorem 3.1.3. [28] *Let G be a group acting on a graph Γ. Let F be a segment in Γ. Suppose that F is a fundamental domain for the action of G. Let P, Q be the vertices and y be the edge of F. Let G_P, G_Q and G_y be the stabilizers of P, Q and y respectively. Then the following are equivalent:*

(i) Γ *is a tree.*

(ii) The canonical homomorphism $G_P *_{G_y} G_Q \to G$ induced by the inclusion $G_P \to G$ and $G_Q \to G$ is an isomorphism.

Theorem 3.1.4. [28] *Let $G = G_1 *_A G_2$ be an amalgam of two groups. Then there is a tree Γ (and only one, up to isomorphism) on which G acts, with fundamental domain a segment such that if the vertices of this segment are $\{P; Q\}$ and the edge are y then $G_1 \simeq G_P$, $G_2 \simeq G_Q$ and $A \simeq G_y$.*

3.1.1 Proof of Theorem 3.1.1 and related algorithms

The construction of a free resolution for $G = G_1 *_A G_2$ below is also the proof of the Theorem 3.1.1.

Consider the group $G = G_1 *_A G_2$ acting on the tree T with details as in Theorem 3.1.4. Then T is a G-space where the 0-cells are vertices of T and 1-cells are the edges of T. One can see that T has two orbits of 0-cells whose representatives are P, Q respectively and one orbit of 1-cells whose representative is y. Let $[e]$ denote the equivalence class of cells in the orbit of e under the action of G, and let $Orb(n)$ denote the set of equivalence classes of n-dimensional cells. The cellular chain complex of length 1 of T,

$$C_*(T) : \ 0 \xrightarrow{d_2} C_1(T) \xrightarrow{d_1} C_0(T) \xrightarrow{\epsilon} \mathbb{Z} \longrightarrow 0 \tag{3.1}$$

with $C_0(T) = \mathbb{Z}G \otimes_{\mathbb{Z}G_P} \mathbb{Z}^P \oplus \mathbb{Z}G \otimes_{\mathbb{Z}G_Q} \mathbb{Z}^Q$, $\qquad C_1(T) = \mathbb{Z}G \otimes_{\mathbb{Z}G_y} \mathbb{Z}^y$

where each $\mathbb{Z}^e$ is a copy of the integers endowed with an "orientation" action of G_e, with e respectively P, Q and y.

Suppose that we are given a free $\mathbb{Z}G_e$-resolution $R_*^{G_e}$ of $\mathbb{Z}$, where e is P, Q, y respectively. Proposition 2.1.2 gives a construction that inputs the non-free $\mathbb{Z}G$-resolution $C_*(X)$ together with the free $\mathbb{Z}G_e$-resolutions $R_*^{G_e}$, and outputs a free $\mathbb{Z}G$-resolution R_*^G of $\mathbb{Z}$. In this section we shall need this construction only for 1-dimensional G-space T. So from this point on we recall the construction in this particularly easy case. The boundary homomorphism $C_1(X) \to C_0(X)$ induces a $\mathbb{Z}G$-equivariant

homomorphism of $\mathbb{Z}G$-chain complexes

$$\begin{array}{ccc}
\downarrow{\scriptstyle \partial_{n+1}} & & \downarrow{\scriptstyle \partial_{n+1}} \\
\bigoplus_{[e]\in Orb(1)} (R_n^{G_e} \otimes_{\mathbb{Z}G_e} \mathbb{Z}^e) \otimes_{\mathbb{Z}G_e} \mathbb{Z}G & \xrightarrow{\delta_n} & \bigoplus_{[v]\in Orb(0)} R_n^{G_v} \otimes_{\mathbb{Z}G_v} \mathbb{Z}G \\
\downarrow{\scriptstyle \partial_n} & & \downarrow{\scriptstyle \partial_n} \\
\bigoplus_{[e]\in Orb(1)} (R_{n-1}^{G_e} \otimes_{\mathbb{Z}G_e} \mathbb{Z}^e) \otimes_{\mathbb{Z}G_e} \mathbb{Z}G & \xrightarrow{\delta_{n-1}} & \bigoplus_{[v]\in Orb(0)} R_{n-1}^{G_v} \otimes_{\mathbb{Z}G_v} \mathbb{Z}G \\
\downarrow{\scriptstyle \partial_{n-1}} & & \downarrow{\scriptstyle \partial_{n-1}}
\end{array}$$

which we view as a bicomplex. The free $\mathbb{Z}G$-resolution R_*^G is the total complex of this bicomplex. That is $R_n^G = A_{1,n-1} \oplus A_{0,n}$ where

$$A_{1,n-1} = \bigoplus_{[e]\in Orb(1)} (R_{n-1}^{G_e} \otimes_{\mathbb{Z}G_e} \mathbb{Z}^e) \otimes_{\mathbb{G}_e} \mathbb{Z}G, \qquad A_{0,n} = \bigoplus_{[v]\in Orb(0)} R_n^{G_v} \otimes_{\mathbb{G}_v} \mathbb{Z}G. \quad (3.2)$$

The boundary homomorphism is

$$d_n \colon A_{1,n-1} \oplus A_{0,n} \to A_{1,n-2} \oplus A_{0,n-1}, \quad x \oplus y \mapsto \partial_{n-1}(x) \oplus (-1)^n \delta_{n-1}(x) + \partial_n(y). \quad (3.3)$$

Recall that a *contracting homotopy* on the $\mathbb{Z}G$-resolution R_*^G is a family of $\mathbb{Z}$-linear homomorphisms $h_n \colon R_n^G \to R_{n+1}^G$ satisfying

$$h_{n-1}\partial_n + \partial_n h_{n-1} = 1$$

for all $n \geq 0$ with $h_{-1} = 0, \partial_0 = 0$. Such a contracting homotopy can be constructed from a contracting homotopy $h' \colon C_0(X) \to C_1(X)$ and contracting homotopies $h_n'' \colon R_n^{G_e} \to R_{n+1}^{G_e}$ on the $\mathbb{Z}G_e$-resolutions using the formula

$$\begin{aligned} h_n(x \oplus y) &= h_{n-1}''(x) \oplus ((-1)^n \partial_n \delta_n h_{n-1}''(x) + h_n''(y)) \ , \\ h_0(x) &= h'(x) \oplus (h_0''(x) - h_0'' \delta_0 h'(x)) \ . \end{aligned} \quad (3.4)$$

The above construction can be implemented in the form of the following algorithms.

Algorithm 3.1.1 Free resolution for $G = G_1 *_A G_2$

Input:

- the non-free $\mathbb{Z}G$-resolution (of length 1) $C_*(\mathcal{T})$ of $\mathbb{Z}$ as in 3.1;
- a free $\mathbb{Z}G_e$-resolution of $\mathbb{Z}$ with contracting homotopy for all cell stabilizers G_e;
- an integer $n \geq 0$.

Output: The first $n+1$ terms of a free $\mathbb{Z}G$-resolution R^G_* of $\mathbb{Z}$. Moreover, if the input C_* is endowed with a contracting homotopy then the output also has a contracting homotopy.

Procedure:

1: The nth-term R^G_n is computed by the formula 3.2;
2: The boundary is constructed by using formula 3.3;
3: If the input is endowed with a contracting homotopy then the contracting homotopy on R^G_* is constructed by using formula 3.4.

This algorithm is the special case of Algorithm 2.1.1 applied to a non-free $\mathbb{Z}G$-resolution of length 1. The differential map d of Proposition 2.1.1 is easier to implement since $d_k = 0$ for all $k > 1$. And we just need to implement the contracting homotopy for the case $q = 0$. So that this case can be implemented more efficiently. Moreover, this special case can be used to prove Proposition 3.2.1 about the periodicity of the homology of G.

The contracting homotopy on the cellular chain complex of the tree associated to the amalgamated product can be defined by using a contracting vector field. An example for constructing such a vector field is described in Section 3.2.

We could use the above construction, together with the standard resolution $R^{C_m}_*$ for finite cyclic groups C_m, to produce a free $\mathbb{Z}G$-resolution R^G_* for $G = SL_2(\mathbb{Z}) \cong C_4 *_{C_2} C_6$. However, for computational efficiency we prefer to use a slightly smaller resolution that is described in Section 2.3.

3.2 Resolution for $SL(2,\mathbb{Z}[1/m])$

3.2.1 Theoretical methods

Let m be a square free integer. Then $SL_2(\mathbb{Z}[1/m])$ is defined as

$$SL_2(\mathbb{Z}[1/m]) = \{A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \ : \ det(A) = 1 \text{ and } a,b,c,d \in \mathbb{Z}[1/m]\}.$$

Let m' be a square free integer, let p be a prime which is coprime to m' and set $m = pm'$. Let $\Gamma_0^{m'}(p) \subset SL_2(\mathbb{Z}[1/m'])$ be the subgroup defined as

$$\Gamma_0^{m'}(p) = \{\begin{pmatrix} a & b \\ c & d \end{pmatrix} \ : \ c \equiv 0 \ mod \ p\}.$$

Then $\Gamma_0^{m'}(p)$ is called *congruence subgroup* of $SL_2(\mathbb{Z}[1/m'])$ of level p.

In addition to the natural inclusion we have the injection $\Gamma_0^{m'}(p) \mapsto SL_2(\mathbb{Z}[1/m'])$ given by

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \mapsto \begin{pmatrix} a & pb \\ p^{-1}c & d \end{pmatrix}.$$

Using these two embeddings one has an isomorphism (see [28])

$$SL_2(\mathbb{Z}[1/(m)]) \cong SL_2(\mathbb{Z}[1/m']) *_{\Gamma_0^{m'}(p)} SL_2(\mathbb{Z}[1/m']). \tag{3.5}$$

For this amalgamated product, we replace one copy of $SL_2(\mathbb{Z}[1/m'])$ by its conjugate group $SL_2(\mathbb{Z}[1/m'])^P$ where $P = \begin{pmatrix} 1 & 0 \\ 0 & p \end{pmatrix}$ so that

$$\Gamma_0^{m'}(p) = SL_2(\mathbb{Z}[1/m']) \cap SL_2(\mathbb{Z}[1/m'])^P.$$

Hence 3.5 can be rewritten as:

$$SL_2(\mathbb{Z}[1/(m)]) \cong SL_2(\mathbb{Z}[1/m']) *_{\Gamma_0^{m'}(p)} SL_2(\mathbb{Z}[1/m'])^P. \tag{3.6}$$

Then by Theorem 3.1.4, $SL_2(\mathbb{Z}[1/(m)])$ acts on a tree $\mathcal{T}$. The action admits a fundamental domain F which is a segment with two vertices e_1^0, e_2^0 and one edge e^1. The cell stabilizers of the vertices respectively $SL_2(\mathbb{Z}[1/m'])$, $SL_2(\mathbb{Z}[1/m'])^P$ and stabilizer of the edge is $\Gamma_0^{m'}(p)$. We can thus construct a free $\mathbb{Z}SL_2(\mathbb{Z}[1/m])$-resolution $R_*^{SL_2(\mathbb{Z}[1/m])}$ from a free $\mathbb{Z}SL_2(\mathbb{Z}[1/m'])$-resolution $R_*^{SL_2(\mathbb{Z}[1/m'])}$ using the construction described in Section 3.1.1.

In principle one can apply this technique recursively to obtain a free $\mathbb{Z}SL_2(\mathbb{Z}[1/m])$-resolution for any m. For the recursion to work we need an algorithm for a contracting homotopy on the tree associated to the amalgamated sum. Such an algorithm boils down to one for expressing an arbitrary matrix $A \in SL_2(\mathbb{Z}[1/m])$ as a product of elements of each of the two copies of $SL_2(\mathbb{Z}[1/m'])$. Let $g \in SL_2(\mathbb{Z}[1/m])$. Note that, for all $h \in SL_2(\mathbb{Z}[1/m'])$, ge^1 and ghe^1 share a single common point ge_1^0. And for all $k \in SL_2(\mathbb{Z}[1/m'])^P$, ge^1 and gke^1 share a single common point ge_2^0. Suppose that g can be expressed as $g = h_1k_1h_2k_2...h_rk_r$ where $h_i \in SL_2(\mathbb{Z}[1/m'])$, $k_i \in SL_2(\mathbb{Z}[1/m'])^P$ and $k_i \notin SL_2(\mathbb{Z}[1/m'])$ for all $i \geq 1$ and $h_i \neq 1$ for all $1 < i$, $k_i \neq 1$ for all $i < r$. The contracting homotopy $h_0 : C_0(T) \longrightarrow C_1(T)$ is defined recursively as follows

$$h_0(ge_1^0) = \begin{cases} h_0(gk_r^{-1}h_r^{-1}e_1^0) & \text{if } k_r = 1, \\ ge^1 \oplus h_0(gk_r^{-1}e_2^0) & \text{if } k_r \neq 1. \end{cases} \tag{3.7}$$

and

$$h_0(ge_2^0) = h_0(gk_r^{-1}e_2^0) = gk_r^{-1}e^1 \oplus h_0(gk_r^{-1}h_r^{-1}e_1^0). \tag{3.8}$$

The required algorithm is a slight variant of the Euclidean type algorithm 2.3.1 described in section 2.3.

All the above information has been implemented in HAP as follows: (See Appendix 5.1 for the specification of the input/output data types.)

Algorithm 3.2.1 A $\mathbb{Z}G$-equivariant CW-space for $G = SL_2(\mathbb{Z}[1/m]) *_{\Gamma_0^m(p)} SL_2(\mathbb{Z}[1/m])$

Input: A pair of positive integers (m, p)

Output: A non-free $\mathbb{Z}G$-resolution $C_*(\mathcal{T})$ with contracting homotopy.

Procedure:

1: Find the chain complex $C_*(\mathcal{T})$ by using 2.2.
2: The contracting homotopy on $C_*(\mathcal{T})$ is calculated using a slight variant of the algorithm 2.3.1 and formulas 3.7, 3.8.

Algorithm 3.2.2 Free resolution for $G = SL(2,\mathbb{Z}[1/m])$

Input: A pair of integers (m, n), m is square-free.

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

Procedure:

Function:=ResolutionSL2Z(m, n)
1: **if** $m = 1$ **then**
2: return free resolution for $SL(2,\mathbb{Z})$ as in Section 2.3.
3: **end if**
4: Factorize m as a product of prime numbers, $m = p_1p_2...p_k$.
5: Let $m' = p_2...p_k$ and then $m = m'p_1$.
6: Compute RS:=ResolutionSL2Z(m',n);
7: Compute RH:= Resolution for the congruence subgroup $\Gamma_0^{m'}(p_1)$.
8: Compute RSp:= Conjugated resolution of RS by the matrix P:=[[1,0],[0,p]].
9: Compute C:= The chain complex constructed by algorithm 3.2.1 with stabilizers' resolutions are RS, RSp and RH.
10: Compute R:= the free resolution constructed by using algorithm 2.1.1 with the inputs C and n.
return R.
EndFunction.

In practice, we can use the lemma below to reduce the run-time.

Proposition 3.2.1. *Let $m = p_1p_2\dots p_n$ is a product of k primes, then the homology of $SL_2(\mathbb{Z}[1/m])$ is periodic of period 2 in degree greater than $k+1$.*

Proof. Let $G = SL_2(\mathbb{Z}[1/m])$. The Proposition is proved if we can show that:

i) For all $n > k+1$, the $\mathbb{Z}G$-modules satisfy

$$R_n(G) = R_{n+2}(G), \tag{3.9}$$

ii) and the boundary maps satisfy

$$d_n = d_{n+2}, \tag{3.10}$$

We know that 3.9, 3.10 are true in case $G = SL_2(\mathbb{Z})$. We will prove that they are also true for any positive k by induction. Let $m' = p_1 p_2 \dots p_{k-1}$ and $p = p_k$ and let $H = SL_2(\mathbb{Z}[1/m'])$. We assume that the free $\mathbb{Z}H$-resolution R_*^H is periodic of period 2 in degree greater than k.

Following the algorithm in section 3.1.1, the free $\mathbb{Z}G$-resolution R_*^G can be constructed by combining the free $\mathbb{Z}H$-resolution R_*^H, its conjugated resolution and the resolution of its congruence subgroup. In this case, G acts on tree $\mathcal{T}$ with the fundamental domain for this action a segment in which the edge y joining two vertices P, Q. Then $\mathcal{T}$ can be seen as a $G-$space with a single orbit of 1-cells represented by e^1 and two orbits of 0-cells represented by e_0^1 and e_0^2. The boundary map of the cellular chain complex $C_*(\mathcal{T})$ is

$$0 \longrightarrow C_1(\mathcal{T}) \xrightarrow{\varphi} C_0(\mathcal{T}) \longrightarrow 0$$

where $\varphi(e^1) = e_0^1 - e_0^2$.

The stabilizer of e^1, e_0^1, e_0^2 is respectively $\Gamma_0^{m'}(p)$, H and its conjugate H^P where $P = diag(1,p)$. We know that $\Gamma_0^{m'}(p)$ is a subgroup of both H and H^P. Fix the resolution R_*^H for H and we choose the conjugated resolution $R_*^{H^P}$ for H^P and $\mathbb{Z}\Gamma_0^{m'}$-resolution R_*^H for $\Gamma_0^{m'}(p)$ since $\Gamma_0^{m'}(p)$ is a subgroup of H.

By the formula 3.2, we have $A_{1,n} = A_{1,n+2}$ and $A_{0,n+1} = A_{0,n+3}$ for all $n > k$. Thus,

$$A_{1,n} \oplus A_{0,n+1} = A_{1,n+2} \oplus A_{0,n+3}.$$

Hence for all $n > k$,

$$R^G_{n+1} = R^G_{n+3}.$$

The periodicity of the boundary map in equation 3.10 can be proved by using formulae 3.3. By induction, ∂ is periodic, we just only need to prove the periodicity of δ.

$$R^H_n \otimes_{\mathbb{Z}G_{e^1}} \mathbb{Z}G \xrightarrow{\delta_n} R^H_n \otimes_{\mathbb{Z}G_{e^0_1}} \mathbb{Z}G \oplus R^{H^P}_n \otimes_{\mathbb{Z}G_{e^0_2}} \mathbb{Z}G.$$

For any $g \in G$ and f is a generator of R^H_n, we define

$$\delta_n(f \otimes_{\mathbb{Z}G_{e^1}} g) = f \otimes_{\mathbb{Z}G_{e^0_1}} g \oplus (-1)f \otimes_{\mathbb{Z}G_{e^0_2}} P^{-1}gP.$$

Since δ is actually induced from identity maps then δ and ∂ are commute. Therefore δ is a chain map. The periodicity of δ comes directly from the periodicity of R^H_*. This gives us the full proof for this lemma.

□

3.2.2 Practical results

The user interface to our implementation of the above algorithm is illustrated by the following GAP session which computes $H_5(G,\mathbb{Z}) \cong \mathbb{Z}_6 \oplus \mathbb{Z}_6$ for $G = \Gamma^2_0(3) \subset SL_2(\mathbb{Z}[1/2])$.

GAP session

```
gap> R:=ResolutionSL2Z(2,6);
Resolution of length 6 in characteristic 0 for SL(2,Z[1/2]) .

gap> G:=CongruenceSubgroup(2,3);
CongruenceSubgroup of SL(2,Z[1/2]) level 3
```

```
gap> S:=ResolutionFiniteSubgroup(R,G);
Resolution of length 6 in characteristic 0 for CongruenceSubgroup
of SL(2,Z[1/2]) level 3 .

gap> C:=TensorWithIntegers(S);
Chain complex of length 6 in characteristic 0 .

gap> Homology(C,5);
[ 6, 6 ]
```

The following GAP session illustrates the calculation of the cup product using the construction in section 1.3.3.

It shows that the cohomology ring $H^*(G,\mathbb{Z})$ has three generators $x,\ y,\ z$ of degree 2 and that

$$x \cup y = 0, \quad x \cup z = 2e_1^{(4)} + 4e_2^{(4)}, \quad y \cup z = 3e_2^{(4)}$$

where $e_1^{(4)},\ e_2^{(4)}$ are generators of $H^4(G,\mathbb{Z})$. Thus, the ring $H^*(G,\mathbb{Z})$ has no generator of degree 4.

GAP session

```
gap> Cohomology(HomToIntegers(S),4);
[ 6, 6 ]
gap> IntegralRingGenerators(S,2);
[ [ 1, 0, 0 ], [ 0, 1, 0 ], [ 0, 0, 1 ] ]
gap> IntegralCupProduct(S,[1,0,0],[0,1,0],2,2);
[ 0, 0 ]
gap> IntegralCupProduct(S,[1,0,0],[0,0,1],2,2);
[ 2, 4 ]
gap> IntegralCupProduct(S,[0,1,0],[0,0,1],2,2);
[ 0, 3 ]
gap> IntegralRingGenerators(S,4);
[  ]
```

An implementation of the algorithm in Section 3.2.1 is available in the HAP [11] package for the GAP [16] computational algebra system and has been used to com-

pute $H_n(SL_2(\mathbb{Z}[1/m]),\mathbb{Z})$ for all $m \leq 50, m \neq 30, 42$ and all $n \geq 0$. When $m = 30$ or $m = 42$ the implementation is practical only for $n = 1, 2$. Table 1 summarizes these computations.

The two main bottlenecks in our algorithm seem to be: the $\mathbb{Z}G$-rank of R_n^G in higher degrees n; the length of the boundary $\partial(e^n)$ of certain generators of R_n^G in higher degrees.

To illustrate the first bottleneck, consider the group $G = SL_2(\mathbb{Z}[1/(2\cdot 3\cdot 5\cdot 7\cdot 11\cdot 13)]$. Our current implementation attempts to build a resolution R_*^G in which R_n^G has $\mathbb{Z}G$-rank 1075200 for all $n \geq 7$.

To illustrate the second bottleneck, consider the group $G = SL_2(\mathbb{Z}[1/30])$. Our implementation attempts to build a resolution R_*^G in which R_n^G has $\mathbb{Z}G$-rank 480 in degrees $n \geq 4$. However, one generator f of R_4^G has boundary $\partial_4(f)$ involving 1359551 $\mathbb{Z}$-free generators of R_3^G. The implementation only succeeds in computing the boundaries of 292 $\mathbb{Z}G$-free generators of R_4^G before running out of memory on a Linux PC with 16 GB of random access memory.

Some attempts have been made at applying higher-dimensional Tietze type reductions to R_*^G to overcome these bottlenecks. Tietze reduction is an algorithm which inputs a set S of words in a free $\mathbb{Z}G$-module, and a word w in the module, it returns a word w' such that $\{S, w'\}$ generates the same abelian group as $\{S, w\}$. The word w' is possibly shorter (and certainly no longer) than w. This technique can be applied to reduce the number of generators in a free resolution. We can review it quickly as follows: Consider the boundary map: $d_{n+1} : R_{n+1}^G \to R_n^G$. Let x be a generator of R_{n+1}^G then $d_{n+1}(x) \in R_n^G$ and can be expressed as a word w of generators in R_n^G. We know that $d_{n+1}(x) = 0$ in $H_n(G)$ then if there is a generator y appears just only one time in w, then y can be expressed as a word of the other generators. Substitute y by that word in all the boundary $d_{n+1}(x)$ for all generators x in R_{n+1}^G and reduce the rank of R_n^G by 1. We hope that by applying this technique, the new resolution has less generators than the original one. So far the attempts have had limited success.

Chapter 4

Crystallographic groups with cubical fundamental region

4.1 Introduction

An *affine transformation* of $\mathbb{R}^n$ is a map $A : \mathbb{R}^n \to \mathbb{R}^n$, $x \mapsto qx + t$, where $q \in GL_n(\mathbb{R})$, $x, t \in \mathbb{R}^n$. An *n-dimensional crystallographic group* is a group G of affine transformations of $\mathbb{R}^n$ containing a free abelian subgroup T of rank n whose quotient G/T is a finite group. If G acts freely on $\mathbb{R}^n$ we call it a *Bieberbach group*. To be precise, an n-dimensional crystallographic group is a discrete group of the group of isometries $\mathrm{Isom}(\mathbb{R}) = \mathcal{O}(n) \ltimes \mathbb{R}$ of n-dimensional Euclidean space, having a compact fundamental domain. By the first Bieberbach's theorem [3] the translation subgroup, containing all pure translations inside a crystallographic group G, is normal and of finite index in G. This translation subgroup is a free abelian group of rank n and be generated by n linear independent translations. This leads to a short exact sequence

$$1 \to \mathbb{Z}^n \to G \to P \to 1$$

where $P \cong G/\mathbb{Z}^n$ is finite and we call P the *point grou*p of the crystallographic group G. An example of a 2-dimensional crystallographic group is the group generated by two affine transformations as follows. Let X be the unit translation along y-axis, $(x, y) \mapsto (x, y + 1)$; and Y be the glide reflection $(x, y) \mapsto (x, -y) + (1/2, 0)$. The group G generated by $\{X, Y\}$ is a 2-dimensional crystallographic group whose point group is $G/\mathbb{Z}^2 \cong C_2$, the cyclic group of order 2. The orbit space $\mathbb{R}^2/G$ is the Klein bottle. This group is a Bieberbach group, i.e. G acts freely on $\mathbb{R}^2$. In general, the action of crystallographic group on $\mathbb{R}^n$ is not free then we could not find the orbit space $\mathbb{R}^n/G$. By that we provide a definition 4.1.1 of fundamental region. More preliminaries of crystallographic groups can be found in [3].

Let G be a crystallographic group acting on $\mathbb{R}^n$.

Definition 4.1.1. A *fundamental region* for the action of G is a CW-space $\mathfrak{F}$ such that

1. $\mathfrak{F} \subseteq \mathbb{R}^n$;
2. for any cell $e \subset \mathfrak{F}$ and $g \in G$, we have $ge \cap e = \emptyset$ or $ge \cap e = e$;
3. no CW-subspace $\mathfrak{F}' \leq \mathfrak{F}$ is a fundamental region;

4. for all $x \in \mathbb{R}^n$, there exists $g \in G$ with $x \in g\mathfrak{F}$.

Definition 4.1.2. A fundamental region $\mathfrak{F}$ gives rise to a CW-decomposition of $\mathbb{R}^n$ with cells ge for $g \in G$, e a cell in $\mathfrak{F}$. We call this CW-structure on $\mathbb{R}^n$ a *tessellation*.

The CW-space $X = \mathbb{R}^n$ rising from the fundamental region $\mathfrak{F}$ is a contractible G-space where all stabilizers are finite (see Lemma 4.3.1 for the proof). The finiteness of stabilizer subgroups is an important advantage since the HAP package [11] can compute a free resolution for any finite group by the method explained in [10]. From a contractible G-space X, we can construct a classifying space for G by using the method described in [12]. In this chapter, we study the classifying space and cohomology structure of G in those cases where a cubical fundamental region can be found. The main advantages of a cubical fundamental region are that:

1. we can easily give explicit formulas for a contracting homotopy of the cellular chain complex $C_*(X)$ which can be used to compute the cohomology ring structure for G;

2. cubical subdivision of the fundamental region can be used, if necessary, to obtain the Bredon homology of G. We will not treat Bredon homology in this thesis, see [30] for more information.

All algorithms mentioned in this chapter are implemented as a sub-package of HAP [11]. Typical applications of the sub-package are Propositions 4.1.1 and 4.1.2. See section 4.2 to get the idea how elements of a crystallographic group can be represented as matrices. Proofs for these propositions are given in section 4.3 and 4.6.

Proposition 4.1.1. *Let G be the 3-dimensional Bieberbach crystallographic group generated by the set of matrices*

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/2 & 1 \end{pmatrix}, \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1/2 & 1/2 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}.$$

The integral cohomology ring is

$$H^*(G,\mathbb{Z}) = \mathbb{Z}[x,y,z]/J$$

where $J = \langle x^2, xz, y^2, yz, z^2, 2y, 2z\rangle$ *and* $deg(x) = 1$, $deg(y) = deg(z) = 2$. *By* $deg(x) = n$, *we mean* $x \in H^n(G,\mathbb{Z})$.

Proposition 4.1.2. *Let* G *be the 3-dimensional crystallographic group generated by the set of matrices*

$$\begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 0 & 0 \\ -1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}.$$

Then the action of G *on* $\mathbb{R}^3$ *does not admit a parallelepiped fundamental region.*

In order to calculate the group homology of crystallographic groups and construct their ring structures, in this chapter, we introduce a new algorithm which attempts to find a cubical or parallelepiped fundamental region for a crystallographic group. This fundamental region can be used to achieve the above goals. As we already mentioned in Section 1.2, in GAP [16], there are two methods for calculating the group cohomology of crystallographic groups implemented in HAP [11]. One of those two methods is developed and implemented by Graham Ellis [11]. Another method is provided by Marc Röder [26]. These two methods are described in detail in the following sub-sections.

4.1.1 Wall's extension method

Let G be an extension of the normal subgroup T by its quotient group P. In the paper [31], Wall introduces a technique to construct a free resolution for G by using given free resolutions for T and P. This technique is applied to crystallographic groups by the method described in [12]. This algorithm applies Wall's technique to the exact sequence

$$1 \to T \to G \to P \to 1$$

involving the finite point group P and the free abelian translation subgroup T. Free resolutions R_*^T and R_*^P are constructed in HAP [11]. Since T is a subgroup of G then $R_*^T \otimes_T \mathbb{Z}G$ is a $\mathbb{Z}G$-chain complex. Assume that R_s^P is free on c_s generators. We define $A_{r,s}$ to be the direct sum of c_s copies of $R_r^G \otimes \mathbb{Z}G$. Let $R_n^G = \oplus_{r+s=n} A_{r,s}$. Wall [31] gives us a construction of a differential map $d : R_*^G \to R_*^G$ such that (R_*^G, d) is a free $\mathbb{Z}G$-resolution (See section 2.1 for the construction of d). In principle, this method works for all groups and yields cohomology ring for such groups. But it also has some disadvantages which are:

1. For some crystallographic groups, it produces a big resolution which has a large rank in each degree. The algorithm is implemented in HAP. For some cases it doesn't stop after one hour of running and sometimes gives an error of exceeding permitted memory. See the below GAP session for an example.

GAP session

```
gap> G:=SpaceGroup(4,4700);
SpaceGroupOnRightBBNWZ( 4, 32, 14, 4, 4 )
gap> H:=Image(IsomorphismPcpGroup(G));
Pcp-group with orders [ 2, 2, 2, 2, 2, 2, 0, 0, 0, 0 ]
gap> ResolutionAlmostCrystalGroup(H,5);
Error, exceeded the permitted memory
(‘-o’ command line option) in
res[i] := func( C[i] ); called from
List( b, function ( x )
```

```
return [ - x[1], x[2] ];
end ) called from
ChangeSign( x[1], c ) called from
Contraction( i, y ) called from
Contraction( i, y ) called from
Contraction( i - 1, x ) called from
...  at line 22 of *stdin*
you can 'return;'
brk>
```

2. Wall's technique is not well-suited for calculating Bredon homology.

4.1.2 Röder's method

Röder [26] has designed and implemented an algorithm for arbitrary Bieberbach groups. The algorithm uses convex hull computations to construct a fundamental region for a Bieberbach group. The advantage of this method is it produces a finite dimensional regular CW-complex for G an n-dimensional Bieberbach group. That space is the universal cover of a classifying space for G since G acts freely on it. The disadvantages of Röder's approach are:

1. The package uses convex hull computations which will make the functions slow for high-dimensional Bieberbach groups.

2. For finding the free resolution, the current implementation and some of the current theory applies only to Bieberbach groups.

3. The fundamental region produced by this algorithm is a polytope which deeply depends on the choice of the starting point of a Dirichlet-Voronoi construction. Most polytopes are too complicated to give any obvious method for finding a contracting homotopy and so yield only additive structure not ring structure.

4. The method is not well-suited for calculating Bredon homology.

4.1.3 Our contribution

We observe that a high proportion of low-dimensional crystallographic groups admit a cubical fundamental region (See below for the definition). The main benefits of our method are that:
(i) it applies to both Bieberbach and non-Bieberbach groups with cubical fundamental region. (ii) it yields the cohomology ring and is well-suited for computing Bredon homology.

4.2 Some definitions

Let $\mathbb{R}^n$ be Euclidean space with the standard inner product and the standard metric. A map $f : \mathbb{R}^n \to \mathbb{R}^n$ is an *isometry* if it preserves distance. The set of all isometries of $\mathbb{R}^n$ is a group with respect to composition of maps. We denote this group by $\mathbb{E}(n)$. Examples of isometries are orthogonal mappings and translations. In fact, one can prove that if $f : \mathbb{R}^n \to \mathbb{R}^n$ is an isometry of $\mathbb{R}^n$, then there exists a translation t and an orthogonal linear map $A : \mathbb{R}^n \to \mathbb{R}^n$ such that $f = t \circ A$.

The action of $\mathbb{E}(n)$ on $\mathbb{R}^n$ is given by

$$\mathbb{E}(n) \times \mathbb{R}^n \to \mathbb{R}^n, \quad (f, v) \mapsto t \circ A(v)$$

The set of all translations of $\mathbb{R}^n$ is a normal subgroup of $\mathbb{E}(n)$. The set of all orthogonal linear maps $\mathbb{R}^n \to \mathbb{R}^n$ is a group which is denoted by $\mathbb{O}(n)$. One can prove that $\mathbb{E}(n)$ is isomorphic to semi-direct product of $\mathbb{O}(n)$ and $\mathbb{R}^n$.

For the purpose of this thesis, we introduce an equivalent definition of crystallographic group as follows:

We define a matrix $a \in GL_{n+1}(\mathbb{R})$ to be *crystallographic* if it is of the form

$$a = \left(\begin{array}{c|c} p & 0 \\ \hline t & 1 \end{array}\right)$$

with p an $n \times n$ matrix, t a row vector and 0 the zero column vector of length n. Sometimes we write it as $a = (p + t)$ for convenience. We say that p is the *finite part* of a. We say that a is a *translation* if its finite part is the $n \times n$ identity matrix.

Definition 4.2.1. An n-dimensional *crystallographic group* G can be defined to be a group of $(n + 1) \times (n + 1)$ crystallographic matrices whose finite parts form a finite subgroup $P_G \subset GL_n(\mathbb{R})$ and whose translations form a free abelian subgroup $T_G \subset G$ of rank n. We call T_G the *translation subgroup* of G. This subgroup is normal in G with quotient $G/T_G \cong P_G$, the *point group* of G. We mostly write T and P instead of T_G and P_G.

In general, the point group is not a subgroup of the orthogonal group of dimension n but we can orthogonalize it by using a suitable basis of $\mathbb{R}^n$ as in the following process. Let G be a crystallographic group as in Definition 4.2.1 with a point group P whose elements need not to be orthogonal.

Let

$$M = \frac{1}{|P|} \sum_{a \in P} a^t a$$

where a^t is the transposed matrix of a.

We define a metric on $\mathbb{R}^n$ using the inner product

$$\langle u, v \rangle := u^t M v. \tag{4.1}$$

We call M the *Gramian matrix of the average inner product* or *Gramian* for short. Let $\mathbb{R}^n_M$ denote the vector space $\mathbb{R}^n$ with inner product 4.1. We use the notation $Gram(P)$ to denote the Gramian matrix related to the point group P. We will prove that G is a group of isometries of $\mathbb{R}^n_M$. Let $g \in G$. Since pure translation is always an isometry of $\mathbb{R}^n_M$ then we just only need to prove that the finite part of g is also an isometry. Let b is the finite part of g. Consider

$$\langle bu, bv \rangle = (bu)^t M (bv) = u^t b^t M b v = u^t (b^t M b) v$$

We also have $b^t M b = b^t (\frac{1}{|P|} \sum_{a \in P} a^t a) b = \frac{1}{|P|} \sum_{a \in P} (ab)^t (ab) = M$ since P is finite.

Therefore $\langle bu, bv\rangle = \langle u, v\rangle$. And then g is an isometry of $\mathbb{R}^n$ with respects to the metric defined by 4.1.

Since M is symmetric then there exists an invertible matrix $E \in GL_n(\mathbb{R}^n)$ such that $M = E^{-1}DE$, where D is the diagonal matrix $(\lambda_1, \lambda_2, ..., \lambda_n)$ with λ_i are positive real eigenvalues for all i. Let $S = diag(\sqrt{\lambda_1}, \sqrt{\lambda_2}, ..., \sqrt{\lambda_n})$ and $Q = ES$. Then Q is the change-of-basis matrix that turns the Gramian back to identity matrix. In other words, the conjugate of G by Q is a group of isometries of $\mathbb{R}^n$ in the standard metric. This leads to the below Lemma.

Lemma 4.2.1. *Let G be an n-dimensional crystallographic group. There exists an invertible $n \times n$-matrix Q such that the conjugate of G by Q is a group of isometries of $\mathbb{R}^n$ in the standard metric.*

By the Lemma 4.2.1, from now on we will work only with crystallographic matrices whose finite parts are orthogonal and so their point groups are subgroups of $\mathbb{O}(n)$. Moreover, the translation basis is the standard basis.

Let G be an n-dimensional crystallographic group. The action of G on $\mathbb{R}^n$ is induced by the action of $\mathbb{E}(n)$ as follows:

Any matrix $a \in G$ acts on a column vector $v \in \mathbb{R}^n$ by the formula

$$a \colon \mathbb{R}^n \to \mathbb{R}^n, v \mapsto pv + t \ .$$

We define a *lattice* to be a discrete subgroup L of the additive subgroup of $\mathbb{R}^n$. We say that L is n-dimensional if it is free abelian of rank n. We say that L is a *G-lattice* if $a \cdot v \in L$ for all $a \in G, v \in L$.

Definition 4.2.2. Let G be an n-dimensional crystallographic group. We say that a G-lattice $L \subset \mathbb{R}^n$ admits a *G-full basis* $B_L = \{v_1, \ldots, v_n\} \subset L$ if the following hold:

1. The set $B_L = \{v_1, \ldots, v_n\} \subset L$ is a $\mathbb{Z}$-basis for L.

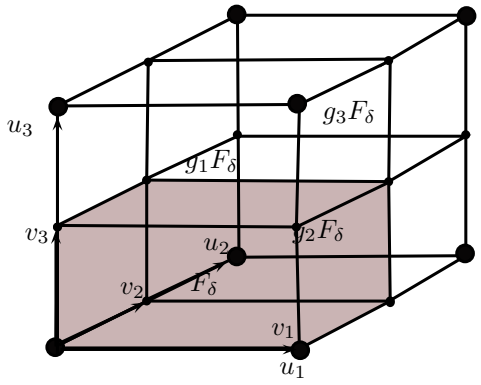


Figure 4.1

2. There exists integers $\lambda_1, \lambda_2, \cdots, \lambda_n$ such that $\{\lambda_1 v_1, \lambda_2 v_2, \cdots, \lambda_n v_n\}$ form a basis for a T-lattice.

3. Let $c = \frac{1}{2}(v_1 + \cdots + v_n)$. For each $v = c + (\alpha_1 v_1 + \alpha_2 v_2 + \cdots + \alpha_n v_n)$, $0 \leq \alpha_i \leq \lambda_i - 1$, there exists some $g \in G$ satisfying

$$g \cdot c = v.$$

Definition 4.2.3. Let G be an n-dimensional crystallographic group. We shall call G a *cubical crystallographic group* if there exists a G-lattice which admits a G-full basis.

Example 4.2.1. Let G be the crystallographic group in Proposition 4.1.1. Figure 4.1 illustrates an example of a G-full basis. In the figure, large dots represent elements in the translation subgroup T. Elements in the lattice L are represented by both large and small dots.
The set of vectors $\{v_1, v_2, v_3\} = \{(1,0,0), (0,1/2,0), (0,0,1/2)\}$ is a G-full basis for the G-lattice L. And $\{u_1, u_2, u_3\} = \{1v_1, 2v_2, 2v_3\} = \{(1,0,0), (0,1,0), (0,0,1)\}$ is a basis for a T-lattice. Thus, G is a cubical crystallographic group.

4.3 Cubical fundamental region for crystallographic groups

In this section, we will introduce an algorithm which attempts to find a cubical fundamental region for a *cubical crystallographic* group. We also give a proof of the Proposition 4.1.2.

Let G be a crystallographic group. Then the action of G on the Euclidean space $\mathbb{R}^n$ admits a fundamental region which gives rise to a CW-decomposition of $\mathbb{R}^n$.

Definition 4.3.1. Let $v \in \mathbb{R}^n$ and let G be an n-dimensional crystallographic group. The *Dirichlet-Voronoi region* of G with respect to v is

$$\mathcal{D}_G(v) = \{u \in \mathbb{R}^n : \| v - u \| \leq \| gv - u \|, \text{ for each } g \in G\} \tag{4.2}$$

We refer to v as the *kernel point*.

Standard theory of polytopes shows that $\mathcal{D}_G(v)$ is a *polytope*, i.e, a convex hull of a set of finitely many points. It is easy to see a Dirichlet-Voronoi region is a fundamental region for the action of G on $\mathbb{R}^n$.

Lemma 4.3.1. *Let G be a crystallographic group acting on the Euclidean space $\mathbb{R}^n$. The action admits a fundamental region $\mathfrak{F} = \mathcal{D}_G(v)$ which gives rise to a CW-structure on $X = \mathbb{R}^n$. Let e be an arbitrary cell in X. Then the stabilizer subgroup $G_e = \{g \in G : \ ge = e\}$ is finite.*

Proof. One can see that $\mathcal{D}_G(v)$ is a CW-subspace of the CW-space $X = \mathbb{R}^n$ which is risen from the fundamental region $\mathcal{D}_G(v)$. For each dimension $k \leq n$, every orbit of k-cells in X has a representative in $\mathcal{D}_G(v)$. Let e be a k-cell in the CW-space $X = \mathbb{R}^n$ then the closure $\bar{e}$ is a convex hull of finitely many points $V = \{x_1, x_2, .., x_r\}$. Let $g \in G_e$ then g is a permutation of V. Let $c = \frac{1}{r}(x_1 + x_2, ..., x_r)$ then g fixes c that means $g \in G_c$ the stabilizer of c. Thus G_e is a subgroup of G_c. We will prove that G_c is finite. Let $g = p + t$ is an element in G where $p \in P$. If $g \in G_c$ then $gc = c$, that means $pc + t = c$. Thus $t = pc - c$. Hence, for each $p \in P$, there is at

most one $g = p + t \in G$ which is in the cell stabilizer G_c. Since P is finite then G_c is finite. The lemma has been proved.

□

Definition 4.3.2. A fundamental region F is *cubical* if it is *affine equivalent* to an n-cube (i.e. there exists an affine transformation sending F to an n-cube).

Theorem 4.3.2. *Let G be a crystallographic group of dimension n. The following data are equivalent:*

i) A G-lattice L with specified G-full basis B_L.

ii) A regular CW-structure on $\mathbb{R}^n$ for which the G-action permutes cells and for which there is a single orbit of n-cells, the orbit being represented by a cubical n-cell; and for which the origin is one of the 0-cells.

Proof. Let G be an n-dimensional crystallographic group. Suppose that there is a G-lattice L with specified G-full basis $B_L = \{v_1, \ldots, v_n\}$. Let F_L denotes the parallelepiped determined by the origin and the set of vectors $\{v_1, \ldots, v_n\}$ as follows:

- consider the origin as the first vertex;
- the next n vertices are the translations of the origin by $v_1, \ldots, v_n$ respectively;
- the other vertices are determined by the parallelism of edges.

By the condition (2) of definition 4.2.2, there exists some integers $\lambda_1, \lambda_2, \cdots, \lambda_n$ such that $\{u_1, \ldots, u_n\} = \{\lambda_1 v_1, \lambda_2 v_2, \cdots, \lambda_n v_n\}$ form a basis B_T for a T-lattice L_T. Vectors in B_T form a parallelepiped F_T which is tessellated by some images of F_L under the action of G. Moreover, $\mathbb{R}^n$ is tessellated by F_T and its images under the action of T. Therefore, the images of F_L under the action of G yield a tessellation of $\mathbb{R}^n$. Such a tessellation induces a regular CW-structure X on $\mathbb{R}^n$ given by the k-cells are the open k-dimensional parallelepipeds which are the k-faces of some images of F_L, $0 \leq k \leq n$. The condition (3) in Definition 4.2.2 proves that the action of G permutes cells and furthermore that there is a unique n-cell orbit represented by

the interior of F_L. On the other hand, if there is a regular CW-structure on $\mathbb{R}^n$ for which the G-action permutes cells and for which there is a unique n-cell orbit represented by a cubical n-cell. Choose the cubical n-cell in which the origin lies on it boundary. Then such a cubical n-cell provides a parallelepiped which is the fundamental region of the action of G. That parallelepiped is determined by a set of vectors B_L which can be considered as a G-full basis for a G-lattice L generated by B_L. □

Suppose that there is a G-full basis B_L for a G-lattice L. This basis determines an n-dimensional parallelepiped F_L. So by Theorem 4.3.2 there is also a corresponding CW-structure X on $\mathbb{R}^n$. Since the action of G permutes (maybe not freely) all cells and X is contractible, we can construct the cellular chain complex $C_*(X)$. Let T be the translation subgroup of G. Then G fits into the exact sequence

$$1 \to T \xrightarrow{i} G \xrightarrow{p} G/T \to 1$$

The action of G on $\mathbb{R}^n$ induces an action of G/T on the n-torus $\mathbb{T}^n \cong \mathbb{R}^n/T$. So there is a CW-structure Y on the torus $\mathbb{T}^n$ with the cell structure induced from X. We can view $\mathbb{T}^n$ as the n-dimensional parallelepiped with the opposite hyperplane segments identified. By Theorem 4.3.2, X is a G-space with a single orbit of n-cells and hence so is Y. Let $\bar{\sigma}$ is the representative of the orbit of n-cells in Y. Since there is a single orbit of n-cells and elements in P permutes cells of Y then the images of $\bar{\sigma}$ by the action of P are all the n-cells in Y. Therefore, the number of n-cells in Y, say d, is a divisor of the order of the point group P. Thus we can say, $\mathbb{T}^n$ is a (cubical) tessellation by d n-dimensional parallelepipeds. It is equivalent to that the parallelepiped F is a (cubical) tessellation by d sub-parallelepipeds as in the Figure 1 for instance. And that means $\lambda_1\lambda_2\cdots\lambda_n = d$. We will try to find a G-full basis by slicing up F_T into d parts. The CW-complex X consists of k-cells e^k or geometrically k-dimensional parallelepipeds and each is completely determined by its center point. Moreover, the action by an elements of G permutes k-cells and therefore moves the center point of one cell to the center point of the other. Thus, we can identify each k-cell e^k by its center point.

Consider a T-lattice L_T determined by a basis $B_T = \{u_1, u_2, \cdots, u_n\}$. Decompose

d into product of n positive integers, $d = \lambda_1 \times \lambda_2 \times ... \times \lambda_n$. Say $\delta = (\lambda_1, \lambda_2, ..., \lambda_n)$ a *partition vector* of F , that means the ith-edge of F is divided into λ_i parts and F is tessellated by d sub-parallelepipeds. By the partition vector δ, let $B_\delta = \{v_1, v_2, \cdots, v_n\}$, where $v_i = \lambda_i^{-1} u_i$. Vectors in B_δ form a sub-parallelepiped F_δ. Let L_δ be the lattice generated by B_δ. The questions are that: (i) Is L_δ a G-lattice, closed under the action of G? Is B_δ a G-full basis for L_δ? Those questions are answered by following propositions in which S is the transversal $\{(g + t_g)|g \in P\}$ of the translation subgroup T in G.

Proposition 4.3.3. *L_δ is a G-lattice with the specified G-full basis B_δ if and only if these conditions are satisfied:*

(i) B_δ satisfies the condition (3) in the definition of G-full basis.

(ii) For all $x \in S$ and $v_i \in B_\delta$, $x \cdot v_i \in L_\delta$, for all $1 \leq i \leq n$; and $x \cdot O \in L_\delta$ where O is the origin.

Proposition 4.3.4. *Let u, v be two points in $\mathbb{R}^n$. Then they are equivalent under the action of G if and only if there exists some $(g+t_g) \in S$ such that $(g+t_g)u - v \in T$.*

Proof. Recall that u, v are equivalent under the action of G if and only if there exists an element $(g + t) \in G$ such that $(g + t)u = v$. We know that $(g + t)$ and $(g + t_g)$ lie in the same coset of T in G then $t = t_g + t'$ for some $t' \in T$. Therefore, $(g + t_g)u - v = t' \in T$. □

All above discussion yields the following algorithm which attempts to find a cubical fundamental region for crystallographic group G.

Algorithm 4.3.1 G-full basis

Input: A set of crystallographic matrices generating a crystallographic group G.

Output: It outputs either "fail" or a G-full basis for a G-lattice L.

Procedure:

1: Compute a translation basis B_T for a T-lattice L_T. We order B_T and let $(B_T)_i$ denote its i^{th} vector.
2: Create a list D of all positive divisors of the order of the point group.
3: **for** x in D **do**
4: Find a list of all vectors $\lambda = (\lambda_1, \lambda_2, ..., \lambda_n) \in \mathbb{Z}^n$ such that $\lambda_1\lambda_2...\lambda_n = x$. We call λ a *partition vector*.
5: **for** each partition vector λ **do**
6: $B_\lambda = (\lambda_i^{-1} * (B_T)_i : 1 \leq i \leq n)$
7: Test if B_λ is a G-full basis for a G-lattice by using the definition.
8: **if** yes **then**
9: **return** B_λ.
10: **end if**
11: **end for**
12: **end for**
13: **return** failed.
14: End.

Example 4.3.1. Let G be the crystallographic group generated by the list of crystallographic matrices given in the Proposition 4.1.1. The GAP session below proves that $\{(1,0,0),(0,1/2,0),(0,0,1/2)\}$ is a set of vectors which determine a cubical fundamental region for the action of G on the Euclidean space $\mathbb{R}^3$. Then G is a cubical crystallographic group.

GAP session

```
gap> gens:=[[[1,0,0,0],[0,-1,0,0],[0,0,1,0],[0,0,1/2,1]],
> [[-1,0,0,0],[0,-1,0,0],[0,0,1,0],[0,1/2,1/2,1]],
> [[1,0,0,0],[0,1,0,0],[0,0,1,0],[1,0,0,1]],
> [[1,0,0,0],[0,1,0,0],[0,0,1,0],[0,1,0,1]]];;
gap> G:=Group(gens);;
gap> B:=CrystGFullBasis(G);
[ [ [ 1, 0, 0 ],[ 0, 1/2, 0 ],[ 0, 0, 1/2 ] ],[ 1/2, 1/4, 1/4 ] ]
```

Example 4.3.2. Let G be the crystallographic group generated by the list of crystallographic matrices given in the Proposition 4.1.2. The GAP session below shows

that our method can not find a cubical fundamental region for the action of G.

GAP session

```
gap> gens:=[[[-1,0,0,0],[0,-1,0,0],[0,0,-1,0],[0,0,0,1]],
> [[-1,0,0,0],[0,-1,0,0],[0,0,1,0],[0,0,0,1]],
> [[0,1,0,0],[-1,-1,0,0],[0,0,1,0],[0,0,0,1]],
> [[1,0,0,0],[0,1,0,0],[0,0,1,0],[1,0,0,1]],
> [[1,0,0,0],[0,1,0,0],[0,0,1,0],[0,1,0,1]],
> [[1,0,0,0],[0,1,0,0],[0,0,1,0],[0,0,1,1]]];;
gap> G:=AffineCrystGroup(gens);;
gap> B:=CrystGFullBasis(G);
fail
```

In fact, by looking at the generators of G, we know that the point group P is a subgroup of G. So if G acts on a cubical lattice then P acts faithfully on a cube. That means P is a subgroup of the group of automorphisms of a cube. One can find the complete list of subgroups of the group of automorphisms of a cube and find all of their structures as follows: {"1", "C2", "C3", "C2 x C2", "C4", "C6", "S3", "C2 x C2 x C2", "D8", "C4 x C2", "A4", "D12", "C2 x D8", "C2 x A4", "S4", "C2 x S4"}. The structure of the point group P of G is "C6 x C2" does not belong to the list. Thus, P is not isomorphic to any subgroup of the group of automorphism of a cube. Hence, G is not a cubical crystallographic group. It is also a proof for Proposition 4.1.2.

4.4 Non-free resolutions for crystallographic groups

4.4.1 Construction of non-free resolutions

Let G be a crystallographic group with the free abelian translation subgroup T and the finite point group P of order m. On replacing G by a suitable conjugate group we can assume that the translation subgroup T has basis B_T the translations $x \mapsto x + e_i$ $(1 \leq i \leq n)$ with e_i the standard basis vectors of $\mathbb{R}^n$. Suppose that we have already found a G-full basis B_δ for a G-lattice L_δ. By Theorem 4.3.2, there

is a CW-structure on $X = \mathbb{R}^n$ on which the G-action permutes cells. The CW-space X consists of k-cells σ^k which are identified with their centers of the form $(t_1/2, t_2/2, ..., t_n/2)$, where t_i are integers and the dimension k is the number of t_i that are odd. We now provide some remarks which will be used to construct a G-equivariant CW-space .

(1) The parallelepiped F determined by B_T consists of d n-cubes, where d is a divisor of the order of the point group. For each k, $0 \leq k \leq n$, let b_k be the set of all k-faces of those n-cubes. It is easy to see that the representative for every k-cell orbits in X can be chosen from b_k.

(2) We define an *orientation* of k-dimensional Euclidean space is an ordering of the standard basis. Recall that $\mathbb{R}^n$ is a (cubical) tessellation by F and its images under the action of G. Thus, all its cells are parallelepipeds and each k-cell is contained in a k-dimensional hyperplane. Let all the cells of the space X have orientations induced from the orientation of $\mathbb{R}^n$. Suppose g is an element of G acts on a k-dimensional cell τ^k. Geometrically $g\tau^k = \sigma^k$. The orientation of $g\tau^k$ is defined as follows: Assume that B_{τ^k} the chosen basis for the k-dimensional space containing k-cell τ^k and B_{σ^k} the chosen basis for the k-dimensional space containing σ^k. The orientation is defined by the sign of the determinant of the matrix which exactly is the matrix of basis transformation from gB_{τ^k} to B_{σ^k}.

(3) Let τ^k be a k-cell representative of a k-cell orbit, $\tau^k = (t_1/2, t_2/2, ..t_n/2)$. Suppose that $\{t_{p_1}, t_{p_2}, ..., t_{p_k}\}$ is a full list of all k odd numerators in the coordinate of τ^k. Then we define the jth-front face of τ^k and the jth-back face

$$F_j(\tau^k) = (\frac{t_1}{2}, ..., \frac{t_{p_j-1}}{2}, \frac{t_{p_j}-1}{2}, \frac{t_{p_j+1}}{2}, ..., \frac{t_n}{2})$$

$$B_j(\tau^k) = (\frac{t_1}{2}, ..., \frac{t_{p_j-1}}{2}, \frac{t_{p_j}+1}{2}, \frac{t_{p_j+1}}{2}, ..., \frac{t_n}{2})$$

The boundary map $\partial_k : C_k \to C_{k-1}$ is defined:

$$\partial_k(\tau^k) = \sum_{j=1}^{k} (-1)^j [F_j(\tau^k) - B_j(\tau^k)]$$

(4) The stabilizer of a k-cell τ^k is the stabilizer of its center $Z(\tau^k)$, that is the set $Stab(\tau^k) = \{g \in G : gZ(\tau^k) = Z(\tau^k)\}$.

The above comments are used to design an algorithm for constructing a G-equivariant CW-space, where G is a crystallographic group which succeeds to find a cubical fundamental region by using algorithm 4.3.1.

Algorithm 4.4.1 G-equivariant CW-space

Input: a set of crystallographic matrices that generate a crystallographic group G together with a G-full basis for a G-lattice L.

Output: A non-free $\mathbb{Z}G$-resolution of $\mathbb{Z}$ with finite stabilizer groups.

Procedure:

1: **Step 1.** Find a list $CellList$ in which $CellList[k]$ is a list of all representatives for k-dimensional orbits, where $0 \leq k \leq n$ as follows:
Create an empty list $CellList := []$. For each $k,\ 1 \leq k \leq n$:
- Create an empty list $CellList[k] := []$.
- Create a list of all centers of k-faces of the parallelepiped F which is determined by the input G-full basis:

$$cells = \{c_1^k, c_2^k, \cdots, c_m^k\}.$$

- Run through all elements in the list $cells$, if c_i^k lies in the same orbits of any point in the list $CellList[k]$ then check for the next one; otherwise add c_i^k to the list $CellList[k]$.

2: **Step 2.** Find the stabilizers for each cells in $CellList$.
- For the cell whose center is $x \in \mathbb{R}^n$, run through the transversal of the translation subgroup T in G to find all element $(g+t_g)$ such that $(g+t_g)x - x = t \in T$ then $(g + t_g - t)$ is an element in the stabilizer of x.

3: **Step 3.** Construct a non-free $\mathbb{Z}G$-resolution $C_*(X)$.
- Create a "non-free resolution" data type:

$$0 \longrightarrow C_n(X) \longrightarrow \ldots \longrightarrow C_1(X) \longrightarrow C_0(X) \longrightarrow \mathbb{Z} \longrightarrow 1$$

where $C_i(X)$ is generated by i-cells.
- By **remark 3**, the boundary maps $\partial_k : C_k \to C_{k-1}$ is defined:

$$\partial_k(\tau^k) = \sum_{j=1}^{k} (-1)^j [F_j(\tau^k) - B_j(\tau^k)].$$

- Identify $F_j(\tau^k)$ and $B_j(\tau^k)$ with their representatives in the list $CellList$ together with the sign of orientation mentioned in **remark 2**.

4: **Step 4.** return $C_*(X)$.

5: End.

4.4.2 Contracting homotopy for cubical case

One of the main reasons that we focus on cubical fundamental cells is that we can easily give a discrete vector field on the cubical CW-structure as follows.

Figure 4.2: Discrete vector field on $\mathbb{R}$

Consider the CW-decomposition of the real line $\mathbb{R}$ with 0-cells the singletons $\{n\}$ and 1-cells the open intervals $(n, n+1)$ for $n \in \mathbb{Z}$. A discrete vector field on this CW-space, involving just one critical 0-cell and no critical 1 -cells, is shown in Figure 4.2. By repeated use of Proposition 4.4.1 the discrete vector field on $\mathbb{R}$ extends to a discrete vector field on $X = \mathbb{R}^n = \mathbb{R} \times \mathbb{R}^{n-1}$.

Proposition 4.4.1. *Let X be a regular CW-space endowed with an admissible discrete vector field. Let E denote the set of critical cells of X. Then there is an admissible discrete vector field on $X \times X$ whose critical cells are those cells $e \times e'$ for $e,\ e' \in E$. This discrete vector field on $X \times X$ can be constructed by the following rules for $e^p \times f^q$ an arbitrary cell of $X \times X$*

- *if e^p is a source of an arrow $e^p \to e^{p+1}$ in X then there is an arrow $e^p \times f^q \to e^{p+1} \to f^q$ in $X \times X$;*
- *if e^p is critical and f^q is a source of an arrow $f^q \to f^{q+1}$ in X then there is an arrow $e^p \times f^q \to e^p \times f^{q+1}$ in $X \times X$;*
- *if both e^p and f^q are critical in X then $e^p \times f^q$ is critical in $X \times X$.*

The proof for Proposition 4.4.1 is easy.

In order to give an explicit formula for a contracting homotopy on the cellular chain complex $C_*(X)$ constructed in section 4.4.1, we will construct a contracting vector field on the corresponding cubical G-space.

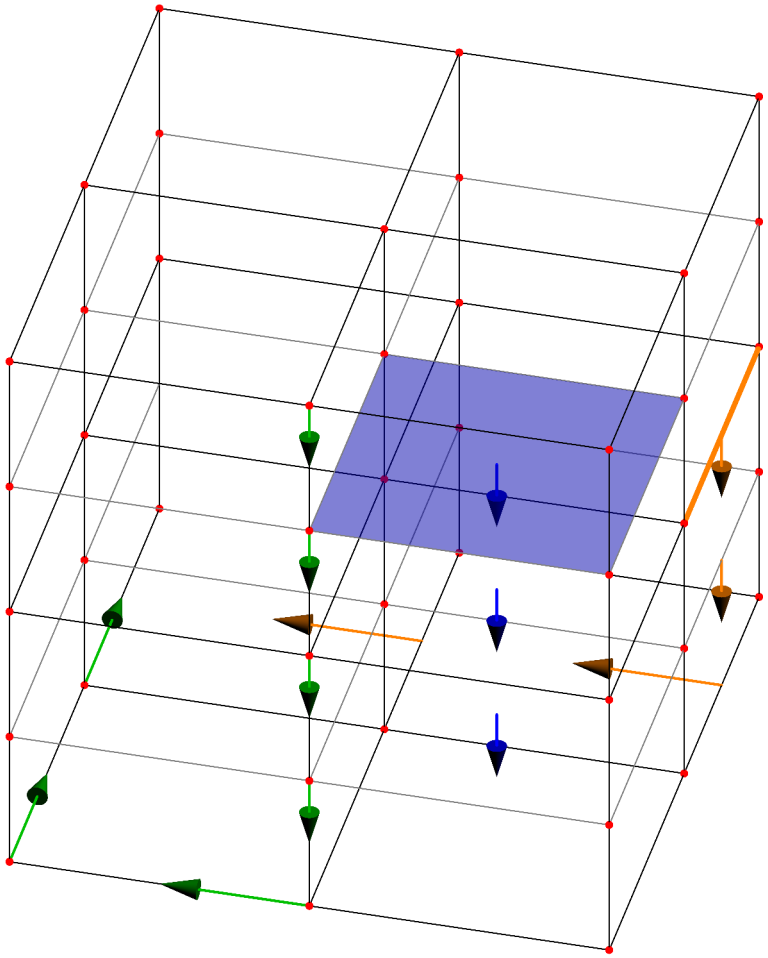


Figure 4.3: A contracting vector field on $X = R^3$

Suppose that, for the action of a crystallographic G on the Euclidean space $X = \mathbb{R}^n$, there is a G-lattice L with specified G-full basis $B_L = \{v_1, v_2, \dots v_n\}$. By Theorem 4.3.2, there is a single orbit of n-cells represented by σ_0 the open n-dimensional parallelepiped determined by vector in B_L and centered at $c_0 = (v_1/2, v_2/2, \dots, v_n/2)$. Let σ^k be a k-cell which is identified with its center $c(\sigma^k) = (x_1, x_2, \dots, x_n)$. The cell σ^k is a k-face of an n-cell. By definition 4.2.2 of a G-full basis, the center $c(\sigma^k)$ is an integral combination of vectors $\{v_1/2, v_2/2, \dots v_n/2\}$, i.e. there are some integers $\alpha_1, \alpha_2, \dots, \alpha_n$ such that

$$c(\sigma^k) = (\alpha_1 v_1 + \alpha_2 v_2 + \cdots + \alpha_n v_n).$$

Since the dimension of σ^k is k then there are exactly k of α_i which are odd.

We now define a contracting vector field as a map

$$V : X^{(n)} \to X^{(n+1)}$$

which sends σ^k to a $(k+1)$-cell t^{k+1} by following the algorithm below.

Algorithm 4.4.2 Contracting Vector Field

Input: a vector $(x_1, x_2, \ldots, x_n)$ which is the center of a k-cell.

Output: a vector $(y_1, y_2, \ldots, y_n)$ which is a center of a $(k+1) - cell$.

Procedure:

```
1: for i from 1 to n do
2:     if not x_i=0 then
3:         if x_i is odd then
               return Empty.
4:         else
5:             y := x;
6:             y[i] := y[i] − IntSign(x[i]).
7:         end if
8:     end if
9: end for
       return y.
```

A contracting homotopy on $C_*(X)$ can be constructed by following the retracting V-paths given by Algorithm 4.4.2.

4.5 Free resolutions for crystallographic groups

The construction of a free $\mathbb{Z}G$-resolution follows from Section 2.1.1, where G is a cubical crystallographic group. Notice that, by Lemma 4.3.1, all stabilizer subgroups of G are finite which is an advantage since HAP [11] can produce a free resolution for each finite group. The algorithm for constructing such a free resolution is described below.

Algorithm 4.5.1 Free resolution for a crystallographic group

Input: a crystallographic group G generated by a set of crystallographic matrices and positive integer n.

Output: If Algorithm 4.3.1 successfully finds a G-full basis then it outputs the first $n+1$ terms of a free $\mathbb{Z}G$-resolution. If G goes through the check for not admitting a parallelepiped fundamental region it outputs "false". Otherwise it outputs "fail".

Procedure:

1: Attempt to find a G-full basis B for a G-lattice by using Algorithm 4.3.1.
2: **if** failed for finding B **then**
return fail.
3: **else if** false for finding B **then**
return the action of G does not admit a cubical fundamental domain.
4: **else**
5: Construct a $\mathbb{Z}G$-equivariant CW-space $C_*(X)$ by using Algorithm 4.4.1 which inputs the basis B and a set of generators of G.
6: Construct a free $\mathbb{Z}G$ resolution R_*^G for group G as in Algorithm 2.1.1.
return R_*^G.
7: **end if**

4.6 Cup product, cohomology ring structure and a proof for Proposition 4.1.1

Recall that the cup product in cohomology is an operation which turns the cohomology groups of a group G into a graded ring: $H^*(G,\mathbb{Z}) = \bigoplus_{k\geq 0} H^k(G,\mathbb{Z})$. For a crystallographic group G for which Algorithm 4.3.1 successfully finds a cubical fundamental region by using Algorithm 4.3.1, a cup product can be constructed using Algorithm 4.4.2. The following GAP session illustrates some computations of such a cup product for group G given in Proposition 4.1.1. The session also proves Proposition 4.1.1.

GAP session

```
gap> gens:=[[[1,0,0,0],[0,-1,0,0],[0,0,1,0],[0,0,1/2,1]],
> [[-1,0,0,0],[0,-1,0,0],[0,0,1,0],[0,1/2,1/2,1]],
> [[-1,0,0,0],[0,-1,0,0],[0,0,1,0],[0,1/2,1/2,1]],
```

```
> [[1,0,0,0],[0,1,0,0],[0,0,1,0],[1,0,0,1]],
> [[1,0,0,0],[0,1,0,0],[0,0,1,0],[0,1,0,1]]];;
gap>  G:=Group(gens);;
gap> B:=CrystGFullBasis(G);
[ [ [ 1, 0, 0 ],[ 0, 1/2, 0 ],[ 0, 0, 1/2 ] ],[ 1/2, 1/4, 1/4 ] ]
gap> C:=CrystGcomplex(gens,B,1);
Non-free resolution in characteristic
0 for <matrix group of size infinity with 5 generators> .
gap> S:=HomToIntegers(R);
Cochain complex of length 10 in characteristic 0 .
gap>  Cohomology(S,0);
[ 0 ]          # H^0(G,Z) is generated by 1
gap>  Cohomology(S,1);
[ 0 ]          # H^1(G,Z) is generated by x, deg(x) = 1
gap>  Cohomology(S,2);
[ 2, 2 ]       # H^2(G,Z) is generated by y, z s.t
               # deg(y)=deg(z)=2 and 2y=2z=0
gap>  Cohomology(S,3);
[ 2 ]          # H^3(G,Z) is generated by t: deg(t)=3
               and 2t = 0
gap>  Cohomology(S,4);
[ ]            # H^n(G,Z) = 0 forall n > 3
gap>  IntegralCupProduct(C,[1],[1],1,1);
[ 0, 0 ]       # x^2 = 0
gap>  IntegralCupProduct(C,[1],[1,0],1,2);
[ 1 ]          # xy = t
gap>  IntegralCupProduct(C,[1],[0,1],1,2);
[ 0 ]          # xz = 0
```

Since $H^n(G,\mathbb{Z}) = 0$ for all $n \geq 4$ then $xt = y^2 = yz = yt = z^2 = zt = t^2 = 0$.

By the above GAP session, we also have

$$x^2 = 0, \quad xy = t, \quad xz = 0.$$

Therefore, the cohomology ring is

$$H^*(G, Z) = Z[x, y, z]/J$$

where $J = \langle x^2, xz, y^2, yz, z^2, 2y, 2z \rangle$ and $deg(x) = 1,\ deg(y) = deg(z) = 2$.

4.7 Three dimensional Bieberbach groups

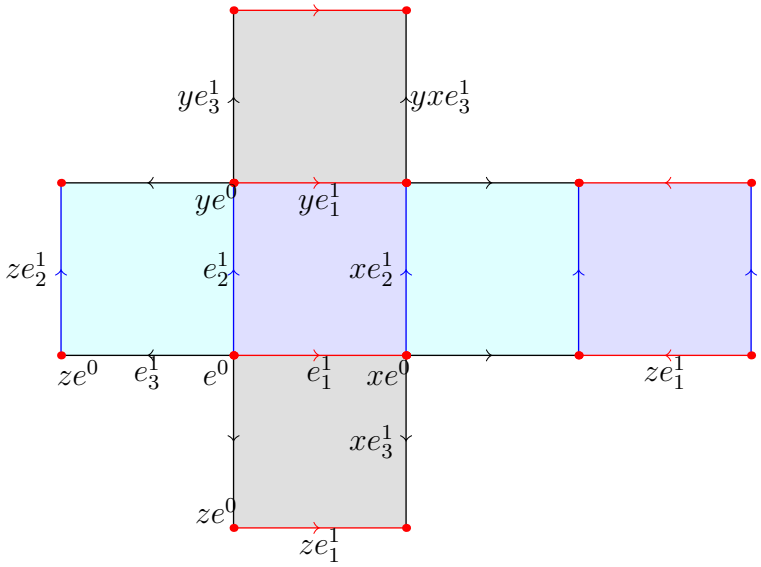
In dimension 3, there are ten Bieberbach groups eight of which admit a cubical fundamental region. In this section, we will give a full list of those eight fundamental regions and also compute the cohomology ring structures based on the contracting homotopy of Algorithm 4.4.2. For those eight groups, the fundamental region is a cube F. By coloring the 2-faces of F, those having the same color are identified under the action of the group. By coloring and adding arrows to the edges, edges with the same color are identified and the orientations are represented by arrows. Vertices with the same color are identified. For convenience, we flatten the cube by cutting seven edges (See the pictures below for details).

4.7.1 First group

Let G be the crystallographic group generated by

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}.$$

The fundamental region produced by Algorithm 4.3.1 is



and corresponds to the presentation

$$G = \langle x, y, z : \ \emptyset \rangle.$$

The group cohomology are

$$H^0(G, \mathbb{Z}) = \mathbb{Z}$$

$$H^1(G, \mathbb{Z}) = \mathbb{Z}^3$$

$$H^2(G, \mathbb{Z}) = \mathbb{Z}^3$$

$$H^3(G, \mathbb{Z}) = \mathbb{Z}$$

$$H^n(G, \mathbb{Z}) = 0, \quad \text{for all } n \geq 4.$$

The cohomology ring is:

$$H^*(G, \mathbb{Z}) = \mathbb{Z}[x, y, z]$$

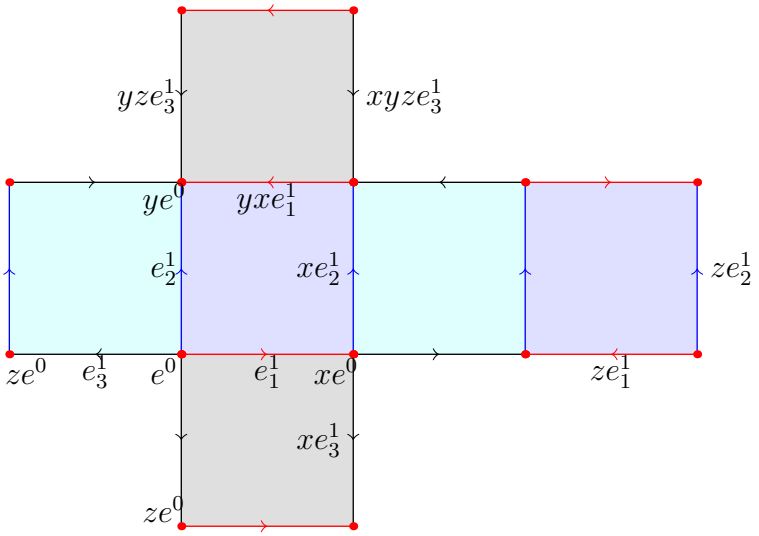
where $deg(x) = deg(y) = deg(z) = 1$.

4.7.2 Second group

Let G be the crystallographic group generated by

$$\begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1/2 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}.$$

The fundamental region produced by Algorithm 4.3.1 is



and corresponds to the presentation

$$G = \langle x, y, z | yxy^{-1}x^{-1} = zxz^{-1}x^{-1} = zyzy^{-1} \rangle.$$

The group cohomology are

$$H^0(G, \mathbb{Z}) = \mathbb{Z}$$

$$H^1(G, \mathbb{Z}) = \mathbb{Z}$$

$$H^2(G, \mathbb{Z}) = \mathbb{Z}_2^2 \oplus \mathbb{Z}$$

$$H^3(G, \mathbb{Z}) = \mathbb{Z}$$

$$H^n(G, \mathbb{Z}) = 0, \quad \text{for all } n \geq 4.$$

The cohomology ring is:

$$H^*(G, \mathbb{Z}) = \mathbb{Z}[x, y, z, t]/\langle x^2, xy, xz, y^2, yz, yt, z^2, zt, t^2, 2y, 2z \rangle$$

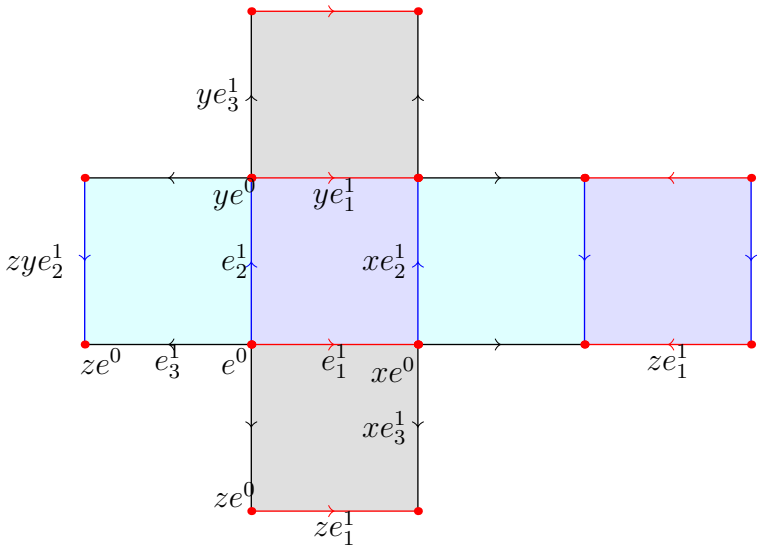
where $deg(x) = 1$, $deg(y) = deg(z) = deg(t) = 2$.

4.7.3 Third group

Let G be the crystallographic group generated by

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/2 & 1 \end{pmatrix}.$$

The fundamental region produced by Algorithm 4.3.1 is



and corresponds to the presentation

$$G = \langle x, y, z : \ yxy^{-1}x^{-1} = zxz^{-1}y^{-1} = zy^{-1}z^{-1}y^{-1} = 1 \rangle.$$

The group cohomology are

$$H^0(G, \mathbb{Z}) = \mathbb{Z}$$

$$H^1(G, \mathbb{Z}) = \mathbb{Z}^2$$

$$H^2(G, \mathbb{Z}) = \mathbb{Z}_2 \oplus \mathbb{Z}$$

$$H^3(G, \mathbb{Z}) = \mathbb{Z}_2$$

$$H^n(G, \mathbb{Z}) = 0, \quad \text{for all } n \geq 4.$$

Cohomology ring

$$H^*(G,\mathbb{Z}) = \mathbb{Z}[x,y,z]/\langle x^2, y^2, yz, z^2, 2z\rangle,$$

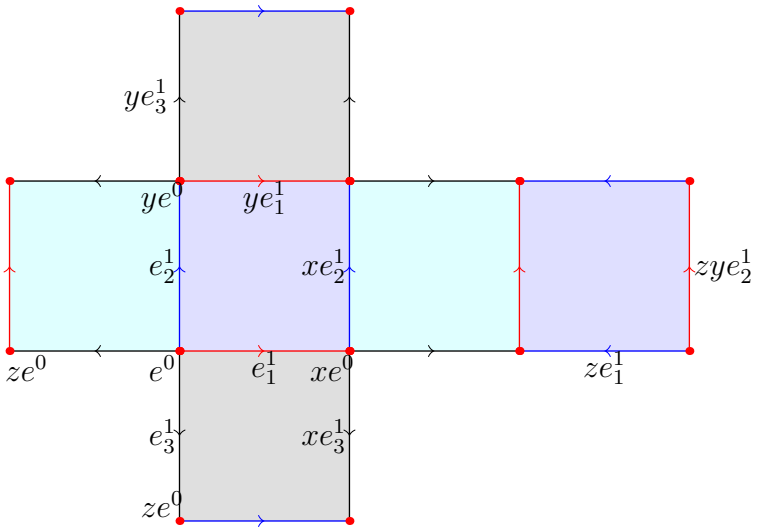
where $deg(x) = deg(y) = 1,\ deg(z) = 2.$

4.7.4 Fourth group

Let G be the crystallographic group generated by

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/2 & 1 \end{pmatrix}.$$

The fundamental region produced by Algorithm 4.3.1 is



and corresponds to the presentation

$$G = \langle x, y, z :\ yxy^{-1}x^{-1} = zyz^{-1}x^{-1} = zxz^{-1}y^{-1} = 1\rangle.$$

Group cohomology:

$$H^0(G,\mathbb{Z}) = \mathbb{Z}$$

$$H^1(G,\mathbb{Z}) = \mathbb{Z}^2$$

$$H^2(G,\mathbb{Z}) = \mathbb{Z}$$

$$H^3(G,\mathbb{Z}) = \mathbb{Z}_2$$

$$H^n(G,\mathbb{Z}) = 0, \quad \text{for all } n \geq 4.$$

Cohomology ring

$$H^*(G,\mathbb{Z}) = \mathbb{Z}[x,y,z]/\langle x^2, xy+2z, 2xz, y^2, yz, z^2\rangle,$$

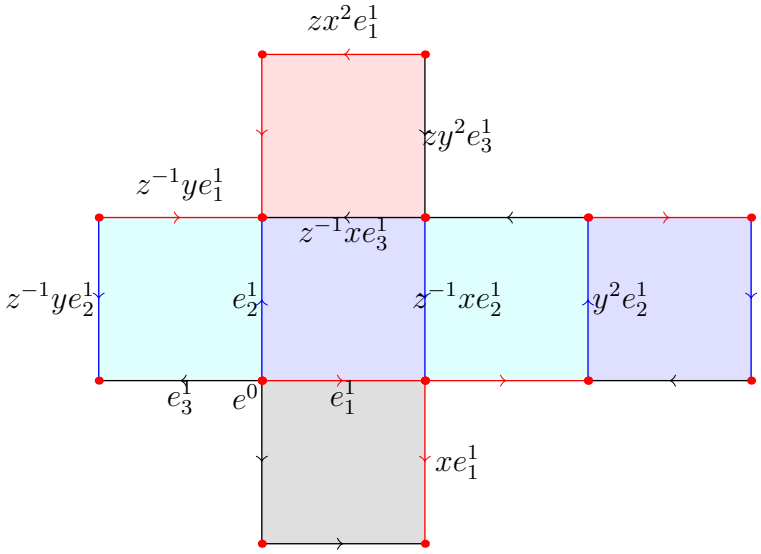
where $deg(x) = deg(y) = 1,\ deg(z) = 2.$

4.7.5 Fifth group

Let G be the crystallographic group generated by

$$\begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -1/2 & 0 & 1/2 & 1 \end{pmatrix}, \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1/2 & 0 & 1/2 & 1 \end{pmatrix}, \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & -1/2 & 0 & 1 \end{pmatrix}.$$

The fundamental region produced by Algorithm 4.3.1 is



and corresponds to the presentation

$$G = \langle x,y,z : \ zy^{-1}zx^{-1} = xzyz = x^2y^{-2} = 1\rangle.$$

Group cohomology:

$$H^0(G,\mathbb{Z}) = \mathbb{Z}$$

$$H^1(G,\mathbb{Z}) = 0$$

$$H^2(G,\mathbb{Z}) = \mathbb{Z}_4^2$$

$$H^3(G,\mathbb{Z}) = \mathbb{Z}$$

$$H^n(G,\mathbb{Z}) = 0, \quad \text{for all } n \geq 4.$$

Cohomology ring

$$H^*(G,\mathbb{Z}) = \mathbb{Z}[x,y,z]/\langle x^2, xy, xz, y^2, yz, 4x, 4y, z^2\rangle,$$

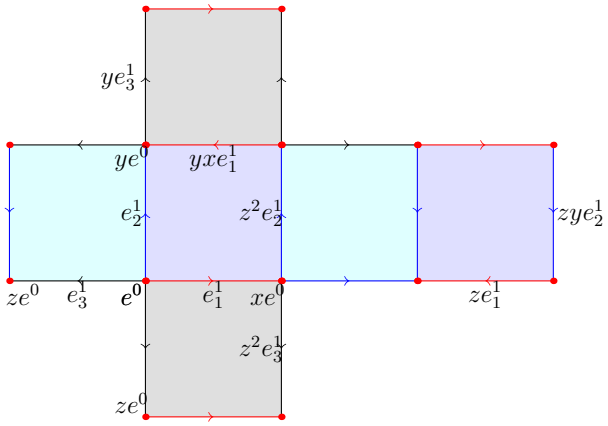
where $deg(x) = deg(y) = 2, deg(z) = 3$.

4.7.6 Sixth group

Let G be the crystallographic group generated by

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1/2 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/2 & 1 \end{pmatrix}.$$

The fundamental region produced by Algorithm 4.3.1 is



and corresponds to the presentation

$$G = \langle x, y, z : \ yx^{-1}y^{-1}x^{-1} = zxz^{-1}x^{-1} = zy^{-1}z^{-1}y^{-1} = 1 \rangle.$$

Group cohomology:

$$H^0(G, \mathbb{Z}) = \mathbb{Z}$$

$$H^1(G, \mathbb{Z}) = \mathbb{Z}$$

$$H^2(G, \mathbb{Z}) = \mathbb{Z}_2^2$$

$$H^3(G, \mathbb{Z}) = \mathbb{Z}_2$$

$$H^n(G, \mathbb{Z}) = 0, \quad \text{for all } n \geq 4.$$

Cohomology ring

$$H^*(G, \mathbb{Z}) = \mathbb{Z}[x, y, z]/\langle x^2, 2xy, xz, y^2, z^2, yz, 2y, 2z \rangle,$$

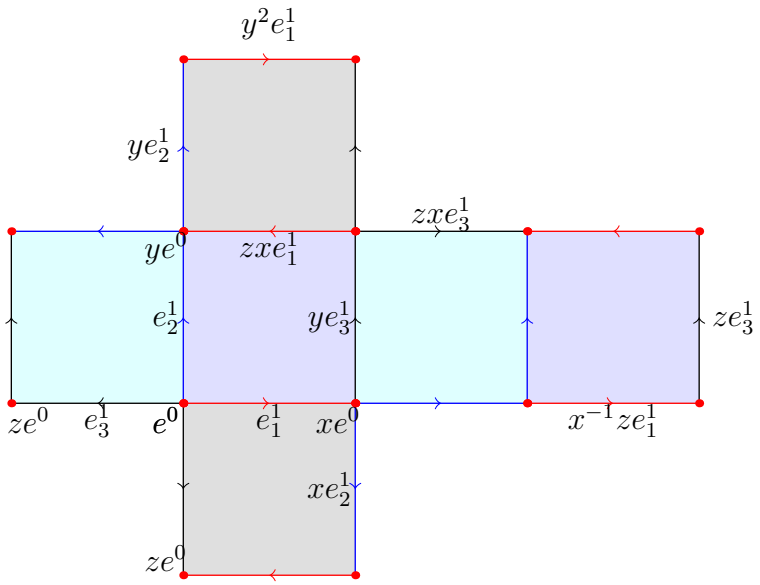
where $deg(x) = 1,\ deg(y) = deg(z) = 2.$

4.7.7 Seventh group

Let G be the crystallographic group generated by

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1/2 & 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1/2 & 1/2 & 1 \end{pmatrix}, \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1/2 & 1/2 & 1 \end{pmatrix}.$$

The fundamental region produced by Algorithm 4.3.1 is



and corresponds to the presentation

$$G = \langle x, y, z : \ yx^{-1}z^{-1}x^{-1} = zx^{-1}y^{-1}x^{-1} = z^2y^{-2} = 1 \rangle.$$

Group cohomology:

$$H^0(G, \mathbb{Z}) = \mathbb{Z}$$

$$H^1(G, \mathbb{Z}) = \mathbb{Z}$$

$$H^2(G, \mathbb{Z}) = \mathbb{Z}_4$$

$$H^3(G, \mathbb{Z}) = \mathbb{Z}_2$$

$$H^n(G, \mathbb{Z}) = 0, \quad \text{for all } n \geq 4.$$

Cohomology ring

$$H^*(G, \mathbb{Z}) = \mathbb{Z}[x, y]/\langle x^2, 2xy, y^2, 4y \rangle,$$

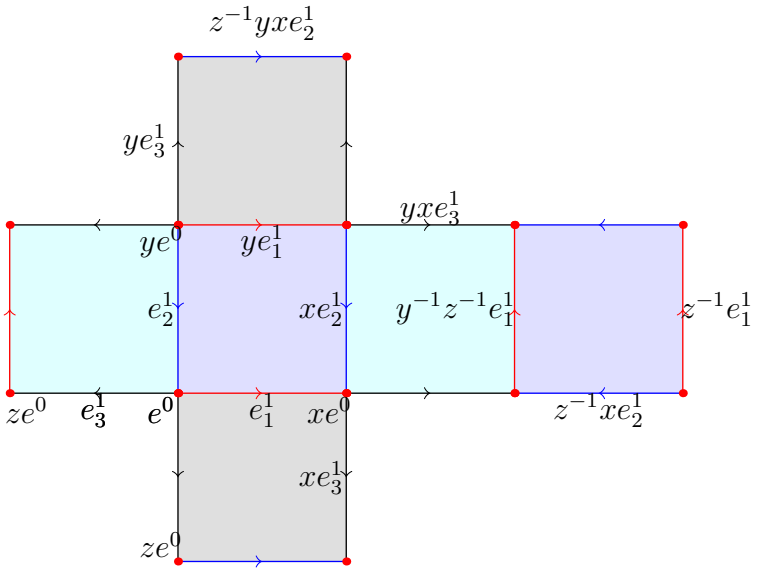
where $deg(x) = 1, \ deg(y) = 2$.

4.7.8 Eighth group

Let G be the crystallographic group generated by

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/4 & 1 \end{pmatrix}.$$

The fundamental region produced by Algorithm 4.3.1 is



and corresponds to the presentation

$$G = \langle x, y, z : \ yx^{-1}y^{-1}x = zyz^{-1}x^{-1} = zxz^{-1}y = 1 \rangle.$$

Group cohomology:

$$H^0(G, \mathbb{Z}) = \mathbb{Z}$$

$$H^1(G, \mathbb{Z}) = \mathbb{Z}$$

$$H^2(G, \mathbb{Z}) = \mathbb{Z}_2 \oplus \mathbb{Z}$$

$$H^3(G, \mathbb{Z}) = \mathbb{Z}$$

$$H^n(G, \mathbb{Z}) = 0, \quad \text{for all } n \geq 4.$$

Cohomology ring

$$H^*(G, \mathbb{Z}) = \mathbb{Z}[x, y, z]/\langle x^2, xy, y^2, 2y, yz, z^2 \rangle,$$

where $deg(x) = 1,\ deg(y) = deg(z) = 2$.

4.7.9 Ninth group

Let G be the crystallographic group generated by

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ -1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/3 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}.$$

We are going to prove that the crystallographic group G generated by these matrices does not admit a cubical fundamental region.

Let $T < G$ be the translation subgroup and $P = G/T$ be the point group. Let $\mathbb{T}^3 = \mathbb{R}^3/T$ the 3-torus got from $\mathbb{R}^n$ by killing the action of T. Then P acts on $\mathbb{T}^3$. We endow $\mathbb{T}^3$ with a CW-structure obtained from a fundamental region F for the action of G on $\mathbb{R}^n$. Thus $\mathbb{T}$ has a divisor of $|P|$ n-cells, each n-cell being isometric to F.

Let Isom($\mathbb{T}^3$) be the group of isometries of $\mathbb{T}^3$ that preserve the CW-structure. There is an exact sequence $A \rightarrowtail \text{Isom}(\mathbb{T}^3) \twoheadrightarrow \text{Isom}(\mathbb{T}^3)/A$ where A is a finite abelian group of "translations".

Then P is a subgroup of Isom($\mathbb{T}^3$) and, moreover, P intersects trivially with A.

In this case, $P \simeq C_3$ the cyclic group of order 3 and $\mathbb{T}^3$ is the 3-torus with a CW-structure involving either one or three isometric 3-cells. Moreover, P contains some element of order 3 that is not a translation (i.e. is not in A). Suppose that the 3-cells are cube.

If there are three 3-cells then $\mathbb{T}^3$ must be a quotient of three cubes in a row (See Figure 4.4). One can see that there is no non-translation of order 3 of $\mathbb{T}^3$.

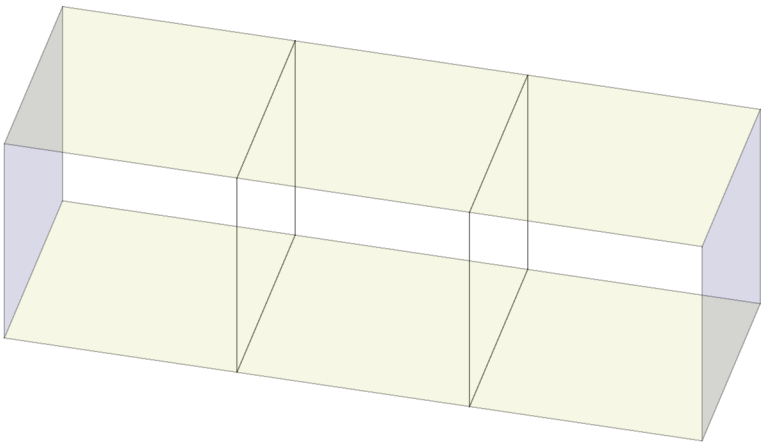


Figure 4.4

If there is a single 3-cell in $\mathbb{T}^3$ then P has to be a subgroup of G that is contrary to the information given by the generators of group G.

Hence, G does not admit a cubical fundamental region.

4.7.10 Tenth group

Let G be the crystallographic group generated by

$$\begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/2 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 0 & 0 \\ -1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/3 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}.$$

By the same argument as the ninth group, we can prove that this group does not admit a cubical fundamental region.

4.8 Experimental results

The experimental results given in this section are the application of our subpackage on the list of space groups in GAP 's library. GAP provides a library of all crystallographic groups of dimensions 2, 3, and 4; this covers many of the data that have been listed in the book [5]. It has been brought into GAP format by Volkmar

Felsch. Table 4.1 shows experimental result for Algorithm 4.3.1. The algorithm outputs "found", "fail" or "non-identity Gramian matrix" for each group G. Thus,

$Dimension$	$Total$	$Found$	$Fail$	$Gram(P) \neq I_n$
2	17	12	5	0
3	219	114	19	86
4	4783	1996	716	2071

Table 4.1: Statistical result for dimension 2,3 and 4.

for instance, the algorithm succeeds in computing a cubical fundamental region for 1996 of 4783 4-dimensional crystallographic groups. The current implementation of Algorithm 4.3.1 works only for group with $Gram(P) = I_n$. But recall that, under changing of basis represented by an invertible matrix Q, the gramian matrix will be changed by a matrix congruence to $Q^t\,\mathrm{Gram}(P)Q$. So readers could find more cubical crystallographic groups by conjugating such groups by sufficient matrices.

The following GAP sessions will give a comparison of our method to Wall's and Röder's.

4.8.1 Comparison to Wall's extension method

The GAP session below illustrates Wall's extension method. It takes 58183 ms to construct a free resolution for the 3550th group in the list [5] with the rank 1, 6, 18, 38, 66, 102 respect to six first degrees.

GAP session

```
gap> G:=SpaceGroup(4,3550);;
gap> H:=Image(IsomorphismPcpGroup(G));;
gap> R:=ResolutionAlmostCrystalGroup(H,5);;
gap> time;
58183
gap> List([0..5],R!.dimension);
[ 1, 6, 18, 38, 66, 102 ]
```

Our new method give a free resolution in 3104 ms and the corresponding rankss are 1, 3, 6, 7, 7, 7.

GAP session

```
gap> R:=ResolutionCubicalCrystGroup(SpaceGroup(4,3550),5);
Resolution of length 5 in characteristic 0 for <matrix group with
7 generators> .

gap> time;
3104
gap> List([0..5],R!.dimension);
[ 1, 3, 6, 7, 7, 7 ]
```

In this example, our method provides a free resolution much smaller and the construction is about 20 times faster than Wall's.

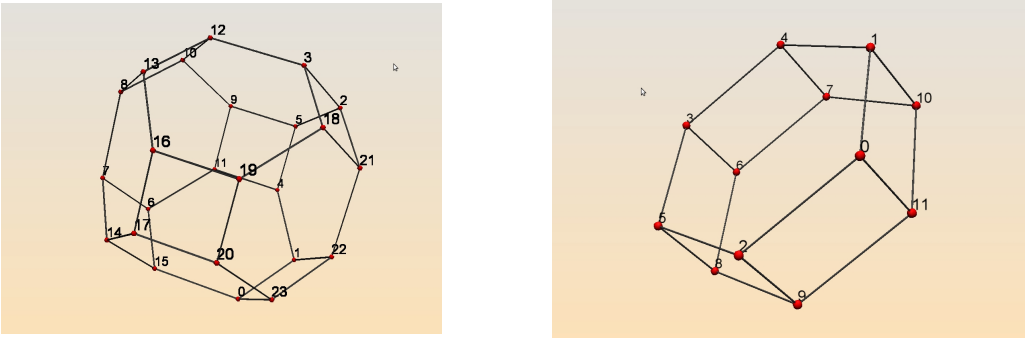
4.8.2 Comparison to Röder's method

As we mentioned in the Introduction, Röder's method may produce a fundamental region in a complicated shape and deeply depends on the choice of the starting point for a Dirichlet-Voronoi construction. The GAP session below illustrate the difficulty of finding a good choice of the starting point.

GAP session

```
gap> G:=SpaceGroup(3,2);;
gap> fd:=FundamentalDomainStandardSpaceGroup([0,0,0],G);
Error, center point not in general position
gap> fd:=FundamentalDomainStandardSpaceGroup([1/2,1/2,1/2],G);
Error, center point not in general position
gap> fd:=FundamentalDomainStandardSpaceGroup([1/2,1/3,1/4],G);
<polymake object>
gap> Polymake(fd,"VISUAL_GRAPH");
<Figure 4.3 (a)>
gap> fd:=FundamentalDomainStandardSpaceGroup([1/4,1/4,1/4],G);
<polymake object>
gap> Polymake(fd,"VISUAL_GRAPH");
```

<Figure 4.3 (b)>



(a) Fundamental region with respect to [1/4,1/4,1/4]

(b) Fundamental region with respect to [1/2,1/3,1/4]

Figure 4.5: Examples of fundamental domains

In Röder's method, the choice of the starting point for a Dirichlet-Voronoi construction is difficult to get a nice fundamental region in order to find a free resolution and furthermore ring structure. It is one of the reasons that motivates this chapter.

GAP session

```
gap> G:=SpaceGroup(3,2);;
gap> B:=CrystGFullBasis(G);
[ [ [ 1, 0, 0 ], [ 0, 1, 0 ], [ 0, 0, 1/2 ] ], [ 1/2, 1/2, 1/4 ] ]
gap> R:=ResolutionCubicalCrystGroup(SpaceGroup(3,2),5);
Resolution of length 5 in characteristic 0 for <matrix group with
4 generators> .

gap> time;
280
```

Our method gives us a fundamental region F which is a parallelepiped centered at $(1/2, 1/2, 1/4)$ and determined by 3 vectors $\{(1,0,0),(0,1,0),(0,0,1/2)\}$. A free resolution is obtained in only 280 millisecond. And also on this resolution, our method gives a explicit contracting homotopy which is not given by Röder's method.

Bibliography

[1] Alejandro Adem and Nadim Naffah. On the cohomology of $SL(2, \mathbb{Z}[1/p])$. *London Mathematical society lecture note series*, pages 1–9, 1998. (Cited on pages 3 and 26.)

[2] Alejandro Adem et al. Compatible actions and cohomology of crystallographic groups. *Journal of Algebra*, 320(1):341–353, 2008. (Cited on page 3.)

[3] Szczepaski Andrzej. *Geometry of Crystallographic Groups*, volume 4 of *Algebra and Discrete Mathematics*. World Scientific, 2012. (Cited on page 41.)

[4] P. Brendel, P. Dlotko, G. Ellis, M. Juda, and M. Mrozek. Computing fundamental groups from point clouds. *preprint*, 2014. (`http://hamilton.nuigalway.ie/preprints/fundamental.pdf`). (Cited on page 18.)

[5] Harold Brown, Rolf Bülow, Joachim Neubüser, Hans Wondratschek, and Hans Zassenhaus. *Crystallographic Groups of Four-Dimensional Space*. John Wiley, New York, 1978. (Cited on pages 74 and 75.)

[6] Kenneth S. Brown. *Cohomology of Groups*, volume 87 of *Graduate Texts in Mathematics*. Springer, 2nd edition, 1994. (Cited on pages 9, 10, and 11.)

[7] V. A Churkin. Crystallographic groups with two lattices and metric lie algebras. *Algebra and logic*, 52.6), 2014. (Cited on page 3.)

[8] Marston Conder. Group actions on the cubic tree. *J. Algebraic Combin.*, 1(3):209–218, 1992. (Cited on page 19.)

[9] Karel Dekimpe and Nansen Petrosyan. Homology of hantzsche-wendt groups. In *Discrete Groups and Geometric Structures*, volume 501, pages 87–102. American Mathematical Society, Providence, RI, 2009. (Cited on page 3.)

[10] Graham Ellis. Computing group resolutions. *Journal of Symbolic Computation*, 38(3):1077–1118, 2004. (Cited on page 42.)

[11] Graham Ellis. *HAP – Homological Algebra Programming, Version 1.10.13*, 2013. (`http://www.gap-system.org/Packages/hap.html`). (Cited on pages ix, 3, 9, 11, 26, 38, 42, 43, 44, 60, 83, and 84.)

[12] Graham Ellis, James Harris, and Emil Sköldberg. Polytopal resolutions for finite groups. *Journal für die reine und angewandte Mathematik (Crelles Journal)*, 598, 2006. (Cited on pages 15, 26, 42, and 44.)

[13] Graham Ellis and Fintan Hegarty. Computational homotopy of finite regular cw-spaces. *Journal of Homotopy and Related Structures*, 9.1, 2014. (Cited on page 18.)

[14] Robin Forman. Morse theory for cell complexes. *Advances in Mathematics*, 134 (1), 1998. (Cited on page 16.)

[15] Robin Forman. A users guide to discrete morse theory. *Sm. Lothar. Combin*, 48, 2002. (Cited on pages 16, 17, and 18.)

[16] The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.7.2*, 2013. (`http://www.gap-system.org`). (Cited on pages ix, 3, 26, 38, and 43.)

[17] Moritz Groth. Course on algebraic topology i, 2013. (`http://www.math.ru.nl/ mgroth/teaching/algtopI14.html`). (Cited on page 6.)

[18] Soren Hansen. Lecture notes on algebraic topology, 2005. (`http://www.math.ksu.edu/ hansen/CWcomplexes.pdf`). (Cited on pages 4, 5, and 6.)

[19] Allen Hatcher. *Algebraic topology*. Cambridge University Press, New York, NY, USA, 2010. (Cited on pages 4, 5, 6, 7, and 8.)

[20] Kevin Hutchinson. A refined bloch group and the third homology of of a field. *Journal of Pure and Applied Algebra*, 217(11):2003–2035, 2013. (Cited on page 3.)

[21] Kevin Hutchinson. The second homology of sl_2 of s-integers. *arXiv preprint arXiv:1412.0953*, 2014. (Cited on page 3.)

[22] D. Jones. *A general theory of polyhedral sets*. Dissertationes Math. 1988. (Cited on pages 16, 17, and 18.)

[23] Dimitry Kozlov. *Combinatorial Algebraic Topology*, volume 21 of *Algorithms and Computation in Mathematics*. Springer, 2008. (Cited on page 8.)

[24] William S. Massey. *A basic course in algebraic topology*. Springer, 1991. (Cited on page 8.)

[25] James R. Munkres. *Elements of Algebraic Topology*. Westview Press, 1996. (Cited on page 4.)

[26] Marc Röder. *HAPcryst, A HAP extension for crytallographic groups, Version 0.1.11*, 2013. (`http://www.gap-system.org/Packages/hapcryst.html`). (Cited on pages 3, 43, and 45.)

[27] Ana Romero and Francis Sergeraert. Discrete vector fields and fundamental algebraic topology. *arXiv preprint arXiv:1005.5685*, 2010. (Cited on pages 16 and 17.)

[28] J. S. Serre. *Trees*. Monograph in Mathematics. Springer, 2002. (Cited on pages 29, 30, and 33.)

[29] Andrzej Szczepaski. *Geometry of Crystallographic Groups*. World Scientific, 2012. (Not cited.)

[30] Rubn Snchez-Garca. Bredon homology and equivariant k-homology of sl (3, z). *Journal of Pure and Applied Algebra*, 212(5):1046–1059, 2008. (Cited on page 42.)

[31] C. T. C. Wall. Resolutions for extensions of groups. *Mathematical Proceedings of the Cambridge Philosophical Society*, 57(2), 1961. (Cited on pages 3, 14, 15, 26, and 44.)

[32] Charles A. Weibel. *An introduction to homological algebra*, volume 38 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, Cambridge, 1994. (Cited on page 7.)

[33] J.H.C. Whitehead. Simple homotopy types. 72, 1950. (Cited on page 18.)

[34] F. Williams and R. Wisner. Cohomology of certain congruence subgroups of the modular group. *Proc. Amer. Math. Soc.*, 126(5):1331–1336, 1998. (Cited on page 26.)

Chapter 5

Appendix: GAP code

5.1 Data types

1. HAP non-free $\mathbb{Z}G$-resolution [11]

A non-free $\mathbb{Z}G$-resolution is an exact sequence of $\mathbb{Z}G$-modules

$$\cdots \to A_n \to A_{n-1} \to \cdots \to A_1 \to A_0 \to \mathbb{Z}$$

in which the modules are not necessarily free. We can represent such a sequence (non-uniquely) by first choosing free modules M_k mapping onto the A_k and then lifting the homomorphisms to a sequence

$$\cdots \to M_n \to M_{n-1} \to \cdots \to M_1 \to M_0 \to \mathbb{Z}$$

of homomorphisms of free $\mathbb{Z}G$-modules. Note that in the lifted sequence homomorphisms will not in general square to zero.

A non-free ZG-resolution

$$\cdots \to M_n \to M_{n-1} \to \cdots \to M_1 \to M_0 \to \mathbb{Z}$$

is represented by a component object R with the following components.

- R!.dimension(k) is a function which returns the Z_G-rank of the module M_k.
- R!.boundary(k,j) is a function which returns the image in M_{k-1} of the j-th free generator of M_k.
- R!.homotopy(k,[i,g]) is a function which returns the image in M_{k+1}, under a contracting homotopy $M_k \to M_{k+1}$, of the element [[i,g]] in M_k. (The elements [[i,g]] freely generate M_k as an abelian group.) For most non-free resolutions a contracting homotopy is not constructed, in which case R!.homotopy is set equal to "fail".
- R!.elts is a (partial) list of (possibly duplicate) elements in G. In some cases R!.elts is a pseudo list and not a list.

- R!.group is the group in question (and could be a permutation group, matrix group, finitely presented group etc.).

- R!.stabilizer(k,j) returns the subgroup of G consisting of those elements that fix, up to sign, the i-th free generator of M_k.

- R!.action(k,j,g) is a function which returns +1 or -1 according to how the group element Elts[g] acts on the "orientation" of the j-th free generator in dimension k.

- R!.properties is a list of pairs ["name", value] where "name" is a string and value is a numerical or boolean value. Example pairs are: ["length", n] which records that there are n terms in the resolution; ["characteristic", p] which records the characteristic of $\mathbb{Z}$; ["reduced", true] which record that M_0 is isomorphic to $\mathbb{Z}G$; ["type","resolution"] which records the type of the object R.

The operation IsHapNonFreeResoluton(R) returns "true" for a non-free resolution.

2. HAP free $\mathbb{Z}G$-resolution [11]

A HAP free $\mathbb{Z}G$-resolution

$$\cdots \to M_n \to M_{n-1} \to \cdots \to M_1 \to M_0 \to \mathbb{Z}$$

is represented as a component object R with the following components:

- R!.dimension(k) is a function which returns the $\mathbb{Z}G$-rank of the module M_k.

- R!.boundary(k,j) is a function which returns the image in M_{k-1} of the j-th free generator of M_k.

- R!.homotopy(k,[i,g]) is a function which returns the image in M_{k+1}, under a contracting homotopy $M_k \to M_{k+1}$, of the element [[i,g]] in M_k.

- R!.elts is a (partial) list of (possibly duplicate) elements in G. In some cases R!.elts is a pseudo list (see below) and not a list.

- R!.group is the group G.

- R!.properties is a list of pairs ["name", value] where "name" is a string and value is a numerical or boolean value.

3. SL2Z Group

A SL2Z-group $G = SL(2, Z[1/m])$ is represented as a component object X with the following components:

- X!.name returns the text name of group G.
- X!.primes returns m.
- X!.GeneratorsOfMagmaWithInverses is a list of generators of group G.

4. Congruence Subgroup

A congruence subgroup of $G = SL2Z[1/m]$ of level p is represented as a component object X with the following components:

- X!.name returns the text name of G.
- X!.levels returns $[m, p]$.
- X!.GeneratorsOfMagmaWithInverses returns a list of generators of G.
- X!.DimensionOfMatrixGroup returns the dimension of matrices in G.

5.2 List of functions

1. **SL2ZTree(m,p)** (see GAP code 5.3.4)

   - Input: A pair of positive integers (m,p).
   - Output: A G-equivarient CW-space where G is the free product of two copies of SL(1/m) amalgamated by its congruence group of level p. This is a component object C with components
     - $C!.group$ returns the group $G = SL2(Z[1/mp])$.
     - $C!.elts$ returns a list of some of the elements of G.
     - $C!.dimension(n)$ returns the number of distinct G-orbits of n-dimensional cells in the G- equivariant CW-space $|C|$.
     - $C!.boundary(n,k)$ returns the boundary of the n-cell e^n_k.
     - $C!.stabilizer(n,k)$ returns the stabilizer subgroup of the n-cell E^n_k.
     - $C!.homotopy(n,[i,j])$ returns the maximal V-path from the n-cell $g_i e^n_j$ in the discrete vector field constructed in section 3.2.

   (See Algorithm 3.2.1.)

2. **ResolutionGTree(R,n)** (see GAP code 5.3.7)

   - Input:
     - a non-free $\mathbb{Z}G$-resolution C_* of $\mathbb{Z}$ together with free $\mathbb{Z}G_e$-resolution of $\mathbb{Z}$ for all stabilizers G_e with contracting homotopy, and
     - an integer $n \geq 0$.
   - Output: The first $n+1$ terms of a free $\mathbb{Z}G$-resolution R^G_* of $\mathbb{Z}$. Moreover, if the input C_* is together with a contracting homotopy then the output also together with a contracting homotopy.

   (See Algorithm 2.1.1.)

3. **ConjugateSL2ZGroup(G)** (see GAP code 5.3.2)

- Input: An Arithmetic group G=SL2Z(1/m) and a invertible 2x2 matrix P.
- Output: Conjugation group of G by matrix P.

4. **CongruenceSubgroup(m,p)** (see GAP code 5.3.3)

- Input: A pair of positive integers (m,p).
- Output: The congruence subgroup of SL2Z(1/m) of level p.

5. **SL2ZmElementsDecomposition(g,p)** (see GAP code 5.3.6)

- Input: A matrix g in $SL2Z(2, 1/pn)$ and a positive integer p.
- Output: A list of pairs (a, b) where a in $SL2Z(1/n)$ and b in $SL2Z(1/m)^P$.

6. **SL2ZResolution(m,n)** (see GAP code 5.3.8)

- Input: A pair of positive integers (m,n).
- Output: The first n+1 terms of a free $\mathbb{Z}G$-resolution of of $\mathbb{Z}$ together with a contracting homotopy.

  (See Algorithm 3.2.2.)

7. **TreeOfResolutionsToSL2Zcomplex(D,G)** (see GAP code 5.3.5)

- Input: A list of resolutions D and an arithmetic group $G = SL2Z[1/m]$.
- Output: A G-equivarient CW-space.

8. **CrystGFullBasis(arg)** (see GAP code 5.3.25)

- Input: A crystallographic group G.
- Output: If G admits a cubical fundamental region it outputs a G-full basis for a G-lattice L. If G goes through the check for not admitting a cubical fundamental region it outputs "false". Otherwise it outputs "fail".

  (See Algorithm 4.3.1.)

9. CrystGcomplex(gens,basis,check) (see GAP code 5.3.26)

- Input: A set F of crystallographic matrices, a G-full basis B and check=1.
- Output: G-equivalent CW-space for group G generated by F together with a contracting homotopy.

  (See Algorithm 4.4.1.)

10. ResolutionCubicalCrystGroup(G,n) (see GAP code 5.3.27)

- Input: A crystallographic group G and an positive integer n.
- Output: The first n+1 terms of a free ZG-resolution of Z together with a contracting homotopy.

  (See Algorithm 4.5.1.)

11. VectorToCrystMatrix(v) (see GAP code 5.3.10)

- Input: A n-dimensional vector v.
- Output: A pure translation crystallographic matrix .

12. CrystTranslationMatrixToVector(g) (see GAP code 5.3.11)

- Input: A pure translation crystallographic matrix g.
- Output: An n-dimensional vector v.

13. TranslationSubGroup(G) (see GAP code 5.3.12)

- Input: A crystallographic group G.
- Output: Translation subgroup of G.

14. Method g in G (see GAP code 5.3.13)

- Input: A matrix g and a translation subgroup G of a crystallographic group a positive integer p.
- Output: True if g in G.

15. IsCrystSameOrbit(arg) (see GAP code 5.3.14)

- Input: Two points u, v in R^n and a group G..
- Output: return True if u, v in the same orbit. Otherwise returns False.

16. CombinationDisjointSets(C) (see GAP code 5.3.15)

- Input: A list of k positive integers (a_i).
- Output: A 2-dimensional array L such that $0 \leq L[i][j] < a_j$.

17. IsCrystSufficientLattice(B,S) (see GAP code 5.3.17)

- Input: A lattice's basis B and a transversal S of the translation subgroup in crystallographic group G.
- Output: True if G acts on the lattice, otherwise return False.

18. CrystFinitePartOfMatrix(g) (see GAP code 5.3.18)

- Input: A crystallographic matrix g.
- Output: Finite part of g.

19. ResolutionBoundaryOfWordOnRight(R,n,W) (see GAP code 5.3.19)

- Input: A free resolution R, degree n, a list of word w.
- Output: The boundary of w respects to the right action.

20. CrystCubicalTiling(n) (see GAP code 5.3.20)

- Input: Dimension n.
- Output: A list of some cubical tiling in n-dimensional space.

21. CrystMatrix(M) (see GAP code 5.3.23)

- Input: nxn matrix M.
- Output: the crystallographic form of M..

22. FreeZGResolution(arg) (see GAP code 5.3.29)

- Input: A G-equivalent CW-space with resolutions for all stabilisers and a positive integer n.
- Output: The first n terms of a free ZG-resolution of Z. If the input together with a contracting homotopy then the output together with a contracting homotopy.

  (See Algorithm 2.1.1.)

5.3 GAP Code

5.3.1 RightTransversal(G,H)

```
#############################################################################
#0
#F  RightTransversal
##
##  Define a new method for computing the set of right cosets
##  of a congruence subgroup in its main group SL2Z[1/m]
##
##  Input: A pair of groups (G,H) where G is SL2Z(1/m)
##         and H is its congruence subgroup
##  Output: A set of right cosets of H in G
##
##
InstallOtherMethod(RightTransversal,
"Right Transversal for SL2Z(1/m) and its congruence subgroup",
[IsHAPRationalSpecialLinearGroup,IsMatrixGroup],
function(G,H)
local
    T,trans,NameG,i,j,m,n,p,S,d,enum;

    NameG:=Name(G);
    i:=Position(NameG,'/');
    j:=Position(NameG,']');
    m:=Int(NameG{[i+1..j-1]});
    n:=H!.levels[1];
    p:=H!.levels[2];
    if not n=m then
        return "two groups are not in the same level";
    fi;
    T:=[[1,0],[1,1]];
    S:=[[0,-1],[1,0]];

    enum:=EnumeratorByFunctions(CollectionsFamily( FamilyObj( G ) ),rec(
        ElementNumber:=function(enum,n)
        if n<p+1 then
            return CanonicalRightCountableCosetElement(H,T^n);
        else
            return CanonicalRightCountableCosetElement(H,S);
        fi;
    end,

    NumberElement:=function(enum,elm)
        local i;
        if elm*S^-1 in H then
            return p+1;fi;
        for i in [1..p] do
            if elm*T^-i in H then
                return i;
            fi;
        od;
    return fail;
    end,

    Length:=function(enum)
```

```
        return p+1;
    end,

    PrintObj:=function(enum)
        Print("RightTransversal of ",Name(H)," in ",Name(G));
    end,

    group:=G,
    subgroup:=H,
    ));

    SetIsSSortedList( enum, true );
    return enum;
end);
##
################### end of RightTransversal ##############################
```

5.3.2 ConjugateSL2ZGroup(H,P)

```
#############################################################################
#0
#F  ConjugateSL2ZGroup
##
##  This function will create a conjugation of group SL2Z[1/m]
##  by a matrix P by giving its generator and also set name for that
##  group of conjugation.
##
##  Input: An Arithmetic group G=SL2Z(1/m) and
##         a invertible 2x2 matrix P
##  Output: Conjugation group of G by matrix P
##
##
InstallGlobalFunction(ConjugateSL2ZGroup,
function(H,P)
local m,gens,i,j, G,gensG,NameH,p;
    p:=P[2][2];
    if H=SL(2,Integers) then
        return SL2Z(p);
    fi;
    NameH:=Name(H);
    i:=Position(NameH,'/');
    j:=Position(NameH,']');
    m:=Int(NameH{[i+1..j-1]});

    gens:=GeneratorsOfGroup(SL2Z(1/m));
    gensG:=List(gens,x->P*x*(P^-1));
    G:=Group(gensG);
    SetName(G,Concatenation("SL(2,Z[",String(1/m),"])^",
                                            String(P))  );
    G!.coprimes:=[m,p];
    SetIsHAPRationalMatrixGroup(G,true);
    SetIsHAPRationalSpecialLinearGroup(G,true);

    return G;
end);
################### end of ConjugateSL2ZGroup ############################
```

5.3.3 CongruenceSubgroup(m,p)

```
############################################################################
#0
#F  CongruenceSubgroup
##
##  This function will create a congruence subgroup gamma 0 of
##  the arithmetic group SL2Z[1/m] of level p. It is also
##  set a name and some properties for such a group.
##
##  Input: A pair of positive integers (m,p)
##
##  Output: The congruence subgroup of SL2Z(1/m) of level p
##
##
InstallGlobalFunction(CongruenceSubgroup,
function(m,p)
local H,K,G;
    if m=1 then
        return CongruenceSubgroupGamma0(p);
    fi;
    H:=SL2Z(1/m);
    K:=ConjugateSL2ZGroup(H,[[1,0],[0,p]]);
    G:=Intersection(H,K);
    SetName(G,Concatenation("CongruenceSubgroup of ",Name(H)," level ",
                                                          String(p)));
    G!.levels:=[m,p];
    SetIsHAPRationalMatrixGroup(G,true);
    SetIsHAPRationalSpecialLinearGroup(G,true);
    return G;
end);
##
################### end of CongruenceSubgroup ############################
```

5.3.4 SL2ZTree(m,p)

```
############################################################################
#0
#F  SL2ZTree
##
##  This function will compute a G-equivalent CW-space for
##  group G=SL2Z[1/(m*p)] and a tree X associated with
##  amalgamated product SL2Z[1/m]*SL2z[1/m] over
##  congruence subgroup gamma 0 of level p.
##  Input:  A pair of positive integers (m,p)
##
##  Output: A G-equivarient CW-space where G is the free
##          product of two copiesof SL(1/m) amalgamated by
##          its congruence group of level p
##
InstallGlobalFunction(SL2ZTree,
function(m,p)
local t1,t2,
    Elts,H,K,G,Gamma,
    Id,ID,Idcoset,BoundaryList,
    Boundary,Dimension,Action,Stabilizer,Homotopy,StabGrps,
    pos,RemoveLoops,HtpyRec;
```

```
Elts:=[[[1,0],[0,1]]];

## In this case G=SL2Z
if p=0 then
    H:=Group([[0,-1],[1,0]]);
    K:=Group([[0,-1],[1,1]]);
    G:=SL(2,Integers);
    Gamma:=Group([[-1,0],[0,-1]]);
    Append(Elts,Elements(H));
    Append(Elts,Elements(K));
    Append(Elts,Elements(Gamma));
    Elts:=SSortedList(Elts);

else
    if m=1 then
        H:=SL(2,Integers);
        K:=SL2Z(p);
        Gamma:=CongruenceSubgroupGamma0(p);
    else
        H:=SL2Z(1/m);
        K:=ConjugateSL2ZGroup(H,[[1,0],[0,p]]);
        Gamma:=CongruenceSubgroup(m,p);
    fi;
    G:=SL2Z(1/(m*p));
    ID:=Group(One(G));
    Append(Elts,GeneratorsOfGroup(H));
    Append(Elts,GeneratorsOfGroup(K));
    Append(Elts,GeneratorsOfGroup(Gamma));
    Elts:=SSortedList(Elts);
fi;
SetName(Gamma,"Gamma");
Id:=Position(Elts,[[1,0],[0,1]]);

#########################################################################
#1
#F  pos
##  Find the position of an element in a list
##  Input:  A list Elts and a matrix g
##  Output: If g in Elts return the position of g in the list,
##          otherwise, add g to Elts and return the position.
##
pos:=function(Elts,g)
local posit;

    posit:=Position(Elts,g);
    if posit=fail then
        Add(Elts,g);
        return Length(Elts);
    else
        return posit;
    fi;
end;
#########################################################################

BoundaryList:=[];
t1:=pos(Elts,CanonicalRightCountableCosetElement(H,Elts[Id]^-1)^-1);
t2:=pos(Elts,CanonicalRightCountableCosetElement(K,Elts[Id]^-1)^-1);
Append(BoundaryList,[[[1,t1],[-2,t2]]]);
```

```
#############################################################################
#1
#F  Boundary
##
##  This function presents the boundary map d_n: C_n -> C_{n-1}
##
##  Input:  a pair of integers (n,k) where n is the dimension and
##          k is the position of the generator.
##  Output: a list of words [g,f]
##
Boundary:=function(n,k)
local w;
    if not n=1 then
        return [];
    fi;
    w:=BoundaryList[AbsInt(k)];
    if k>0 then
        return w;
    else
        return NegateWord(w);
    fi;
end;

#############################################################################
#1
#F  Dimension
##
##  This function computes the G-rank of ZG-module C_n
##
##  Input:  a positive integer n
##
##  Output: the G-rank of ZG-module C_n
##
Dimension:=function(n)
    if not n in [0,1] then return 0;fi;
    if n=0 then return 2;fi;
    if n=1 then return 1;fi;
end;
#############################################################################

#############################################################################
#1
#F  Action
##
##  Input:  a triple (n,k,l) of integers
##
##  Output: 1 or -1
##
Action:=function(n,k,l);
    return 1;
end;
#############################################################################

StabGrps:=[];
Add(StabGrps,[H,K]);
Add(StabGrps,[Gamma]);

#############################################################################
```

```
#1
#F  Stabilizer
##
##  Input:  a pair of integers (n,k)
##
##  Output: the kith stabiliser subgroup in dimension n
##
Stabilizer:=function(n,k);
    return StabGrps[n+1][k];
end;
#########################################################################

Idcoset:=pos(Elts,
        CanonicalRightCountableCosetElement(Gamma,Elts[Id]^-1)^-1);

#########################################################################
#1
#F  RemoveLoops
##
##  This function will remove loops in a path.
##
##  Input:  a list of integers d
##
##  Output: a list of integers with no loop.
##
RemoveLoops:=function(d)
local i,h,j,l;
    l:=StructuralCopy(d);
    h:=[[1,0],[0,1]];
    i:=1;
    while i<Length(d) do
        h:=h*d[i];
        if h in H or h in K then
            for j in [1..i-1] do
                Remove(l,1);
            od;
            l[1]:=h;
        fi;
        i:=i+1;
    od;
    return l;
end;
#########################################################################

## Create a record for the homotopy
HtpyRec:=[];
HtpyRec[1]:=[];
HtpyRec[2]:=[];

#########################################################################
#1
#F  Homotopy
##
##  This function presents the homotopy map h_n: C_n -> C_{n+1}
##
##  Input:  a positive integer n and a word w.
##
##  Output: a list of words.
##
```

```
Homotopy:=function(n,w)
local d,path,i,h,k,g,pk,r,t;

if not n=0 then
    return [];
fi;
k:=w[1];
g:=w[2];
pk:=AbsInt(k);
if not IsBound(HtpyRec[pk][g]) then
    d:=SL2ZmElementsDecomposition(Elts[g],p);
    r:=[];
    Add(r,d[1]);
    for i in [2..Length(d)] do
        if d[i]*d[i-1] in H then
            r[Length(r)]:=r[Length(r)]*d[i];
        else
            Add(r,d[i]);
        fi;
    od;
    d:=r;
    r:=[];
    Add(r,d[1]);
    for i in [2..Length(d)] do
        if d[i]*d[i-1] in K then
            r[Length(r)]:=r[Length(r)]*d[i];
        else
            Add(r,d[i]);
        fi;
    od;
    d:=StructuralCopy(r);

    ## kill all the loops in the path d

    d:=RemoveLoops(d);
    if (d[1] in K) and (not d[1] in Gamma) then
        r:=[[[1,0],[0,1]]];
        Append(r,d);
        d:=StructuralCopy(r);
    fi;
    h:=[[1,0],[0,1]];
    path:=[];
    if d[Length(d)] in Gamma and Length(d)>1 then
        Remove(d,Length(d));
    fi;

    if pk=1 then
        if d[Length(d)] in H then
            Remove(d,Length(d));fi;
        for i in [1..Length(d)] do
            h:=h*d[i];
            t:=CanonicalRightCountableCosetElement(Gamma,h^-1)^-1;
            Add(path,[(-1)^(i),pos(Elts,t)]);
        od;
    else
        if Elts[g] in Gamma then
            Add(path,[-1,Idcoset]);
        fi;
        if d[Length(d)] in K then
```

```
                Remove(d,Length(d));
            fi;
            for i in [1..Length(d)] do
                h:=h*d[i];
                t:=CanonicalRightCountableCosetElement(Gamma,h^-1)^-1;
                Add(path,[(-1)^(i),pos(Elts,t)]);
            od;
        fi;
        HtpyRec[pk][g]:=path;
    fi;
    if k>0 then
        return HtpyRec[pk][g];
    else
        return NegateWord(HtpyRec[pk][g]);
    fi;
    end;

   ##########################################################################

return Objectify(HapNonFreeResolution,
    rec(
    dimension:=Dimension,
    boundary:=Boundary,
    homotopy:=Homotopy,
    elts:=Elts,
    group:=G,
    stabilizer:=Stabilizer,
    action:=Action,
    properties:=
    [["length",100],
    ["characteristic",0],
    ["type","resolution"]]  ));
end);
################### end of SL2ZTree #####################################
```

5.3.5 TreeOfResolutionsToSL2Zcomplex(D,G)

```
##########################################################################
#0
#F  TreeOfResolutionsToSL2Zcomplex
##  Input: A list of resolutions D and an arithmetic group G=SL2Z[1/m]
##
##  Output: A G-equivarient CW-space
##
##
InstallGlobalFunction(TreeOfResolutionsToSL2Zcomplex,
function(D,G)
local RH,RK,RGamma,
    H,K,Gamma,
    NameH,
    i,j,m,p,
    C,Resolutions,NamesOfGroups;

    RH:=D[1];
    RK:=D[2];
    RGamma:=D[3];
    H:=RH!.group;
```

```
    K:=RK!.group;
    Gamma:=RGamma!.group;
    NameH:=H!.Name;
    if H=SL(2,Integers) then
        m:=1;
        p:=Gamma!.LevelOfCongruenceSubgroup;
    else
        i:=Position(NameH,'/');
        j:=Position(NameH,']');
        m:=Int(NameH{[i+1..j-1]});
        p:=Gamma!.levels[2];
    fi;
    NamesOfGroups:=[Name(H),Name(K),Name(Gamma)];
    Resolutions:=[RH,RK,RGamma];
    C:=SL2ZTree(m,p);
    C!.resolutions:=[Resolutions,NamesOfGroups];
    return C;
end);
################### end of TreeOfResolutionsToSL2Zcomplex ################
```

5.3.6 SL2ZmElementsDecomposition(g,p)

```
#############################################################################
#O
#F  SL2ZmElementsDecomposition
##
##   Decompose a element in SL(2,Z[1/p*n]) into product of
##   elements in SL(2,Z[1/n])
##   and its conjugation by P=[[1,0],[0,p]]
##
##  Input: A matrix g in SL2Z(2,1/pn)
##         a positive integer p
##  Output: A list of pairs (a,b) where a in SL2Z(1/n) and b in SL2Z(1/m)^P
##
InstallGlobalFunction(SL2ZmElementsDecomposition,
function(g,p)
local i,j,q,k,d,
    S,T,Y,Spr,Tpr,A,B,C,
    l,t,r,m,H,h,K,
    PrimeFactorization,TriangleMatrixDecomposition,InverseOfMatricesList;

    # SL2Z generated by {S,T} or {S,Y}
    S:=[[0,1],[-1,0]];
    T:=[[1,0],[1,1]];
    Y:=[[0,-1],[1,1]];
    l:=[[1,0],[0,1]];
    t:=[];
    r:=[];
    H:=Group(S);
    K:=Group(Y);


    ###########################################################################
    #1
    #F  PrimeFactorization
    ##
    ##  find the power of p in the prime factorisation of n
```

```
##
##  Input:  A pair of positive integers (n,p)
##  Output: the power of p in the prime factorisation of n
##
PrimeFactorization:=function(n,p)
local r,k;
r:=-1;
k:=n;
while IsInt(k) do
    r:=r+1;
    k:=k/p;
od;
return r;
end;
#############################################################################

#############################################################################
#1
#F  TriangleMatrixDecomposition
##
##  Decomposition for a upper triangle matrix with p-factor appears
##  in the numerator of (1,1)-entry
##
##  Input:  A pair of positive integers (n,p)
##  Output: the power of p in the prime factorisation of n
##
TriangleMatrixDecomposition:=function(g,p)
local S,T,Spr,r,
    l,q,d,x,id;

    S:=[[0,1],[-1,0]];
    T:=[[1,0],[1,1]];
    Spr:=[[0,p^-1],[-p,0]];
    l:=[[1,0],[0,1]];
    d:=[];
    id:=[[1,0],[0,1]];
    while g[1][2]<>0 do
        if g[1][2]*g[2][2]<0 and IsInt(g[2][2]/g[1][2])=false then
            q:=Int(g[2][2]/g[1][2])-1;
        else
            q:=Int(g[2][2]/g[1][2]);
        fi;
        l:=T^(-q)*l;
        g:=T^(-q)*g;
        if AbsoluteValue(g[1][2])>AbsoluteValue(g[2][2]) and
                                                g[1][2] <> 0 then
            l:=S*l;
            g:=S*g;
        fi;
    od;
    l:=l^-1;
    x:=PrimeFactorization(NumeratorRat(g[1][1]),p);
    r:=[[g[1][1]*p^-x,0],[g[2][1]*p^x,g[2][2]*p^x]];
    while x>0 do
        Append(d,[S^3,Spr]);
        x:=x-1;
    od;
    d[1]:=l*d[1];
    if r=-id then d[1]:=S^2*d[1];
```

```
    else
        if not r=id then Add(d,r);fi;
    fi;
    return d;
end;
#########################################################################

#########################################################################
#1
#F  InverseOfMatricesList
##
##  Return the inverse of A*B*C as C^-1*B^-1*A^-1
##
##  Input:  a list of matrices [A,B,C,]
##  Output: a list of inversed matrices
##
InverseOfMatricesList:=function(list)
local i,n,d;
    n:=Length(list);
    d:=[];
    for i in [1..n] do
        Add(d,list[n-i+1]^-1);
    od;
    return d;
end;
#########################################################################
if p=0 then       # In case of decompose SL(2,Z) into
                  # product of elements in C4 and C6
    if g=[[1,0],[0,1]] then return [g];fi;
    while g[1][2]<>0 do
        if g[1][1]*g[1][2]<0 and IsInt(g[1][1]/g[1][2])=false then
            q:=Int(g[1][1]/g[1][2])-1;
        else
            q:=Int(g[1][1]/g[1][2]);
        fi;
        g:=g*T^(-q);
        q:=-q;
        if q>0 then
            while q>0 do
                Append(t,[S^3,Y^-1]);
                q:=q-1;
            od;
        fi;
        if q<0 then
            while q<0 do
                Append(t,[Y,S^-3]);
                q:=q+1;
            od;
        fi;
        if AbsoluteValue(g[1][1])<AbsoluteValue(g[1][2]) then
            Add(t,S);
            g:=g*S;
        fi;
    od;
    t:=InverseOfMatricesList(t);
    if g[1][1]>0 then
        q:=g[2][1];
        if q>0 then
            while q>0 do
```

```
            Append(r,[S^3,Y^-1]);
              q:=q-1;
               od;
           fi;
           if q<0 then
               while q<0 do
                   Append(r,[Y,S^-3]);
                   q:=q+1;
               od;
           fi;
       else
           Add(r,S^2);
           q:=-g[2][1];
           if q>0 then
               while q>0 do
                   Append(r,[S^3,Y^-1]);
                   q:=q-1;
               od;
           fi;
           if q<0 then
               while q<0 do
                   Append(r,[Y,S^-3]);
                   q:=q+1;
               od;
           fi;
       fi;
       Append(r,t);
       m:=[];
       Add(m,r[1]);
       for i in [2..Length(r)] do
           if r[i]*r[i-1] in H or r[i]*r[i-1] in K then
               m[Length(m)]:=m[Length(m)]*r[i];
           else
               Add(m,r[i]);
           fi;
       od;
       r:=[];
       Add(r,m[1]);
       for i in [2..Length(m)] do
           if m[i]*m[i-1] in H or m[i]*m[i-1] in K then
               r[Length(r)]:=r[Length(r)]*m[i];
           else
               Add(r,m[i]);
           fi;
       od;
       return r;
   else
   # In case of decompose SL(2,Z[1/p*n]) into product of elements
   # in SL(2,Z[1/n]) and its conjugation by P=[[1,0],[0,p]]

       if PrimeFactorization(DenominatorRat(g[1][1]),p)
         +PrimeFactorization(DenominatorRat(g[1][2]),p)
         +PrimeFactorization(DenominatorRat(g[2][1]),p)
         +PrimeFactorization(DenominatorRat(g[2][2]),p)=0 then
           return [g];
       fi;
       Spr:=[[0,p^-1],[-p,0]];
       Tpr:=[[1,0],[p,1]];
       while g[1][1]<>0 do
```

```
            if g[1][1]*g[2][1]<0 and IsInt(g[2][1]/g[1][1])=false then
                q:=Int(g[2][1]/g[1][1])-1;
            else
                q:=Int(g[2][1]/g[1][1]);
            fi;
            l:=T^(-q)*l;
            g:=T^(-q)*g;
            if AbsoluteValue(g[1][1])>AbsoluteValue(g[2][1]) then
                l:=S*l;
                g:=S*g;
            fi;
        od;
        l:=S*l;
        g:=S*g;
        l:=l^-1;
        k:=PrimeFactorization(NumeratorRat(g[1][1]),p);
        if k>0 then
            d:=TriangleMatrixDecomposition(g,p);
            d[1]:=l*d[1];
        else
            d:=TriangleMatrixDecomposition(g^-1,p);
            if d[Length(d)] in SL2Z(p) and not d[Length(d)]
                        in CongruenceSubgroupGamma0(p) then
                Add(d,l^-1);
                d:=InverseOfMatricesList(d);
            else
                d:=InverseOfMatricesList(d);
                d[1]:=l*d[1];
            fi;
        fi;
        r:=[d[1]];
        for i in [2..Length(d)] do
            if d[i]*d[i-1] in H or d[i]*d[i-1] in K then
                r[Length(r)]:=r[Length(r)]*d[i];
            else
                Add(r,d[i]);
            fi;
        od;
        return r;
    fi;
end);

################### end of SL2ZmElementsDecomposition ######################
```

5.3.7 ResolutionGTree(R,n)

```
##############################################################################
#0
#F  ResolutionGTree
##
##  This function use C.T.C Walls method to build a free ZG-resolution
##  from a G-equivariant CW-complex given resolutions for all stabilisers
##
##  Input: A non free ZG-resolution with resolutions for all stabilizers,
##         a positive integer n
##  Output: The first n+1 terms of a free ZG-reoslution of Z.
##
```

```
##

InstallGlobalFunction(ResolutionGTree,
function(arg)
local
R,n,G,i,L,
StabRes,StabGrps,Triple2Pair,Quad2One,GrpsRes,Pair2Quad,Quad2Pair,
Dimension,Hmap,Gmult,Gmultrec,Mult,AlgRed,CorrectList,Boundary,
Homotopy,FinalHomotopy,
StRes,hmap,Action,PseudoBoundary,PseudoHomotopy, ZeroDimensionHmap,
        ZeroDimensionHtpy,
HmapRec,p,q,r,s,HtpyRec,k,g, ZeroDimensionHmapRec, Pair2QuadRec,
        QuadToPairRec,DimensionRec;

    R:=arg[1];
    n:=arg[2];
    G:=R!.group;

    ######################################################################
    #1
    #F  Action
    ##
    ##  Input:  a triple of integers (p,r,q)
    ##  Output: 1 or -1
    ##

    Action:=function(p,r,g)
    if not IsBound(R!.action) then
        return 1;
    else
        return R!.action(p,r,g);
    fi;
    end;

    ######################################################################
    AlgRed:=AlgebraicReduction;

    Gmultrec:=[];

    ######################################################################
    #1
    #F  Gmult
    ##
    ##  The product of 2 elements in Elts
    ##
    ##  Input:  a pair of positive integers (i,j)
    ##  Output: the position of the product in the list Elts
    ##
    Gmult:=function(i,j)
    local posit,g;
    if not IsBound(Gmultrec[i]) then
        Gmultrec[i]:=[];
    fi;
    if not IsBound(Gmultrec[i][j]) then
        g:=R!.elts[i]*R!.elts[j];
        posit:=Position(R!.elts,g);
        if posit=fail then
            Add(R!.elts,g);
            posit:= Length(R!.elts);
```

```
    fi;
    Gmultrec[i][j]:=posit;
fi;

return Gmultrec[i][j];
end;
#########################################################################

#########################################################################
#1
#F  Mult(g,w)
##
##  Multiply gth-element with a list of words w
##
##  Input:  an integer g and a list of words w
##  Output: the product of q and w
##

Mult:=function(g,w) # Multiply gth-element with a word
local
    l,x;

l:=StructuralCopy(w);
if R!.elts[g]=[] then
    return [];
fi;
Apply(l,y->[y[1],Gmult(g,y[2])]);
return l;
end;
#########################################################################

#########################################################################
#1
#F  GrpsRes
##
##  Return a free resolution for a given group
##
##  Input:  A group G and a positive integer n
##  Output: The first n+1 term of a free ZG-resolution of Z
##
GrpsRes:=function(G,n) # Resolutions of Group
local
    iso,Q,res,x;
if IsBound(R!.resolutions) and HasName(G) then
    x:=Position(R!.resolutions[2], Name(G));
    if not x=fail then
        return R!.resolutions[1][x];
    fi;
fi;
iso:=RegularActionHomomorphism(G);
Q:=Image(iso);
res:=ResolutionFiniteGroup(Q,n);
res!.group:=G;
res!.elts:=List(res!.elts,x->PreImagesRepresentative(iso,x));
return res;
end;
#########################################################################

#Create list of stabilizer subgroups and their resolutions
```

```
StabGrps:= List([0..Length(R)],n->
               List([1..R!.dimension(n)], k->R!.stabilizer(n,k)));
StabRes:=[];
for L in StabGrps do
    Add(StabRes,List(L,g->ExtendScalars(GrpsRes(g,n),G,R!.elts))  );
od;
#########################################################################
CorrectList:=function(list)
local
    l,i;
if list=[] then return [];fi;
l:=StructuralCopy(list[1]);
for i in [2..Length(list)] do
    Append(l,StructuralCopy(list[i]));
od;
return l;
end;
#########################################################################


#########################################################################
#1
#F  Quad2One
##
##  return nth-generator of F_(p,q) from (r,s)th-generator of
##  stabilizer
##
##  Input:  A quadruple of integers (p,q,r,s)
##  Output: n-th generator of F_{p,q}
##
Quad2One:=function(p,q,r,s)
local
    n,d,i,j;
n:=0;
i:=SignInt(s);
s:=AbsInt(s);
d:=List([1..R!.dimension(p)],x->StabRes[p+1][x]!.dimension(q));
for j in [1..r-1] do
    n:=n+d[j];
od;
n:=n+s;
if q=0 and n>R!.dimension(p) then
    n:=R!.dimension(p);
fi;
return i*n;
end;
#########################################################################

#########################################################################
#1
#F  Triple2Pair
##
##  Input:  A triple of integers (p,q,n)
##  Output: a pair of integers (r,s)
##
Triple2Pair:=function(p,q,n)
local
r,s,d,i;
r:=0;
```

```
d:=List([1..R!.dimension(p)],x->StabRes[p+1][x]!.dimension(q));
i:=SignInt(n);
n:=AbsInt(n);
while n>0 do
    r:=r+1;
    s:=n;
    n:=n-d[r];
od;
return [r,i*s];
end;
#########################################################################

# Create a record for horizontal map: d_1
HmapRec:=[];
for p in [1..2] do
    HmapRec[p]:=[];
    for q in [1..n+1] do
        HmapRec[p][q]:=[];
        for r in [1..R!.dimension(p-1)] do
            HmapRec[p][q][r]:=[];
        od;
    od;
od;
ZeroDimensionHmapRec:=[];


#########################################################################
#1
#F  ZeroDimensionHmap
##
##  Input:  An integer k
##  Output: The map d_1 at degree 0
##
ZeroDimensionHmap:=function(k)
local i,j,pk;
pk:=AbsInt(k);
if not IsBound(ZeroDimensionHmapRec[pk]) then
    j:=0;
    for i in [1..pk-1] do
        j:=j+StabRes[1][i]!.dimension(0);
    od;
    j:=j+1;
    ZeroDimensionHmapRec[pk]:=j;
fi;
if k>0 then
    return ZeroDimensionHmapRec[pk];
else return -ZeroDimensionHmapRec[pk];fi;
end;
#########################################################################

#########################################################################
#1
#F  ZeroDimensionHmap
##
##  Horiziontal map d_1:A(p,q)->A(p-1,q), acts on the
##  (r,s) th-generator of A(p,q)
##
##  Input:  An integer k
##  Output: The map d_1 at degree 0
```

```
##
Hmap:=function(p,q,r,s)
local
    i,l,d0,m,bdr,ps,d1d0,w;
ps:=AbsInt(s);
if p<>1 then
    return [];
else
    if not IsBound(HmapRec[p+1][q+1][r][ps]) then
        if q=0 then bdr:=StructuralCopy(R!.boundary(1,1));
            Apply(bdr,w->[ZeroDimensionHmap(w[1]),w[2]]);
            HmapRec[p+1][q+1][r][ps]:=bdr;
        else
            l:=[];m:=[];
            d0:=StructuralCopy(List(StabRes[p+1][r]!.boundary(q,ps),
                                x->[Action(p,r,x[2])*x[1],x[2]]));
            for w in d0 do
                Append(m,Mult(w[2],Hmap(p,q-1,r,w[1])));
            od;
            Apply(m,x->[Triple2Pair(p-1,q-1,x[1]),x[2]]);
            for w in m do
                Append(l,List(StabRes[p][w[1][1]]!.homotopy
                                (q-1,[w[1][2],w[2]]),y->
                            [Quad2One(p-1,q,w[1][1],y[1]),y[2]]));
            od;
            HmapRec[p+1][q+1][r][ps]:=AlgRed(l);
        fi;
    fi;
fi;
if SignInt(s)=1 then
    return HmapRec[p+1][q+1][r][ps];
else
    return NegateWord(HmapRec[p+1][q+1][r][ps]);
fi;
end;
#########################################################################

Pair2QuadRec:=[];

#########################################################################
#1
#F  Pair2Quad
##
##  Input:  A pair of integers (k,n)
##  Output: A triple of integers
##
Pair2Quad:=function(k,nn)
local
    x,n,nnn, p,q,r,s,i,temp,j1,j2;
i:=SignInt(nn);
n:=AbsInt(nn);
nnn:=n;

if not IsBound(Pair2QuadRec[k+1]) then
    Pair2QuadRec[k+1]:=[];
fi;

if not IsBound(Pair2QuadRec[k+1][n]) then
```

```
        temp:=0;
        for j1 in [0..k] do
            for j2 in [1..R!.dimension(j1)] do
                temp:=temp+StabRes[j1+1][j2]!.dimension(k-j1);
            od;
        od;

        p:=-1;
        while n>0 do;
            p:=p+1;
            r:=0;
            while (n>0 and r<R!.dimension(p)) do
                r:=r+1;
                s:=n;
                n:=n-StabRes[p+1][r]!.dimension(k-p);
            od;
        od;
        q:=k-p;
        Pair2QuadRec[k+1][nnn]:=[p,q,r,s];

    fi;

    x:=Pair2QuadRec[k+1][nnn];
    return [x[1],x[2],x[3],i*x[4]];

    end;
    ##########################################################################

    QuadToPairRec:=[];

    ##########################################################################
    #1
    #F  Pair2Quad
    ##
    ##  Input:  A pair of integers (k,n)
    ##  Output: A triple of integers
    ##
    Quad2Pair:=function(p,q,r,s)
    local
        k,n,i,j,p1,q1,r1,s1;

    p1:=p+1;q1:=q+1;r1:=r+1;s1:=AbsInt(s)+1;
    if not IsBound(QuadToPairRec[p1]) then
        QuadToPairRec[p1]:=[];
    fi;
    if not IsBound(QuadToPairRec[p1][q1]) then
        QuadToPairRec[p1][q1]:=[];
    fi;
    if not IsBound(QuadToPairRec[p1][q1][r1]) then
        QuadToPairRec[p1][q1][r1]:=[];
    fi;
    if not IsBound(QuadToPairRec[p1][q1][r1][s1]) then
        k:=p+q;
        n:=0;
        for i in [0..p-1] do
            for j in [1..R!.dimension(i)] do
                n:=n+StabRes[i+1][j]!.dimension(k-i);
            od;
        od;
```

```
    for i in [1..r-1] do
        n:=n+StabRes[p+1][i]!.dimension(k-p);
    od;
    n:=n+AbsInt(s);
    QuadToPairRec[p+1][q+1][r+1][AbsInt(s)+1]:=[k,n];

fi;
k:=QuadToPairRec[p1][q1][r1][s1];
return [k[1],SignInt(s)*k[2]];
end;
#########################################################################

# Create an empty list for the pseudo boundary
PseudoBoundary:=[];
for k in [1..n+1] do
    PseudoBoundary[k]:=[];
od;

#########################################################################
#1
#F  Boundary
##
##  Input:  A pair of integers (k,n)
##  Output: the boundary d(k,n)
##
Boundary:=function(k,n)
local
    d,l,p,q,r,s,w,pn;
pn:=AbsInt(n);
if not IsBound(PseudoBoundary[k+1][pn]) then
    w:=Pair2Quad(k,pn);
    p:=w[1];q:=w[2];r:=w[3];s:=w[4];
    d:=[];
    if q<>0 then
l:=StructuralCopy(List(StabRes[p+1][r]!.boundary(q,s),
          x->[Quad2Pair(p,q-1,r,Action(p,r,x[2])*x[1])[2],x[2]]));
        Append(d,StructuralCopy(l));
    fi;
    if IsEvenInt(q) then
        Append(d,StructuralCopy(Hmap(p,q,r,s)));
    else
        Append(d,StructuralCopy(NegateWord(Hmap(p,q,r,s))));
    fi;
    PseudoBoundary[k+1][pn]:=AlgRed(d);
fi;
if SignInt(n)=1 then
    return PseudoBoundary[k+1][pn];
else
    return NegateWord(PseudoBoundary[k+1][pn]);
fi;
end;
#########################################################################

DimensionRec:=[];

#########################################################################
#1
#F  Boundary
##
```

```
##  Input:  A pair of integers (k,n)
##  Output: A triple of integers
##
Dimension:=function(n)
local
    dim,p,i;

if not IsBound(DimensionRec[n+1]) then
    dim:=0;
    for p in [0..n] do
        for i in [1..R!.dimension(p)] do
            dim:=dim+StabRes[p+1][i]!.dimension(n-p);
        od;
    od;
    DimensionRec[n+1]:= dim;
fi;

return DimensionRec[n+1];
end;
#########################################################################

# Create a record for the homotopy map
HtpyRec:=[];
for k in [1..n] do
    HtpyRec[k]:=[];
    for s in [1..Dimension(k-1)] do
        HtpyRec[k][s]:=[];
    od;
od;

#########################################################################
#1
#F  ZeroDimensionHtpy
##
##  Input:  An integer k
##  Output: the homotopy h(0,[1,k]) of the chain complex
##
ZeroDimensionHtpy:=function(k)
local i,j,r;
i:=0;
while k>0 do
    i:=i+1;
    k:=k-StabRes[1][i]!.dimension(0);
    r:=i;
od;
return r;
end;
#########################################################################

#########################################################################
#1
#F  Homotopy
##
##  Input:  An integer n and a word w=[f,g]
##  Output: the homotopy h(n,w)
##
Homotopy:=function(n,w)
local
    t,g,h0,h11,e,h,dh,
```

```
    p,q,r,s,v,m,pt,ppt,
    h1,d1h1,x,k,y,ps;
t:=w[1];
g:=w[2];
e:=[];
h:=[];
dh:=[];
pt:=AbsInt(t);
v:=Pair2Quad(n,pt);#Print(v);
p:=v[1];q:=v[2];r:=v[3];s:=v[4];
if not IsBound(HtpyRec[n+1][pt][g]) then
    if n=0 then
        ppt:=ZeroDimensionHtpy(pt);
        h1:=StructuralCopy(R!.homotopy(n,[ppt,g]));
        d1h1:=StructuralCopy(AlgRed(CorrectList(List(h1,x->
                                Mult(x[2],Hmap(p+1,q,1,x[1]))))));
    for x in d1h1 do
        k:=Pair2Quad(n,x[1]);
        y:=StructuralCopy(StabRes[k[1]+1][k[3]]!.
                                        homotopy(q,[k[4],x[2]]));
        Apply(y,w->[Quad2Pair(k[1],k[2]+1,k[3],w[1])[2],w[2]]);
        Append(e,y);
    od;
    h0:=StructuralCopy(StabRes[p+1][r]!.homotopy(0,[s,g]));
    Apply(h0,w->[Quad2Pair(p,q+1,r,w[1])[2],w[2]]);
    h11:=List(h1,x->[Quad2Pair(p+1,q,Triple2Pair(p+1,q,x[1])[1],
                            Triple2Pair(p+1,q,x[1])[2])[2],x[2]]);
    Append(h,NegateWord(e));
    Append(h,h0);
    Append(h,h11);
    HtpyRec[n+1][pt][g]:=AlgRed(h);
    else
    if p=0 then
    h0:=StructuralCopy(StabRes[p+1][r]!.homotopy(q,[s,g]));
    Apply(h0,w->[Quad2Pair(p,q+1,r,w[1])[2],w[2]]);
    Append(h,h0);
    else
        ps:=Action(1,1,g)*s;
        m:=StructuralCopy(StabRes[p+1][r]!.homotopy(q,[ps,g]));
        Apply(m,x->[Action(1,1,x[2])*x[1],x[2]]);
        h0:=List(m,x->[Quad2Pair(p,q+1,r,x[1])[2],x[2]]);
        Append(h,h0);
        dh:=AlgRed(CorrectList(List(m,x->
                            Mult(x[2],Hmap(p,q+1,1,x[1])))));
        for x in dh do
            k:=Pair2Quad(n,x[1]);
            y:=StructuralCopy(StabRes[k[1]+1][k[3]]!.
                              homotopy(k[2],[k[4],x[2]]));
            Apply(y,w->[Quad2Pair(k[1],k[2]+1,k[3],w[1])[2],w[2]]);
            Append(e,y);
        od;
        if IsEvenInt(q) then
            Append(h,e);
        else
Append(h,NegateWord(e));
            fi;
    fi;
        HtpyRec[n+1][pt][g]:=AlgRed(h);
    fi;
```

```
    fi;
    if SignInt(t)=1 then
        return HtpyRec[n+1][pt][g];
    else
        return NegateWord(HtpyRec[n+1][pt][g]);
    fi;
    end;

    ######################################################################

    ######################################################################
    #1
    #F  FinalHomotopy
    ##
    ##  Input:  An integer n and a word w=[f,g]
    ##  Output: the homotopy h(n,w)
    ##
    FinalHomotopy:=function(n,g)
    if R!.homotopy=fail then
        return fail;
    else
        return Homotopy(n,g);
    fi;
    end;


    ####ADDED MAY 2012#################################################
    StRes:=function(n,k)
        return StabRes[n+1][k];
    end;

######################################################################
return Objectify(HapResolution,
                rec(
                dimension:=Dimension,
                boundary:=Boundary,
                homotopy:=FinalHomotopy,
                elts:=R!.elts,
                group:=R!.group,
                stabres:=StRes,
                properties:=
                   [["length",n],
                    ["initial_inclusion",true],
                    ["type","resolution"],
                    ["characteristic",EvaluateProperty(
                                 R,"characteristic")]  ] ));
end);

##
################### end of ResolutionGTree ############################
```

5.3.8 SL2ZResolution(m,n)

```
######################################################################
#0
#F  SL2ZResolution
##  Input: A pair of positive integers (m,n)
```

```
##
##  Output: The first n+1 terms of a free ZG-resolution
##          where G is SL2Z(1/m)
##

InstallGlobalFunction(SL2ZResolution,
function(m,n)
local l,p,k,
    C,R,T,RH,RK,RGamma,H,K,Gamma,D,G,F,RF;

l:=Factors(m);
    p:=l[Length(l)];
    k:=m/p;
    R:=ResolutionSL2Z(1,n);
    if m=1 then
        return R;
    else
    RH:=SL2ZResolution(k,n);
    H:=RH!.group;

## Create resolution for K
    RK:=ConjugatedResolution(RH,[[1,0],[0,p]]);
    RK!.group:=ConjugateSL2ZGroup(H,[[1,0],[0,p]]);

## Create resolution for Gamma
    Gamma:=CongruenceSubgroup(k,p);
    RGamma:=ResolutionFiniteSubgroup(RH,Gamma);
    SetName(Gamma,"Gamma");

## Create tree of groups
    D:=[RH,RK,RGamma];
    G:=SL2Z(1/m);

## Compute a non-free complex for SL(2,Z[1/p])
    F:=TreeOfResolutionsToSL2Zcomplex(D,G);
    RF:=ResolutionGTree(F,n);

    return RF;
    fi;
end);

################### end of SL2ZResolution ############################
```

5.3.9 IsIntList(list)

```
#############################################################################
#0
#F  IsIntList
##  Input: A list L
##
##  Output: True if L is a list of integers
##          False otherwise
##

InstallGlobalFunction(IsIntList,
function(list)
local i;
```

```
    for i in list do
        if not IsInt(i) then
            return false;
        fi;
    od;
    return true;
end);
#############################################################################
```

5.3.10 VectorToCrystMatrix(v)

```
#############################################################################
#0
#F  VectorToCrystMatrix
##  Input: A n-dimensional vector v
##
##  Output: A pure translation crystallographic matrix
##
##
InstallGlobalFunction(VectorToCrystMatrix,
function(v)
local M,n;

    v:=Flat(v);
    n:=Length(v);
    M:=IdentityMat(n+1);
    Add(v,1);
    Remove(M);
    Add(M,v);
    return M;
end);
################### end of VectorToCrystMatrix ######################
```

5.3.11 CrystTranslationMatrixToVector(g)

```
#############################################################################
#0
#F  CrystTranslationMatrixToVector
##  Input: A pure translation crystallographic matrix g
##
##  Output: An n-dimensional vector v
##
##
InstallGlobalFunction(CrystTranslationMatrixToVector,
function(g)
local n,v;

    n:=Length(g);
    v:=g[n];
    v:=Flat(v);
    Remove(v);
    return v;
end);
################### end of CrystTranslationMatrixToVector ############
```

5.3.12 TranslationSubGroup(G)

```
#########################################################################
#0
#F  TranslationSubGroup
##  Input: A crystallographic group G
##
##  Output: Translation subgroup of G
##
##
InstallGlobalFunction(TranslationSubGroup,
function(G)
local B,SbGrp,trsltbasis;

    B:=TranslationBasis(G);
    if not IsBound(G!.TranslationBasis) then return false;fi;
    B:=G!.TranslationBasis;
    trsltbasis:=List(B,w->VectorToCrystMatrix(Flat(w)));
    SbGrp:=Group(trsltbasis);
    SbGrp!.TranslationBasis:=B;
    SetIsCrystTranslationSubGroup(SbGrp,true);
    return SbGrp;
end);

################### end of TranslationSubGroup ######################
```

5.3.13 Method g in G

```
#########################################################################
#0
#F  Method in
##  Input: A matrix g and a translation subgroup G of
##                                              a crystallographic group
##         a positive integer p
##  Output: True if g in G
##          False if g not in G
##
InstallOtherMethod(\in,
        "for TranslationSubGroup of a CrystGroup",
          [IsMatrix,IsCrystTranslationSubGroup],
function(g,G)
local B,v,n;

    n:=DimensionSquareMat(g)-1;
    if not LinearPartOfAffineMatOnRight(g)=IdentityMat(n) then
         return false;
    fi;
    B:=G!.TranslationBasis;
    v:=CrystTranslationMatrixToVector(g);

    return IsIntList(v*TransposedMat(B)^-1);

end);

################### end of Method g in G ############################
```

5.3.14 IsCrystSameOrbit(arg)

```
#########################################################################
#0
#F  IsCrystSameOrbit
##  Input: Two points u, v in R^n and a group G.
##
##  Output: return True if u, v in the same orbit.
##          Otherwise returns False.
##
##
InstallGlobalFunction(IsCrystSameOrbit,
function(arg)
    local G,T,H,u,v,B,x,w;

    G:=arg[1];
    if Length(arg)=3 then
        H:=TranslationSubGroup(G);
        T:=RightTransversal(G,H);
        B:=H!.TranslationBasis;
        u:=arg[2];
        v:=arg[3];
    else
        B:=arg[2];
        T:=arg[3];
        u:=arg[4];
        v:=arg[5];
    fi;
    u:=Flat(u);
    v:=Flat(v);
    Add(u,1);
    Add(v,1);
    for x in T do
        w:=u*x-v;
        w:=Flat(w);
        Remove(w);
        if IsIntList(w*TransposedMat(B)^-1) then
            return x*VectorToCrystMatrix(w)^-1;
        fi;
    od;
    return false;
end);

################### end of IsCrystSameOrbit ##############################
```

5.3.15 CombinationDisjointSets(arg)

```
#########################################################################
#0
#F  CombinationDisjointSets
##  Input: A list of k positive integers $(a_i)$.
##
##  Output: A 2-dimensional array L such that 0 <= L[i][j] < a_j.
##
##

InstallGlobalFunction(CombinationDisjointSets,
```

```
function(arg)
local b,list,n1,i,g,h;

    g:=arg[1];
    if g=[] then return [[]];fi;
    n1:=g[1];
    h:=g{[2..Length(g)]};
    list:=[];
    b:=CombinationDisjointSets(h);
    for i in [0..(n1-1)] do
        Append(list,List(b,w->AddFirst(w,i)));
    od;
    return list;
end);

################### end of CombinationDisjointSets ######################
```

5.3.16 AddFirst(list,g)

```
#############################################################################
#0
#F  AddFirst
##  Input: A list w and an element g.
##
##  Output: List with g in the first position
##
##
InstallGlobalFunction(AddFirst,
function(list,g)                # add g in the first position in list
local w;
    w:=[g];
    Append(w,list);
    return w;
end);
##
################### end of AddFirst #####################################
```

5.3.17 IsCrystSufficientLattice(B,S)

```
#############################################################################
#0
#F  IsCrystSufficientLattice
##  Input: A lattice's basis B and a transversal S of the translation
##         subgroup in crystallographic group G.
##
##  Output: True if G acts on the lattice, otherwise return False.
##
##
InstallGlobalFunction(IsCrystSufficientLattice,
function(B,S)
local
    v,x,w,B1,c,i,A,Origin,S1,S2;
    Origin:=0*B[1];
    Origin:=Flat(Origin);
    Add(Origin,1);
```

```
    A:=StructuralCopy(B);
    v:=Sum(A)/2;
    v:=Flat(v);
    Add(v,1);
    for x in S do
        w:=Flat(v*x-v);
        if not IsIntList(w*A^-1) then
            return false;
        fi;
    od;
    for i in [1..Length(A)] do
        A[i]:=Flat(A[i]);
        Add(A[i],1);
    od;
    for x in S do
        B1:=List(A,w->w*x);
        c:=Flat(Origin*x-Origin);
        B1:=List(B1,w->w-c);
        for x in B1 do
            Remove(x);
        od;
        S2:=Set(B1);
        for x in B do
            if not ((x in S2) or (-x in S2)) then
            return false;
            fi;
        od;
    od;
    return true;
end);
#################### end of IsCrystSufficientLattice ######################
```

5.3.18 CrystFinitePartOfMatrix(g)

```
##############################################################################
#0
#F  CrystFinitePartOfMatrix
##  Input: A crystallographic matrix g
##
##  Output: Finite part of g.
##
##
InstallGlobalFunction(CrystFinitePartOfMatrix,
function(g)
local
    x,w,i;
    w:=[];
    for i in [1..(Length(g)-1)] do
        x:=Flat(g[i]);
        Remove(x);
        Add(w,x);
    od;
    return w;
end);
#################### end of CrystFinitePartOfMatrix ######################
```

5.3.19 ResolutionBoundaryOfWordOnRight(R,n,W)

```
#############################################################################
#O
#F  ResolutionBoundaryOfWordOnRight
##  Input: A free resolution R, degree n, a list of word w
##
##  Output: The boundary of w respects to the right action.
##
##
InstallGlobalFunction(ResolutionBoundaryOfWordOnRight,
function(R,n,W)
local
    x, DW, Boundary, Dimension,Elts,pos, ans,H;

    Dimension:=R!.dimension;
    Boundary:=R!.boundary;
    Elts:=R!.elts;
    DW:=[];

    for x in W do
        ans:=Boundary(n,x[1]);
        ans:=List(ans, a->[a[1],Elts[a[2]]]);
        ans:=List(ans, a->[a[1],a[2]*Elts[x[2]]]);
        Append(DW,ans);
    od;

    DW:= AlgebraicReduction(DW);
    for x in DW do
        if not x[2] in Elts then
            Add(Elts,x[2]);
        fi;
    od;
    DW:=List(DW,x->[x[1],Position(Elts,x[2])]);
    DW:=List(DW,x->[R!.Sign(n-1,x[1],x[2])*x[1],x[2]]);
    for x in DW do
        H:=R!.stabilizer(n-1,AbsInt(x[1]));
        pos:=Position(R!.elts,CanonicalRightCountableCosetElement(
                                                H,R!.elts[x[2]]));
        if pos=fail then
            Add(R!.elts,CanonicalRightCountableCosetElement(
                                                H,R!.elts[x[2]]));
            x[2]:=Length(R!.elts);
        else
            x[2]:=pos;
        fi;
    od;
    DW:=List(DW,x->[R!.Sign(n-1,x[1],x[2])*x[1],x[2]]);
    DW:= AlgebraicReduction(DW);
    return DW;
end);
################### end of ResolutionBoundaryOfWordOnRight ###############
```

5.3.20 CrystCubicalTiling(n)

```
#############################################################################
#O
```

```
#F  CrystCubicalTiling
##  Input: Dimension n
##
##  Output: A list of some cubical tiling in n-dimensional space
##
InstallGlobalFunction(CrystCubicalTiling,
function(n)
local
    combin,x,w,Til,i;
    combin:=Combinations([1..n],2);
    Til:=[];
    Add(Til,IdentityMat(n));
    for i in [1..Length(combin)] do
        w:=combin[i];
        x:=IdentityMat(n);
        x[w[1]][w[1]]:=-1/2;
        x[w[1]][w[2]]:=-1/2;
        x[w[2]][w[1]]:=1;
        x[w[2]][w[2]]:=-1;
        Add(Til,x);
    od;
    return Til;
end);
#################### end of CrystCubicalTiling ############################
```

5.3.21 AverageInnerProduct(G,u,v)

```
##############################################################################
#0
#F  AverageInnerProduct
##  Input: An affine group G and 2 vector u,v
##
##  Output: the avarage inner product of u and v.
##
InstallGlobalFunction(AverageInnerProduct,
function(G,u,v)
local
    i,sum,n,Elts;
    n:=Order(G);
    Elts:=Elements(G);
    sum:=0;
    for i in [1..n] do
        sum:=sum+(u*Elts[i])*(v*Elts[i]);
    od;
    sum:=sum/n;
    return sum;
end);

#################### end of AverageInnerProduct ##########################
```

5.3.22 OrthogonalizeBasisByAverageInnerProduct(B,G)

```
##############################################################################
#0
#F  OrthogonalizeBasisByAverageInnerProduct
```

```
##  Input: A basis B and group G
##
##  Output: An orthogonal basis B' and a a matrix change of basis
##
##
InstallGlobalFunction(OrthogonalizeBasisByAverageInnerProduct,
function(B,G)
local
    Project,i,j,A,n;
    A:=StructuralCopy(B);
    n:=Length(B);
    if not RankMat(B)=Length(B) then
        Print("Input is not a basis");
        return fail;
    fi;

    Project:=function(u,v)
                        #This operator projects the vector v orthogonally
                          #onto the line spanned by vector u
        return (AverageInnerProduct(G,u,v)/AverageInnerProduct(G,u,u))*u;
    end;

    for i in [2..n] do
        for j in [1..(i-1)] do
            B[i]:=B[i]-Project(B[j],A[i]);
        od;
    od;
    for i in [1..n] do
        B[i]:=(1/Sqrt(AverageInnerProduct(G,B[i],B[i])))*B[i];
    od;
    return B;
end);

########## end of OrthogonalizeBasisByAverageInnerProduct ################
```

5.3.23 CrystMatrix(M)

```
#############################################################################
#O
#F  CrystMatrix
##  Input: nxn matrix M
##
##  Output: the crystallographic form of M.
##
##
InstallGlobalFunction(CrystMatrix,
function(M)
local
    i,n,x,N;
    N:=StructuralCopy(M);
    if IsMatrix(N) then
        n:=Length(N);
        x:=0*N[1];
        Add(x,1);
        for i in [1..n] do
            Add(N[i],0);
        od;
```

```
        Add(N,x);
    else
        N:=VectorToCrystMatrix(N);
    fi;
return N;
end);

################### end of CrystMatrix ###################################
```

5.3.24 FactorizationNParts(d,n)

```
#############################################################################
#0
#F  FactorizationNParts
##
##  Decompose a positive integer d into product of n integers
##
##  Input: An pair of positive integers (d,n)
##
##  Output: A list of all possible ways to decompose d into
##          product of n integers
##
InstallGlobalFunction(FactorizationNParts,
function(d,n)
local
    t,faclst,x,w,y,j;

    if n=1 then
        return [[d]];
    fi;

    t:=DivisorsInt(d);
    faclst:=[];
    for x in t do
        y:=FactorizationNParts(d/x,n-1);
        for j in y do
            Add(faclst,AddFirst(j,x));
        od;
    od;
    return faclst;
end);
################### end of FactorizationNParts ############################
```

5.3.25 CrystGFullBasis(arg)

```
#############################################################################
#0
#F  CrystGFullBasis
##
##  Search for a G-fullbasis of a G-lattice
##  for the given crystallographic G generated by
##  the set of generators.
##
##  Input: a crystallographic group G
##
```

```
##  Output: If $G$ admits a parallelepiped fundamental region
##          it outputs a $G$-full basis for a $G$-lattice $L$.
##          If $G$ goes through the check for not admitting a
##          parallelepiped fundamental region it outputs "false".
##          Otherwise it outputs "fail".
##
##
InstallGlobalFunction(CrystGFullBasis,
function(arg)
local
    G,P,Bt,d,n,vect,
    i,j,a,faclst,S,gens,L,c,
    B_delta,ctr,v,
    x,SbGrp,T,coef,SubgroupsOfAutCube;


# Create a list of subgroup of the automorphism group of an n-cube
    SubgroupsOfAutCube:=[

    [ "1", "C2", "C2 x C2", "C4", "D8" ],

    [ "1", "C2", "C3", "C2 x C2", "C4", "C6", "S3", "C2 x C2 x C2",
    "D8", "C4 x C2", "A4", "D12", "C2 x D8", "C2 x A4", "S4", "C2 x S4" ],

    [ "1", "C2", "C3", "C2 x C2", "C4", "S3", "C6", "C2 x C2 x C2", "D8",
    "C4 x C2", "C8", "Q8", "D12", "A4", "C6 x C2", "(C4 x C2) : C2",
    "C2 x D8", "C2 x C2 x C2 x C2", "C4 x C4", "C8 : C2", "C4 : C4",
    "D16", "QD16", "C4 x C2 x C2", "C2 x A4", "S4", "SL(2,3)",
    "C2 x C2 x S3", "C2 x C2 x D8", "(C4 x C2 x C2) : C2", "C4 x D8",
    "((C4 x C2) : C2) : C2", "(C2 x D8) : C2", "(C2 x C2 x C2 x C2) : C2",
    "(C8 : C2) : C2", "(C4 x C4) : C2", "C2 x S4", "C2 x C2 x A4",
    "GL(2,3)", "((C4 x C4) : C2) : C2", "(((C4 x C2) : C2) : C2) : C2",
    "D8 x D8", "((C8 : C2) : C2) : C2", "C2 x C2 x S4",
    "((C2 x D8) : C2) : C3",
    "(D8 x D8) : C2", "(((C2 x D8) : C2) : C3) : C2",
    "((((C2 x D8) : C2) : C3) : C2) : C2" ]

    ];
    ####################end of Subgroup of Aut(cube)##########################

    gens:=GeneratorsOfGroup(arg[1]);
    G:=AffineCrystGroup(gens);
    SbGrp:=TranslationSubGroup(G);
    Bt:=G!.TranslationBasis;
    n:=G!.DimensionOfMatrixGroup-1;
    P:=PointGroup(G);

    if not StructureDescription(P) in SubgroupsOfAutCube[n-1] then
        return false;
    fi;

    if not GramianOfAverageScalarProductFromFiniteMatrixGroup(P)=
                                                IdentityMat(n) then
        return "Gramian matrix is not identity matrix";
    fi;

    if Length(arg)=1 then
        L:=CrystCubicalTiling(n);
        Add(L,Bt,1);
```

```
        for i in [1..Length(L)] do
            B_delta:=CrystGFullBasis(G,[L[i],Sum(L[i])/2]);
            if IsList(B_delta) then
                return B_delta;
            fi;
        od;
        return fail;
    else #begin of length 2 input
        L:=arg[2][1];
        c:=arg[2][2];
        vect:=Sum(L)/2-c;
        S:=RightTransversal(G,SbGrp);
        d:=DivisorsInt(Order(P));
        i:=Length(d);

        while i>0 do
            faclst:=FactorizationNParts(d[i],n);
            for x in faclst do
                B_delta:=List([1..n],i->x[i]^-1*L[i]);
                if IsCrystSufficientLattice(B_delta,S) then

 #test if one vertex of fundamental domain is origin
                    ctr:=Sum(B_delta)/2-vect;
                    coef:=CombinationDisjointSets(x);
                    j:=1;
                    while j <= d[i] do
                        v:=ctr+coef[j]*B_delta;
                        if IsCrystSameOrbit(G,Bt,S,ctr,v)=false then
                            break;
                        fi;
                        j:=j+1;
                    od;
                    if j=d[i]+1 then
                        return [B_delta,ctr];
                    fi;

#test if center of fundamental domain is origin
                    ctr:=0*vect;
                    j:=1;
                    while j <= d[i] do
                        v:=ctr+coef[j]*B_delta;
                        if IsCrystSameOrbit(G,Bt,S,ctr,v)=false then
                            break;
                        fi;
                        j:=j+1;
                    od;
                    if j=d[i]+1 then
                        return [B_delta,ctr];
                    fi;
                fi;
            od;
            i:=i-1;

        od;
        return fail;
    fi;  #end of length 2 input

end);
################### end of CrystGFullBasis ################################
```

5.3.26 CrystGcomplex(gens,basis,check)

```
#############################################################################
#0
#F  CrystGcomplex
##  Input: A set F of crystallographic matrices, a G-full basis B and
##         check=1  (check is for future use of implementation of Bredon
##                                                                homology)
##
##  Output: G-equivalent CW-space for group G generated by F.
##
##
InstallGlobalFunction(CrystGcomplex,
function(gens,basis,check)
local i,x,k,combin,n,j,r,m,vect,c,
    B,G,T,S,Bt,Action,Sign,FinalBoundary,BoundaryList,
    L,kcells,cells,w,StabGrp,ActionRecord,lnth,PseudoRotSubGroup,
    RotSubGroupList,
    Dimension,SearchOrbit,pos,StabilizerOfPoint,PseudoBoundary,
    RotSubGroup,
    Elts,Boundary,Stabilizer,DVF,DVFRec,Homotopy,rmult,FinalHomotopy;

    B:=basis[1];
    c:=basis[2];
    vect:=c-Sum(B)/2;

    vect:=0*vect;

    G:=AffineCrystGroup(gens);
    T:=TranslationSubGroup(G);
    Bt:=T!.TranslationBasis;
    S:=RightTransversal(G,T);
    n:=DimensionOfMatrixGroup(G)-1;
    Elts:=[One(G)];
    Append(Elts,gens);
    lnth:=1000;

    if check=1 then    # B is the G-full basis

        L:=[];
        for k in [0..n] do
            L[k+1]:=[];

            ###   list all centers of k-cells

            kcells:=[];
            combin:=Combinations([1..n],k);
            for x in combin do
                w:=[];
                for i in [1..n] do
                    if i in x then
                    Add(w,[1/2]);
                    else Add(w,[0,1]);
                    fi;
                od;
                cells:=Cartesian(w);
                Append(kcells,cells*B+vect);
            od;
```

```
            ###  search for k-orbits
            Add(L[k+1],kcells[1]);
            for i in [2..Length(kcells)] do
                r:=0;
                for j in [1..Length(L[k+1])] do
                    if IsList(IsCrystSameOrbit(G,Bt,S,
                                        kcells[i],L[k+1][j])) then
                    break;
                    fi;
                    r:=r+1;
                od;
                if r=Length(L[k+1]) then Add(L[k+1],kcells[i]);fi;
            od;
        od;

# slice the fundamental cell into 2^n parts to get a
# proper action of G on R^n
    elif check=0 then
        B:=List(B,x->x/2);
        L:=[];
        for k in [0..n] do
            L[k+1]:=[];

            ###   list all centers of k-cells

            kcells:=[];
            combin:=Combinations([1..n],k);
            for x in combin do
            w:=[];
                for i in [1..n] do
                    if i in x then
                    Add(w,[1/2,3/2]);
                    else Add(w,[0,1,2]);
                    fi;
                od;
            cells:=Cartesian(w);
            Append(kcells,cells*B+vect);
            od;

            ###  search for k-orbits
            Add(L[k+1],kcells[1]);
            for i in [2..Length(kcells)] do
                r:=0;
                for j in [1..Length(L[k+1])] do
                    if IsList(IsCrystSameOrbit(G,Bt,S,
                                        kcells[i],L[k+1][j])) then
                        break;
                    fi;
                        r:=r+1;
                od;
                if r=Length(L[k+1]) then Add(L[k+1],kcells[i]);fi;
            od;
        od;
    else
        Print("check is either 1 for B is G-full basis and
                                    0 for proper action", "\n");
        return fail;
    fi;
```

```
    ######################################################################
    #1
    #F  Dimension
    ##
    ##  Input:  An integer k
    ##  Output: ZG-rank of C_k(X)
    ##
    Dimension:=function(k)
        if k>n then
            return 0;
        fi;
        return Length(L[k+1]);
    end;
    ######################################################################

    ######################################################################
    #1
    #F  pos
    ##
    ##  Input:  A matrix g
    ##  Output: If g in Elts then return the position of g, otherwise
    ##          add g to Elts and return the position.
    ##
    pos:=function(g)
    local p;
        p:=Position(Elts,g);
        if p=fail then
            Add(Elts,g);
            return Length(Elts);
        else
            return p;
        fi;
    end;
    ######################################################################

    ######################################################################
    #1
    #F  SearchOrbit
    ##
    ##  Input:  A matrix g
    ##  Output: If g in Elts then return the position of g, otherwise
    ##          add g to Elts and return the position.
    ##
    SearchOrbit:=function(g,k)
    local i,p,h;
        for i in [1..Length(L[k+1])] do
            p:=IsCrystSameOrbit(G,Bt,S,L[k+1][i],g);
            if IsList(p) then
                h:=pos(p);
                return [i,h];
            fi;
        od;
    end;
    ######################################################################

# Create a record for the Action
    ActionRecord:=[];
    for m in [1..lnth+1] do
```

```
    ActionRecord[m]:=[];
    for k in [1..Dimension(m-1)] do
        ActionRecord[m][k]:=[];
    od;
od;


#########################################################################
#1
#F  rmult
##
##  Input:  A list L, degree k, position g of an element
##  Output: Product of g and L by the action on right.
##
rmult:=function(L,k,g)
local x,w,t,h,y,vv;
    vv:=[];
    for x in [1..Length(L)] do
        w:=Elts[L[x][2]]*Elts[g];
        L[x][1]:=Sign(k,L[x][1],pos(w))*L[x][1];
        w:=CanonicalRightCosetElement(StabGrp[k+1]
                                          [AbsInt(L[x][1])],w);
        t:=pos(w);
        Add(vv,[Sign(k,L[x][1],t)*L[x][1],t]);
    od;
    return vv;
end;
#########################################################################

#########################################################################
#1
#F  Action
##
##  Input:  Degree m, position k of a generator and position g of
##          an element.
##  Output: 1 or -1.
##
Action:=function(m,k,g)
local id,r,u,H,abk,ans,x,h,l,i;

    abk:=AbsInt(k);

    if not IsBound(ActionRecord[m+1][abk][g]) then
        H:=StabGrp[m+1][abk];

        if Order(H)=infinity then

        # We are assuming that any infinite stabilizer
        # group acts trivially.

            ActionRecord[m+1][abk][g]:=1;
        else
            id:=CanonicalRightCosetElement(H,Identity(H));
            r:=CanonicalRightCosetElement(H,Elts[g]^-1);
            r:=id^-1*r;
            u:=r*Elts[g];

            if u in RotSubGroupList[m+1][abk] then
                ans:= 1;
```

```
            else
                ans:= -1;
            fi;

            ActionRecord[m+1][abk][g]:=ans;
        fi;
    fi;
    return ActionRecord[m+1][abk][g];
end;
#########################################################################

#########################################################################
#1
#F  Action
##
##  Input:  Degree m, position k of a generator and position g of
##          an element.
##  Output: 1 or -1.
##
PseudoBoundary:=function(k,s)
local f,x,bdry,i,Fnt,Bck,j,ss;
    ss:=AbsInt(s);
    f:=L[k+1][ss];
    if k=0 then return [];fi;
    #x:=f*B^-1;
    x:=(f-vect)*B^-1;
    bdry:=[];
    j:=0;
    for i in [1..n] do
        Fnt:=StructuralCopy(x);
        Bck:=StructuralCopy(x);
        if not IsInt(x[i]) then
            j:=j+1;
            Fnt[i]:=Fnt[i]-1/2;
            Bck[i]:=Bck[i]+1/2;
            #Fnt:=Fnt*B;
            #Bck:=Bck*B;

            Fnt:=Fnt*B+vect;
            Bck:=Bck*B+vect;
            Append(bdry,[SearchOrbit(Fnt,k-1),SearchOrbit(Bck,k-1)]);
            #Append(bdry,[SearchOrbit(Fnt,k-1),SearchOrbit(Bck,k-1)]);
        fi;
    od;
    return bdry;
end;
#########################################################################

#########################################################################
#1
#F  Sign
##
##  Input:  Degree m, position k of a generator and position g of
##          an element.
##  Output: 1 or -1.
##
Sign:=function(m,k,g)
local x,h,p,r,c,i,y,f,s,kk,e,B1,B2,w;
```

```
    kk:=AbsInt(k);
    if m=0 then return 1;fi;
    h:=Elts[g];
    p:=CrystFinitePartOfMatrix(h);
    e:=L[m+1][kk];
    #x:=e*B^-1;
    x:=e*B^-1;
    r:=[];
    for i in [1..Length(x)] do
        if not IsInt(x[i]) then
            Add(r,i);
        fi;
    od;
    B1:=B{r};
    B1:=B1*p;
    e:=Flat(e);
    Add(e,1);
    f:=e*h;
    Remove(f);
    y:=f*B^-1;
    c:=[];
    for i in [1..Length(y)] do
        if not IsInt(y[i]) then
            Add(c,i);
        fi;
    od;

    B2:=B{c};
    s:=[];
    for i in [1..Length(B2)] do
        Add(s,SolutionMat(B1,B2[i]));
    od;
    #Print(s);
    return SignRat(Determinant(s));
end;
#############################################################################

#############################################################################
#1
#F  Boundary
##
##  Input:  degree k and position s of a generator.
##
##  Output: the boundary d(k,s).
##
Boundary:=function(k,s)
local psbdry,j,w,bdry;

    psbdry:=PseudoBoundary(k,s);
    bdry:=[];
    for j in [1..Length(psbdry)] do
        w:=psbdry[j];
        if (j mod 4 = 3) or (j mod 4 = 2) then
            #if IsEvenInt(j) then
            Add(bdry,Negate([Sign(k-1,w[1],w[2])*w[1],w[2]]));
        else
            Add(bdry,[Sign(k-1,w[1],w[2])*w[1],w[2]]);
        fi;
    od;
```

```
    if s<0 then
        return NegateWord(bdry);
    else
        return bdry;
    fi;
end;
#########################################################################

# Create a list of boundary
BoundaryList:=[];
for i in [1..n] do
    BoundaryList[i]:=[];
    for j in [1..Dimension(i)] do
        BoundaryList[i][j]:=Boundary(i,j);
    od;
od;
#########################################################################

#########################################################################
#1
#F  FinalBoundary
##
##  Input:  degree n and position k of a generator.
##
##  Output: the boundary d(k,s).
##
FinalBoundary:=function(n,k)
if k>0 then
    return BoundaryList[n][k];
else
    return NegateWord(BoundaryList[n][AbsInt(k)]);
fi;
end;
#########################################################################

#########################################################################
#1
#F  StabilizerOfPoint
##
##  Input:  a point g in R^n.
##
##  Output: The stabilizer subgroup of g.
##
StabilizerOfPoint:=function(g)
local H,stbgens,i,h,p;
    g:=Flat(g);
    Add(g,1);
    stbgens:=[];
    for i in [1..Length(S)] do
        h:=g*S[i]-g;
        Remove(h);
        p:=h*Bt^-1;
        if IsIntList(p) then Add(stbgens,S[i]*
                                    VectorToCrystMatrix(h)^-1);fi;
    od;
    H:=Group(stbgens);
    return H;
```

```
end;
#########################################################################

#########################################################################
# Create a empty list for containing the stabilizer subgroup
StabGrp:=[];
for i in [1..(n+1)] do
    StabGrp[i]:=[];
    for j in [1..Length(L[i])] do
        StabGrp[i][j]:=StabilizerOfPoint(L[i][j]);
    od;
od;
#########################################################################

#########################################################################
#1
#F  Stabilizer
##
##  Input:  degree m and position k of a generator (the k-th m-cell).
##
##  Output: The stabilizer subgroup for the above cell.
##
Stabilizer:=function(m,k)
    local kk;
    kk:=AbsInt(k);
    return StabGrp[m+1][k];
end;
#########################################################################

#########################################################################
#1
#F  PseudoRotSubGroup
##
##  Input:  degree m and position k of a generator (the k-th m-cell).
##
##  Output: The rotation subgroup of the above cell.
##
PseudoRotSubGroup:=function(m,k)
local x,kk,l,h,i,w,r,y,H,id,eltsH,g,RotSbGrp;
    kk:=AbsInt(k);
    RotSbGrp:=[];
    H:=StabGrp[m+1][k];
    eltsH:=Elements(H);

    for g in eltsH do
        if Sign(m,k,pos(g))=1 then
            Add(RotSbGrp,g);
        fi;
    od;
    RotSubGroupList[m+1][kk]:=Group(RotSbGrp);
    return Group(RotSbGrp);
end;
#########################################################################

#########################################################################
# Create an empty list for containing the rotation subgroups
RotSubGroupList:=[];
for i in [1..(n+1)] do
    RotSubGroupList[i]:=[];
```

```
    for j in [1..Length(L[i])] do
        RotSubGroupList[i][j]:=PseudoRotSubGroup(i-1,j);
    od;
od;
#########################################################################

#########################################################################
#1
#F  RotSubGroup
##
##  Input:  degree m and position k of a generator (the k-th m-cell).
##
##  Output: The rotation subgroup of the above cell.
##
RotSubGroup:=function(m,k)
local kk;
    kk:=AbsInt(k);
    return RotSubGroupList[m+1][kk];
end;
#########################################################################

#########################################################################
# Create a record for discrete vector field
DVFRec:=[];
for k in [1..n+1] do
    DVFRec[k]:=[];
    for i in [1..Length(L[k])] do
        DVFRec[k][i]:=[];
    od;
od;
#########################################################################

if check=1 then

    ###################################################################
    #1
    #F  DVF
    ##
    ##  input an n-cell acts like the starting point of an arrow
    ##  the function returns n+1-cell acts like the end
    ##  point of the above arrow
    ##  those cells presented by its center.
    ##
    ##  Input:  an n-cell.
    ##
    ##  Output: n+1-cell.
    ##
    DVF:=function(k,w)
    local
        f,x,g,i,y,ww,s,b,j;
        ww:=[AbsInt(w[1]),w[2]];
        if not IsBound(DVFRec[k+1][ww[1]][ww[2]]) then
            x:=StructuralCopy(L[k+1][ww[1]]);
            Add(x,1);
            x:=x*Elts[ww[2]];
            Remove(x);
            f:=(x-vect)*B^-1;
            for i in [1..n] do
                if not f[i]=0 then
```

```
                if not IsInt(f[i]) then
                    DVFRec[k+1][ww[1]][ww[2]]:=[];
                    return DVFRec[k+1][ww[1]][ww[2]];
                else
                    s:=SignRat(f[i]);
                    f[i]:=f[i]-s*1/2;
                    x:=f*B;
                    y:=SearchOrbit(x,k+1);
                    y[2]:=pos(CanonicalRightCosetElement
                              (StabGrp[k+2][y[1]],Elts[y[2]]));

                    DVFRec[k+1][ww[1]][ww[2]]:=y;
                    return DVFRec[k+1][ww[1]][ww[2]];
                fi;
            fi;
        od;
        DVFRec[k+1][ww[1]][ww[2]]:=[];
        return DVFRec[k+1][ww[1]][ww[2]];
    else
        return DVFRec[k+1][ww[1]][ww[2]];
    fi;
end;
#########################################################################

#########################################################################
#1
#F  Homotopy
##
##  Input:  Degree k and a word w.
##
##  Output: The homotopy h(k,w).
##
Homotopy:=function(k,w)
local
    h,d,x,y,i,ww,b,p1,p2,s1,s2,v,s,p,t,a,u;

    if w=[] then return [];fi;
    a:=Sign(AbsInt(k),w[1],w[2]);
    d:=[];
    w[2]:=pos(CanonicalRightCosetElement(StabGrp[k+1][AbsInt(w[1])],
                                                    Elts[w[2]]));
    w[1]:=a*Sign(k,w[1],w[2])*w[1];
    ww:=[AbsInt(w[1]),w[2]];
    h:=StructuralCopy(DVF(k,ww));
    if h=[] then
        return [];
    fi;

    x:=PseudoBoundary(k+1,h[1]);
    u:=List(x,v->[v[1],Elts[v[2]]*Elts[h[2]]]);
    u:=List(u,v->[v[1],pos(CanonicalRightCosetElement
                        (StabGrp[k+1][AbsInt(v[1])],v[2]))]);
    p:=Position(u,ww);
    s:=1;;
    b:=StructuralCopy(FinalBoundary(k+1,h[1]));
    b:=rmult(b,k,h[2]);
    c:=StructuralCopy(b);
    t:=SignInt(b[p][1]);
    Remove(c,p);
```

```
            Add(d,h);
            for i in [1..Length(c)] do
                Append(d,NegateWord(Homotopy(k,c[i])));
            od;

            if w[1]*t<0 then
                return NegateWord(d);
            else
                return d;
            fi;
        end;
        ######################################################################

    else
        DVF:=fail;
        FinalHomotopy:=fail;
    fi;


    ######################################################################
    return Objectify(HapNonFreeResolution,
            rec(
            dimension:=Dimension,
            boundary:=FinalBoundary,
        PseudoBoundary:=PseudoBoundary,
        dvf:=DVF,
        CellList:=L,
        Sign:=Sign,
            homotopy:=Homotopy,
            elts:=Elts,
            group:=G,
            stabilizer:=Stabilizer,
            action:=Action,
        RotSubGroup:=RotSubGroup,
            properties:=
            [["length",100],
             ["characteristic",0],
             ["type","resolution"]]  ));

end);
################### end of CrystGcomplex  ############################
```

5.3.27 ResolutionCubicalCrystGroup(G,n)

```
#############################################################################
#0
#F  ResolutionCubicalCrystGroup
##  Input:  A crystallographic group G and an positive integer n
##
##  Output: The first n+1 terms of a free ZG-resolution of Z.
##
##
InstallGlobalFunction(ResolutionCubicalCrystGroup,
function(G,n)
local gens,B,C,R,Gram, pos, Homotopy,Cnew;
    Gram:=GramianOfAverageScalarProductFromFiniteMatrixGroup(
                                                    PointGroup(G));
```

```
    if Gram=IdentityMat(DimensionOfMatrixGroup(PointGroup(G))) then
        gens:=GeneratorsOfGroup(G);
        G:=AffineCrystGroup(gens);
        B:=CrystGFullBasis(G);
        if IsList(B) then
            C:=CrystGcomplex(gens,B,1);
            Cnew:=CrystGcomplex(gens,B,1);
            Apply(Cnew!.elts,x->x^-1);

            pos:=function(L,g)
            local p;
                p:=Position(L,g);
                if p=fail then
                    Add(L,g);
                    return Length(L);
                else
                    return p;
                fi;
            end;

            Homotopy:=function(n,w)
            local p,h;
                p:=pos(C!.elts,Cnew!.elts[w[2]]^-1);
                h:=StructuralCopy(C!.homotopy(n,[w[1],p]));
                Apply(h,x->[x[1],pos(Cnew!.elts,C!.elts[x[2]]^-1)]);
                return h;
            end;

            Cnew!.homotopy:=Homotopy;
            R:=FreeZGResolution(Cnew,n);
            return R;
        else
            return fail;
        fi;
    else
        Print("Gramian matrix is not identity \n");
        return fail;
    fi;
end);
################### end of ResolutionCubicalCrystGroup ###################
```

5.3.28 BredonChainComplex(C)

```
#############################################################################
#O
#F  BredonChainComplex
##  Input:
##
##  Output:
##
##
InstallGlobalFunction(BredonChainComplex,
function(C)
local StabIrrTable,i,j,N,
      Dimension,PairToTriple,BoundaryMatrix,Boundary,
      TripleToPair,StabGrp,BoundaryRec;
```

```
StabGrp:=[];
i:=0;
while C!.dimension(i)>0 do
    StabGrp[i+1]:=[];
    for j in [1..C!.dimension(i)] do
    Add(StabGrp[i+1],C!.stabilizer(i,j));
    od;
    i:=i+1;
od;

StabIrrTable:=[];
i:=0;
while C!.dimension(i)>0 do
    StabIrrTable[i+1]:=[];
    for j in [1..C!.dimension(i)] do
    Add(StabIrrTable[i+1],OrdinaryCharacterTable(StabGrp[i+1][j]));
    od;
    i:=i+1;
od;
N:=i-1;

Dimension:=function(k)
local d,i;
    d:=0;
    for i in [1..C!.dimension(k)] do
        d:=d+Size(StabIrrTable[d+1][i]);
    od;
    return d;
end;

PairToTriple:=function(i,j)
local k,x;
    k:=j;
    x:=1;
    while k>Size(StabIrrTable[i+1][1]) do
        k:=k-Size(StabIrrTable[i+1][x]);
        x:=x+1;
    od;
    return [i,x,k];
    end;

TripleToPair:=function(i,j,k)
local d,x;
    d:=0;
    for x in [1..(j-1)] do
        d:=d+Size(StabIrrTable[i+1][x]);
    od;
    d:=d+k;
    return [i,k];
end;

BoundaryMatrix:=function(n,k)
local bdry,x,Coeffs,Mat,W,A,B,i,xx;
    bdry:=C!.boundary(n,k);
    Mat:=[];
    for i in [1..Length(bdry)] do
        x:=bdry[i][1];
        xx:=AbsInt(x);
        B:=StabGrp[n][xx];
```

```
            A:=OrdinaryCharacterTable(ConjugateGroup(B,
                                      C!.elts[bdry[i][2]]^-1));
            W:=Induced(StabIrrTable[n+1][k],A,Irr(
                                       StabIrrTable[n+1][k]));
            Coeffs:=MatScalarProducts(A,Irr(A),W);
            Add(Mat,[SignInt(x),xx,Coeffs]);
        od;
        return Mat;

    BoundaryRec:=[];
    for i in [1..N] do
        BoundaryRec[i]:=[];
        for j in [1..C!.dimension(i)] do
        Add(BoundaryRec[i],BoundaryMatrix(i,j));
        od;
    od;

    Boundary:=function(n,k)
    local w,x,y;
        w:=PairToTriple(n,k);
        x:=BoundaryRec[n][k];
    end;

    return Objectify(HapNonFreeResolution,
                rec(
                dimension:=Dimension,
                boundarymatrix:=BoundaryMatrix,
                boundary:=Boundary,
                homotopy:=fail,
                group:=Integers,
                properties:=
                [["length",1000],
                 ["characteristic",0],
                 ["type","resolution"]]  ));
end);
#################### end of BredonChainComplex ############################
```

5.3.29 FreeZGResolution(arg)

```
Note:    The first part of this function written by Graham Ellis
         The "homotpy" part is written by Bui Anh Tuan

##########################################################################
#O
#F  sl2zngens.gi
##  Input: A G-equivalent CW-space with resolutions for all stabilisers
##         a positive integer n.
##  Output: The first n terms of a free ZG-resolution of Z. If the input
##          together with a contracting homotopy then the output together
##          with a contracting homotopy.
##
InstallGlobalFunction(FreeZGResolution,
function(arg)
local
    P,N,prime,
    Dimension, DimensionRecord, DimRecs, FiltDimRecs, BinGp,
```

```
Boundary, BoundaryP, Pair2Quad, Pair2QuadRec,
Quad2Pair,Quad2PairRec, HtpyGen, HtpyWord,
StabGrps, StabResls, ResolutionFG,
Action, AlgRed,  EltsG, G, Mult, MultRecord,
DelGen, DelWord, DelGenRec,PseudoBoundary,FinalBoundary,
FilteredLength, FilteredDimension, FilteredDimensionRecord,
L,i,k,n,q,r,s,t,
InducedHtpyGen, InducedHtpyWord, DelListSum, #Added Feb 2014 BUI A.T.
Homotopy, HomotopyGen, NegateListWord, VertHtpy,
InducedHtpyList, IndHtpyRec;

SetInfoLevel(InfoWarning,0);

P:=arg[1];
N:=arg[2];
if Length(arg)>2 then
    prime:=Gcd(arg[3],EvaluateProperty(P,"characteristic"));
else
    prime:=EvaluateProperty(P,"characteristic");
fi;

N:=Minimum(EvaluateProperty(P,"length"),N);
G:=P!.group;
EltsG:=P!.elts;
BoundaryP:=P!.boundary;

BinGp:=ContractibleGcomplex("SL(2,O-2)");
BinGp:=BinGp!.stabilizer(0,4);;
BinGp:=Image(RegularActionHomomorphism(BinGp));

#########################################################################
#1
#F  ResolutionFG
##
##  Return resolutions for stabiliser groups if they are given
##  otherwise, find a new one.
##
##  Input:  group G and a positive integer n
##  Output: the first n+1 terms of a free ZG-resolution of Z
##
ResolutionFG:=function(G,n)
local
    x, tmp, iso,iso1,iso2,iso3,res,Q, fn;

    ##Added Jan 2012
    if IsBound(P!.resolutions) and HasName(G) then
        x:=Position(P!.resolutions[2], Name(G));
        if not x=fail then
            return P!.resolutions[1][x];
        fi;
    fi;

    if Order(G)=infinity and IsAbelian(G) then
        #This will only be correct if G is abelian of "rank" equal
        #to the number of generators GAP has for G

        res:=ResolutionGenericGroup(G,n);
        return res;
    fi;
```

```
    iso:=RegularActionHomomorphism(G);
    Q:=Image(iso);

    if IdGroup(Image(iso))=[24,3] then
        iso1:=IsomorphismGroups(Q,BinGp);
        res:=ResolutionFiniteGroup(BinGp,n);
        res!.group:=G;
        res!.elts:=List(res!.elts,x->
        PreImagesRepresentative(iso,PreImagesRepresentative(iso1,x)));
        return res;
    fi;

    res:=ResolutionFiniteGroup(Q,n);
    res!.group:=G;
    res!.elts:=List(res!.elts,x->PreImagesRepresentative(iso,x));
    return res;
end;
############### end of ResolutionFG ################################


if prime>0 then

    ######################################################################
    #1
    #F  AlgRed
    ##
    ##  Algebraic reduction for list of words mod p
    ##
    ##  Input:  a list of words
    ##  Output: reduced list of words
    ##
    AlgRed:= function(ww)
    local
        w,x,v,pos,u;

        w:=StructuralCopy(ww);

        v:=Collected(w);
        for x in v do
            if x[1][1]<0 then
                x[1][1]:=-x[1][1]; x[2]:=-x[2] mod prime;
            fi;
            if x[1][2]<0 then
                x[1][2]:=-x[1][2]; x[2]:=-x[2] mod prime;
            fi;
            x[2]:=x[2] mod prime;
        od;

        u:=[];
        for x in v do
            Append(u,MultiplyWord(x[2],[x[1]]));
        od;

        v:=Collected(u);
        for x in v do
            x[2]:=x[2] mod prime;
        od;

        u:=[];
```

```
        for x in v do
            Append(u,MultiplyWord(x[2],[x[1]]));
        od;

        return u;
    end;
    ######################### end of AlgRed ########################

else


    ##################################################################
    #1
    #F  AlgRed
    ##
    ##  Algebraic reduction for list of words
    ##
    ##  Input:  a list of words
    ##  Output: reduced list of words
    ##
    AlgRed:= function(ww)
    local x,i,v,k,u,w;
        w:=ww;#w:=StructuralCopy(ww);

        for x in w do
            if x[2]<0 then
                x[1]:=-x[1];x[2]:=-x[2];
            fi;
        od;
        v:=Filtered(w,x->x[1]>0);
        for x in w do
            if x[1]<0 then
                k:=Position(v,[-x[1],x[2],x[3]]);
                if (k=fail) then
                    Add(v,x);
                else
                    Unbind(v[k]);
                fi;
            fi;
        od;
        v:=Filtered(v,x->IsBound(x));

        return v;
    end;
    ##################################################################
fi;



#####################################################################
if IsBound(P!.action) and not prime=2 then
    Action:=P!.action;
else
    Action:=function(k,j,g) return 1; end;
fi;
#####################################################################

MultRecord:=[];
```

```
#########################################################################

#########################################################################
#1
#F  Mult
##
##  The product of 2 elements in EltsG
##
##  Input:  a pair of positive integers (g,h)
##  Output: the position of the product in the list EltsG
##
Mult:=function(g,h)
local pos;
    if not IsBound(MultRecord[g]) then
        MultRecord[g]:=[];
    fi;
    if not IsBound(MultRecord[g][h]) then
        pos:= Position(EltsG,EltsG[g]*EltsG[h]);
        if pos=fail then
            Add(EltsG,EltsG[g]*EltsG[h]);
            MultRecord[g][h]:= Length(EltsG);
        else
            MultRecord[g][h]:= pos;
        fi;
    fi;
return MultRecord[g][h];
end;
########################end of Mult#####################################

# Create a list of stabiliser subgroups and their resolutions

StabGrps:= List([0..Length(P)],n->
            List([1..P!.dimension(n)], k->P!.stabilizer(n,k)));

StabResls:=[];
i:=N;
if prime=0 then

    for L in StabGrps do
        Add(StabResls,List(L,
            g->ExtendScalars(ResolutionFG(g,i),G,EltsG))
            );
        i:=Maximum(0,AbsInt(i-1));
    od;
else
    for L in StabGrps do
        Add(StabResls,List(L,
        g->ExtendScalars(ResolutionFiniteGroup(g,i,false,prime),
        G,EltsG)));
        i:=Maximum(0,AbsInt(i-1));
    od;
fi;

DimRecs:=List([0..N],i->[]);

#########################################################################

#########################################################################
#1
```

```
#F  Dimension
##
##  Find the ZG-rank of R_n
##
##  Input:  a non-negative integer n
##  Output: ZG-rank of R_n
##
Dimension:=function(k)
local
    dim,i,R;
    dim:=0;
    for i in [0..k] do
        DimRecs[k+1][i+1]:=[];
        for R in StabResls[i+1] do
            dim:=dim+R!.dimension(k-i);
            Add(DimRecs[k+1][i+1],dim);
        od;
    od;
return dim;
end;
#########################################################################

DimensionRecord:=List([0..N],Dimension);

Dimension:=function(k);
    return DimensionRecord[k+1];
end;
#########################################################################

#########################################################################

# Create a record for the function Quad2Pair

Quad2PairRec:=[];
for q in [0..N] do
    Quad2PairRec[q+1]:=[];
    for r in [1..Length(StabGrps[q+1])] do
        Quad2PairRec[q+1][r]:=[];
        for s in [0..N-q] do
            Quad2PairRec[q+1][r][s+1]:=[];
        od;
    od;
od;
#########################################################################


#########################################################################
#1
#F  Pair2Quad
##
##  The n-th generator in degree k of our final resolution is
##  actually the t-th generator in degree s of the resolution
##  of the r-th stabilizer group of the q-th chain module of the
##  non-free resolution. We need the function f(k,n)=[q,r,s,t]
##
##  Input:  a pair of integers (k,n)
##  Output: [q,r,s,t]
##
Pair2Quad:=function(k,n)
```

```
local
    qq,q,r,s,t;

    for qq in [0..N] do
        if n <= DimRecs[k+1][qq+1][Length(DimRecs[k+1][qq+1])] then
            q:=qq; break;
        fi;
    od;

    r:=PositionProperty(DimRecs[k+1][q+1],x->(n<=x));
    s:=k-q;
    if r-1>0 then
        t:=n-DimRecs[k+1][q+1][r-1];
    else
        if q>=1 then
            t:=n-DimRecs[k+1][q][Length(  DimRecs[k+1][q] )];;
        else
            t:=n;
        fi;
    fi;
    Quad2PairRec[q+1][r][s+1][t]:=[k,n];
return [q,r,s,t];
end;
#####################end of Pair2Quad ##############################

# Create a record for the function Pair2Quad

Pair2QuadRec:=List([1..N+1],i->[]);
for k in [0..N] do
    for n in [1..Dimension(k)] do
        Pair2QuadRec[k+1][n]:=Pair2Quad(k,n);
    od;
od;
#########################################################################

#########################################################################
#1
#F  Pair2Quad
##
##  Input:  a pair of integers (k,n)
##  Output: [q,r,s,t]
##
Pair2Quad:=function(k,n)
local
    a;
    if n>0 then
        return StructuralCopy(Pair2QuadRec[k+1][n]);
    else
        a:=StructuralCopy(Pair2QuadRec[k+1][-n]);
        a[4]:=-a[4];
        return a;
    fi;
end;
#################end of Pair2Quad#####################################

#########################################################################
#1
#F  Quad2Pair
##
```

```
##  Input:  [q,r,s,t]
##  Output: [k,n]
##
Quad2Pair:=function(q,r,s,t)
local
    a,pr,pt;
    if r>0 then
        pr:=r;pt:=t;
    else
        pr:=-r;pt:=-t;
    fi;

    if pt>0 then
        return StructuralCopy(Quad2PairRec[q+1][pr][s+1][pt]);
    else
        a:=StructuralCopy(Quad2PairRec[q+1][pr][s+1][-pt]);
        a[2]:=-a[2];
        return a;
    fi;
end;
#######################end of Quad2pair ############################

#########################################################################
#1
#F  HtpyGen
##
##  This applies the "vertical homotopy" to the free group
##  generator [r,t,g] in "dimension" [q,s].
##  The output is an "r-word" in "dimension" [q,s+1].
##
##  Input:  a free generato [r,t,g] in demension [q,s]
##  Output: an "r-word" in "dimension" [q,s+1]
##
HtpyGen:=function(q,s,r,t,g)
local
    y,pr,pt;

    if r>0 then
        pr:=r;pt:=t;
    else
        pr:=-r;pt:=-t;
    fi;

    y:=StructuralCopy(StabResls[q+1][pr]!.homotopy(s,[pt,g]));
    Apply(y,x->[pr,x[1],x[2]]);
return y;
end;
#######################end of HtpyGen################################

#########################################################################
#1
#F  HtpyWord
##
##  This applies the "vertical homotopy" to the r-word w in
##  dimension [q,s]. The output is an r-word in "dimension" [q,s+1].
##
##  Input: r-word w in dimension [q,s]
##  Output: an r-word in "dimension" [q,s+1]
##
```

```
HtpyWord:=function(q,s,w)
local
    h,z,x,y;
#This applies the "vertical homotopy" to the r-word w in "dimension"
#[q,s]. The output is an r-word in "dimension" [q,s+1].

    h:=[];
    for y in w do
        x:=[Action(q,y[1],y[3])*y[1],y[2],y[3]];
        z:=HtpyGen(q,s,x[1],x[2],x[3]);
        z:=List(z,a->[Action(q,a[1],a[3])*a[1],a[2],a[3]]);
        Append(h,z);
    od;

return AlgRed(h);
end;
######################end of HtpyWord ##############################

# Create a record for function DelGen

DelGenRec:=[];
for k in [1..N+1] do
    DelGenRec[k]:=[];
    for q in [1..N+1] do
        DelGenRec[k][q]:=[];
        for s in [1..N+1] do
            DelGenRec[k][q][s]:=[];
            for r in [1..P!.dimension(q-1)] do
                DelGenRec[k][q][s][r]:=[];
            od;
        od;
    od;
od;
#########################################################################

#########################################################################
#1
#F  DelGen
##
##  For k=0,1,2 ... this is the equivariant homomorphism
##  Del_k:A_{q,s} ---> A_{q-k,s+k-1} applied to a free
##  r-generator [r,t] in dimension [q,s].
##
##  Input:  a positive integer k and
##          a free r-generator [r,t] in dimension [q,s]
##  Output: a word in dimension [q-k,s+k-1].
##
DelGen:=function(k,q,s,r,t)
local
    y,pr,pt,i;

    if r>0 then
        pr:=r;pt:=t;
    else
        pr:=-r;pt:=-t;
    fi;

    if IsBound(DelGenRec[k+1][q+1][s+1][pr][AbsInt(pt)]) then
```

```
        if pt>0 then return
            DelGenRec[k+1][q+1][s+1][pr][pt];
        else
            return List(DelGenRec[k+1][q+1][s+1][pr][-pt],
                                    a->[a[1],-a[2],a[3]]);
        fi;

    fi;

    if k=0 then
        if s=0 then
            return [];
        else
            y:=List(StabResls[q+1][pr]!.boundary(s,pt),
                    x->[Action(q,r,x[2])*x[1],x[2]]);
            if pt>0 then
                DelGenRec[k+1][q+1][s+1][pr][pt]:=
                        AlgRed(List(y,x->[pr,x[1],x[2]]));
                return DelGenRec[k+1][q+1][s+1][pr][pt];
            else
                DelGenRec[k+1][q+1][s+1][pr][-pt]:=
                        AlgRed(List(y,x->[pr,-x[1],x[2]]));
                return AlgRed(List(y,x->[pr,x[1],x[2]]));
            fi;
        fi;
    fi;

    if k=1 then
        if s=0 then
            if q=0 then
                return [];
            fi;
            y:=BoundaryP(q,pr);
            if pt>0 then
                DelGenRec[k+1][q+1][s+1][pr][pt]:=
                            AlgRed(List(y,x->[x[1],1,x[2]]));
                return DelGenRec[k+1][q+1][s+1][pr][pt];
            else
                DelGenRec[k+1][q+1][s+1][pr][-pt]:=
                            AlgRed(List(y,x->[x[1],1,x[2]]));
                return List(y,x->[x[1],-1,x[2]]);
            fi;

        else

            if pt>0 then
                DelGenRec[k+1][q+1][s+1][pr][pt]:=
                AlgRed(HtpyWord(q-1,s-1,DelWord(1,q,s-1,
                                    DelGen(0,q,s,pr,-pt)))) ;
                return DelGenRec[k+1][q+1][s+1][pr][pt];
            else
                DelGenRec[k+1][q+1][s+1][pr][-pt]:=
                AlgRed(HtpyWord(q-1,s-1,DelWord(1,q,s-1,
                                    DelGen(0,q,s,pr,pt)))) ;

                return
                    List(DelGenRec[k+1][q+1][s+1][pr][-pt],
                                    a->[a[1],-a[2],a[3]]);
            fi;
```

```
        fi;
    fi;

    y:=[];
    for i in [1..k] do
        Append(y,
        HtpyWord(q-k,s+k-2,DelWord(i,q-k+i,s+k-i-1,
                                   DelGen(k-i,q,s,pr,-pt)))
        );
    od;
    y:=AlgRed(y);

    if pt>0 then
        DelGenRec[k+1][q+1][s+1][pr][pt]:=y;
    else
        DelGenRec[k+1][q+1][s+1][pr][-pt]:=List(y,a->
                                    [a[1],-a[2],a[3]]);
    fi;

return y;
end;
##########################end of DelGen  ############################

#########################################################################
#1
#F  DelWord
##
##  The product of 2 elements in Elts
##
##  Input:  a pair of positive integers (i,j)
##  Output: the position of the product in the list Elts
##
DelWord:=function(k,q,s,w)
local
    y,x;
#For k=0,1,2 ... this is the equivariant homomorphism
#Del_k:A_{q,s} ---> A_{q-k,s+k-1} applied to an r-word [[r,t,g],...]
#in dimension [q,s].

    y:=[];
    for x in w do
        Append(y,List(DelGen(k,q,s,x[1],x[2]),
                              a->[a[1],a[2],Mult(x[3],a[3])]));
    od;

    return y;  #Added Jan 2013. Speeds up the calculation
               ## in some(!!) examples.
    return AlgRed(y);

end;
#########################################################################

#########################################################################
#1
#F  Boundary
##
##  Boundary map d_k: C_k -> C_{k-1}
##
```

```
##  Input:  a pair of positive integers (k,n)
##  Output: The boundary d_k(f_n)
##
Boundary:=function(k,n)
local
    q,s,r,t,x,y,z,i;
    y:=Pair2Quad(k,n);
    q:=y[1];s:=y[3];r:=y[2];t:=y[4];

    y:=[];

    for i in [0..k] do
        if q>=i then
            z:=DelGen(i,q,s,r,t);
            Append(y,List(z,x->
                    [Quad2Pair(q-i,x[1],s+i-1,x[2])[2],x[3]])  );
        else
            break;
        fi;
    od;

return AlgebraicReduction(y);
end;
#########################################################################

PseudoBoundary:=[];
for n in [1..N+1] do
    PseudoBoundary[n]:=[];
od;

#########################################################################
#1
#F  FinalBoundary
##
##  Boundary map d_k: C_k -> C_{k-1}
##
##  Input:  a pair of positive integers (k,n)
##  Output: the boundary d_k(f_n)
##
FinalBoundary:=function(n,k)
local
    pk;
    pk:=AbsInt(k);
    if  not IsBound(PseudoBoundary[n+1][pk]) then
        PseudoBoundary[n+1][pk]:= Boundary(n,pk);
    fi;
    if k>0 then
        return PseudoBoundary[n+1][k];
    else
        return NegateWord(PseudoBoundary[n+1][pk]);
    fi;
end;
#########################################################################



################spectral sequence requirements####################
```

```
FiltDimRecs:=[];
for k in [0..N] do
    FiltDimRecs[k+1]:=[];
    for i in [1..Dimension(k)] do
        FiltDimRecs[k+1][i]:=Pair2Quad(k,i)[1];
    od;
od;

FilteredLength:=Maximum(Flat(FiltDimRecs));

#########################################################################
FilteredDimension:=function(r,k);

    return Length(Filtered(FiltDimRecs[k+1],x->x<=r));

end;
#########################################################################



############ BUI ANH TUAN - FEB 2014 ###############################



#########################################################################
#1
#F  InducedHtpyGen
##
##  This constructs the "induced homotopy" h1 from
##  the given homotopy of the non-free complex
##   h1: A_qs ->A_{q+1}s
##  This applies the "induced homotopy" to the free group generator
##   [r,t,g] in "dimension" [q,s].
##  The output is an "r-word" in "dimension" [q+1,s].
##
##  Input:  five integers
##  Output: an r-word in dimension [q+1,s].
##
InducedHtpyGen:=function(q,s,r,t,g)
local
    y,pr,pt,w,v;

    if r>0 then
        pr:=r;pt:=t;
    else
        pr:=-r;pt:=-t;
    fi;

    y:=StructuralCopy(P!.homotopy(q,[pr,g]));
    if pt>0 then
        Apply(y,x->[x[1],1,x[2]]);
    else
        Apply(y,x->[x[1],-1,x[2]]);
    fi;
return y;
end;
#########################################################################

#########################################################################
```

```
#1
#F  InducedHtpyWord
##
##  This applies the "induced homotopy" to the r-word w in
##  "dimension" [q,s]. The output is an r-word in
##  "dimension" [q+1,s].
##
##  Input:  dimension q,s and a word w
##  Output: an r-word in dimension [q+1,s].
##
InducedHtpyWord:=function(q,s,w)
local
    h,z,x,y;


    h:=[];
    for y in w do
        x:=[y[1],y[2],y[3]];
        z:=StructuralCopy(InducedHtpyGen(q,s,x[1],x[2],x[3]));
        z:=List(z,a->[a[1],a[2],a[3]]);
        Append(h,z);
    od;

return AlgRed(h);
end;
#########################################################################

#########################################################################
#1
#F  InducedHtpyList
##
##  This applies the Horizontal Homotopy to a list of words
##  For each word, this applies the "induced homotopy" to
##  the r-word w in "dimension" [q,s].
##  The output is an r-word in "dimension" [q+1,s].
##
##  Input:  an r-word w in dimension [q,s]
##  Output: an r-word in dimension [q+1,s].
##
InducedHtpyList:=function(w)
local
    h,z,x,y,v,b;

    h:=[];
    for y in w do
        z:=StructuralCopy(InducedHtpyGen(y[1],y[2],y[3],y[4],y[5]));
        z:=List(z,a->[y[1]+1,y[2],a[1],a[2],a[3]]);
        Append(h,z);
    od;
    return h;
end;
#########################################################################

############ d+=d1+d2+..+dq of a list of words####################

#########################################################################
#1
#F  DelListSum
##
```

```
##  Sum of DelWord_k where k from 1 to q
##
##  Input:  a word w
##  Output: the differential map d(w)
##

DelListSum:=function(w)
local
    y,d,x,h,k;
    h:=[];
    for x in w do
        y:=[];
        for k in [1..x[1]] do
            d:=StructuralCopy(DelGen(k,x[1],x[2],x[3],x[4]));
            Apply(d,v->[x[1]-k,x[2]+k-1,v[1],v[2],Mult(x[5],v[3])]);
            Append(y,d);
        od;
        Append(h,y);
    od;

return h;
end;
#########################################################################

#########################################################################
#1
#F  VertHtpy
##
##  Applies to a list of [q,s,r,t,g]: could be in different A_qs
##  return a list of elements of the form [q,s,r,t,g]
##
##  Input:  [q,s,r,t,g]
##  Output: a list of elements of the form [q,s,r,t,g].
##
VertHtpy:=function(w)

local
    h,x,y,v;
    h:=[];;
    for x in w do
        v:=[x[1],x[2],Action(x[1],x[3],x[5])*x[3],x[4],x[5]];
        y:=StructuralCopy(HtpyGen(v[1],v[2],v[3],v[4],v[5]));
        Apply(y,a->[x[1],x[2]+1,Action(x[1],a[1],a[3])*a[1],a[2],a[3]]);
        Append(h,y);
    od;
return h;
end;
#########################################################################

#########################################################################
#1
#F  NegateListWord
##
##  Input:  a list of elements of the form [q,s,r,t,g]
##  Output: the negated word.
##

NegateListWord:=function(w)
    Apply(w,x->[x[1],x[2],-x[3],x[4],x[5]]);
```

```
return w;
end;
#########################################################################

#########################################################################
#1
#F  HomotopyGen
##
##  The product of 2 elements in Elts
##
##  Input:  [q,s,r,t,g]
##  Output: the homotopy h(q,s,r,t,g).
##
HomotopyGen:=function(arg)
local
    f,g,q,s,r,t,x,e,v,y,
    h0, h1, h0dh1, e3, h2, h;

    q:=arg[1];
    s:=arg[2];
    r:=arg[3];
    t:=arg[4];
    g:=arg[5];

    if arg=[] then
        return [];
    else
        if s = 0 then
            h1:=StructuralCopy(InducedHtpyList([[q,s,r,t,g]]));
            h0dh1:=StructuralCopy(VertHtpy(DelListSum(h1)));
            v:=StructuralCopy(DelListSum(h0dh1));
            e3:=[];          # e3=h(d+)h0(d+)d1
            for x in v do
                Append(e3,HomotopyGen(x[1],x[2],x[3],x[4],x[5]));
            od;
            e:=[];           # e=h1-h0(d+)h1+h(d+)h0(d+)h1
            Append(e,h1);
            Append(e,NegateListWord(h0dh1));
            Append(e,e3);
        elif s>0 then
        # s>0 then e=0
            e:=[];
        fi;
        y:=StructuralCopy([q,s,r,t,g]);

        h0:=VertHtpy([y]);

        v:=DelListSum(h0);
        h2:=[];          # h2=h(d+)h0
        for x in v do
            Append(h2,HomotopyGen(x[1],x[2],x[3],x[4],x[5]));
        od;
        h:=[];          # h=h0-d(d+)h0 + e
        Append(h,h0);
        Append(h,NegateListWord(h2));
        Append(h,e);
        return h;
    fi;
end;
```

```
    ########################################################################
    #1
    #F  Homotopy
    ##
    ##
    ##  Input:  degree k an a word w
    ##  Output: the homotopy h_k(w).
    ##

    Homotopy:=function(k,w)
    local
        f,g,q,s,r,t,v,e,h;

    ### h([])=[]
        if w=[] then
            return [];
        fi;
        f:=w[1];
        g:=w[2];
        v:=Pair2Quad(k,f);
        q:=v[1];
        r:=v[2];
        s:=v[3];
        t:=v[4];

        h:=HomotopyGen(q,s,r,t,g);
        Apply(h,x->[Quad2Pair(x[1],x[3],x[2],x[4])[2],x[5]]);
    return AlgebraicReduction(h);
    end;
    ########################################################################

SetInfoLevel(InfoWarning,1);

############################################################################
return Objectify(HapResolution,
                rec(
                inputresl:=P,
                verthtpy:=VertHtpy,
                htpy:=HtpyWord,
                delword:=DelWord,
                dimension:=Dimension,
                filteredDimension:=FilteredDimension,
                boundary:=FinalBoundary,
        inducedhomotopy:=InducedHtpyGen,
        stabrels:=StabResls,
                homotopy:=Homotopy,
                elts:=P!.elts,
                group:=P!.group,
                pseudoBoundary:=PseudoBoundary,
                properties:=
                   [["length",N],
                    ["filtration_length",FilteredLength],
                    ["initial_inclusion",true],
                    ["reduced",true],
                    ["type","resolution"],
                    ["characteristic",prime]  ]));
```

```
end);

################### end of FreeZGResolution ############################
```