Partially Commutative and Differential Graded Algebraic structures

ABDULSATAR JMAH THEIB AL-JUBURIE

Thesis submitted for the degree of
Doctor of Philosophy

School of Mathematics & Statistics
Newcastle University
Newcastle upon Tyne
United Kingdom

January 2015

This thesis is dedicated to my parents, my wife and my little daughter
for their love, endless support
and encouragement.

Acknowledgements

Foremost, I would like to express my special appreciation and thanks to my supervisor Dr. Andrew Duncan, you have been a tremendous mentor for me. I would like to thank you for sharing your knowledge, ideas, and limitless enthusiasm during my time as a graduate student. I would also like to thank you for encouraging my research and for allowing me to grow as a research scientist. Your advice on both research as well as on my career have been invaluable.

I would like to thank my second supervisor Dr. Stefan Kolb for his encouragement and guidance throughout my research.

In addition, I would like to thank my external examiner Dr. Alexander Konovalov and my internal examiner Professor Sarah Rees for their valuable expertise and comments on a previous draft of this thesis.

I would also extend my gratitude to the staff at Newcastle University and the School of Mathematics and Statistics for their professional handling of my student life which I thoroughly enjoyed.

A special thanks to my family. Words can not express how grateful I am to my mother, and father for all of the sacrifices that you have made on my behalf. Your prayer for me was what sustained me thus far.

I would also like to thank to my beloved wife, Zainab Al-Jumaili. Thank you for supporting me for everything, and especially I can't thank you enough for encouraging me throughout this experience. To my beloved daughter Maryam, I would like to express my thanks for being such a good girl always cheering me up.

I would like to thank my friends for their support, encouragement and understanding during the whole time of my study at the University of Newcastle.

Finally I thank my God, for letting me through all the difficulties. I have experienced Your guidance day by day. You are the One who let me finish my degree. I will keep on trusting You for my future. Thank you,

Abdulsatar.

Abstract

The objects of study in this thesis are partially commutative and differential graded algebraic structures. In fact my thesis is in two parts. The first on partially commutative algebraic structures is concerned with automorphism groups of partially commutative groups and their finite presentations. The second on differential graded algebraic structures is concerned with differential graded modules.

I have given a description for $Aut(G_\Gamma)$, the automorphism group of the partially commutative group G_Γ following Day's work, where Γ is a finite simple graph.

I have given a description for the subgroup $Conj(G_\Gamma)$ of automorphism group $Aut(G_\Gamma)$ following Toinet's work.

We have found a finite presentation for the subgroup $Conj_V$ of the automorphism group $Aut(G_\Gamma)$.

I have developed $AutParCommGrp$ (**Finite Presentations of Automorphism Groups of Partially Commutative Groups and Their Subgroups**) a package using the GAP system for computation of a finite presentation for $Aut(G_\Gamma)$, $Conj(G_\Gamma)$ and $Conj_V$ respectively.

In the second part of the thesis we consider the following situation: Let K be a field of characteristic two and let $R = K[x_1, x_2, \cdots, x_n]$ be a graded polynomial ring, graded in the negative way. Suppose M is a differential graded R-module with differential ∂ of degree P. We have constructed a classification for some types of differential graded R-module where $P \leq -2$, $n > 1$. This classification gives a partial algorithm to test whether such modules are solvable. For modules outside the classification we cannot decide, using our methods, whether or not they are solvable. Also, we have proved in one case that M is solvable when R is a graded polynomial ring, graded in the usual way (non-negatively graded) with ($P \geq 2$, $n > 1$). We have developed an algorithm and written a GAP package $SDGM$ (**Solvable Differential Graded Modules**) to check whether the differential graded R-module M with differential ∂ of degree P is solvable or not. Documentation has been written for all the packages above.

Contents

List of Figures

Glossary of Notation

Glossary of Notation	
$\mathit{\Gamma}$	a finite, simple, undirected graph with vertex set V
G	group
$G_{\mathit{\Gamma}}$	the partially commutative group with underlying graph $\mathit{\Gamma}$
pc group	partially commutative group
E	edge set of the simple graph $\mathit{\Gamma}$ (a list of pairs of vertices)
F_n	a free group of rank n
$\mathbb{Z}^n$	a free abelian group of rank n
$Aut(G_{\mathit{\Gamma}})$	the automorphism group of $G_{\mathit{\Gamma}}$
$\mathit{\Omega}$	the set of all Whitehead automorphisms of $G_{\mathit{\Gamma}}$
$\mathit{\Omega}_\ell$	the set of long-range elements of $\mathit{\Omega}$
$\mathit{\Omega}_s$	the set of short-range elements of $\mathit{\Omega}$
L	the union of V and its inverse V^{-1}, i.e., $L = V \cup V^{-1}$
$v(x)$	the vertex of x, be the unique element of $V \cap \{x, x^{-1}\}$ $\forall$ $x \in L$
$st(x)$	the star of the vertex x
$st(x)^{-1}$	a set of inverse elements of $st(x)$
$\ell k(x)$	the link of the vertex x
$\ell k(x)^{-1}$	a set of inverse elements of $\ell k(x)$
$st_L(x)$	the union of $st(v(x))$ and $st(v(x))^{-1}$
$\ell k_L(x)$	the union of $\ell k(v(x))$ and $\ell k(v(x))^{-1}$
$x \geq y$	the domination relation: say x dominates y if $\ell k(y) \subset st(x)$
$x \sim y$	elements x and y of V are equivalent: that is $st(x) = st(y)$
$[x]$	the equivalence class of x under $\sim$
$Aut(\mathit{\Gamma})$	the set of type (1) Whitehead automorphisms of $Aut(G_{\mathit{\Gamma}})$
$\mathit{\Omega}_1$	a special notation for the set of type (1) Whitehead automorphisms
$\mathit{\Omega}_2$	a special notation for the set of type (2) Whitehead automorphisms
(A, a)	a special notation for type (2) Whitehead automorphisms of $Aut(G_{\mathit{\Gamma}})$
$Y^{\perp}$	the orthogonal complement of Y in V
$cl(Y)$	the closure of Y in V, i.e. $cl(Y) = \cap_{z \in Y^{\perp}} st(z)$
$\mathfrak{a}(Y)$	the admissible set of Y, i.e. $\mathfrak{a}(Y) = \cap_{y \in Y}(st(y))^{\perp}$
$d(x, y)$	the distance from x to y; where $x, y \in \mathit{\Gamma}$
$Conj(G)$	the set of conjugating automorphisms of G
$Conj_N(G)$	the subgroup of all normal conjugating automorphisms
$Conj(G_{\mathit{\Gamma}})$	the subgroup of all basis conjugating automorphisms

Glossary of Notation	
$Conj_V(G_\Gamma)$	the subgroup of all vertex conjugating automorphisms
$LInn_S$	the set of all elementary conjugating automorphisms
$LInn_C$	the set of all basic collected conjugating automorphisms
$LInn_R$	the set of regular elementary conjugating automorphisms
$LInn_V$	the set of basic vertex conjugating automorphisms
$Conj_A(G)$	subgroup of $Conj(G)$ generated by all aggregate automorphisms
$Conj_S(G)$	the subgroup of $Conj(G)$ generated by $LInn_S$
$Conj_C(G)$	the subgroup of $Conj(G)$ generated by $LInn_C$
$Dom(x)$	the set of all vertices dominated by x
$Dom(\Gamma)$	the set of all dominated vertices
$out(y)$	set of all x such that $y \in Dom(x)$ and $[y] \neq [x]$ for fixed $y \in V$
$CAT(0)$	cube complexes
DGA	differential graded algebra
DG R-module	differential graded R-module
deg	abbreviation of degree
$f \simeq g$	the two maps f and g are homotopic
$H \leq G$	H is a subgroup of G
$H \lhd G$	H is a normal subgroup of G
$H \unlhd G$	H is a normal subgroup of or equal to G
$H \ntriangleleft G$	H is not normal subgroup of G
$G \cong H$	the two groups G and H are isomorphic
$Stab_G(s)$	the stabilizer of s in G
$Orb_G(s)$	the orbit of s under G
$[x, y]$	the commutator of x and y
$\ltimes$	the left normal factor semi-direct product
$\rtimes$	the right normal factor semi-direct product
$\oplus$	the direct sum
$\otimes$	the tensor product
$\cdot$	the dot product.

Part I

Partially Commutative Algebraic structures

Chapter 1

Introduction

Geometric group theory views algebraic objects as geometric objects. The graph is a geometric object whereas the group is an algebraic object. One relationship between graphs and groups was first observed by Cayley. A graph consists of a vertex set V and an edge set E. Historically, group concepts evolved in the context of geometry. German mathematician Felix Klein proposed a precise definition of geometry using group concepts "Geometry is the study of those properties of space which remain unchanged under a given group of transformations".

Partially commutative groups (pc groups "these are not the same as pc groups in GAP") have drawn much attention in geometric group theory, because of their rich subgroup structure and good algorithmic properties. These groups act on cubical complexes and have a variety of useful applications (see [16], [17], [39], [35] and [36] for example.) In recent times, the study of automorphism groups of partially commutative groups has been of great interest. We denote by $Aut(G)$ the automorphism group of a group G.

We will use Γ to denote a finite **simple graph**. We will write $V = V(\Gamma) = \{x_1, \ldots, x_n\}, (n \geq 1)$ for the finite set of vertices and $E = E(\Gamma) \subset V \times V$ for the set of edges, viewed as unordered pairs of vertices. The requirement that Γ be simple simply means that the diagonal of $V \times V$ is excluded from the set of edges. The **partially commutative group** (also known as a **right-angled Artin group**, a **trace group**, a **semi-free group** or a **graph group**) of Γ, is the group defined by presentation

$$G_\Gamma = \langle V | R_\Gamma \rangle$$

where the relations are

$$R_\Gamma = \{[x_i, x_j] \mid x_i, x_j \in V \text{ and } \{x_i, x_j\} \in E\}$$

where $[x_i\ , x_j] = x_i^{-1}x_j^{-1}x_ix_j$ and (x_i and x_j are **adjacent** if there exists an edge $e \in E$ with $e = \{x_i, x_j\}$). When Γ has no edges then G_Γ is free group of rank n, and when Γ is the complete graph then G_Γ is free abelian group of rank n. In general, partially commutative groups can be thought of as interpolating between these two extremes. Thus it seems reasonable to consider automorphism groups of partially commutative groups as interpolating between $Aut(F_n)$, the automorphism group of a free group, and $GL(n, \mathbb{Z})$, the automorphism group of a free abelian group.

A. Baudisch [8] first studied the partially commutative groups in the 1970's. Then C. Droms [28], [29], [30] further developed the theory in the 1980's and named them "graph groups". Since then, they have been widely studied (as is clear by the bibliography to this thesis.) For an introduction to this class of groups and a survey of the literature see [16]. For example, from Humphries [41] one knows that partially commutative groups are linear; their integral cohomology rings were computed early on by Kim and Roush [48], and Jensen and Meier [44] have extended this to include cohomology with group ring coefficients. More recently, Papadima and Suciu [62] have computed the lower central series, Chern groups and resonance varieties of these groups, while Charney, Crisp and Vogtmann [17] have explored their automorphism groups (in the triangle-free case) and Bestvina, Kleiner and Sageev [12] their rigidity properties. In [71] R. Wade has gave a description of Duchamp and Krob's extension of Magnus' approach to the lower central series of the free group to right-angled Artin groups.

The rich geometry of these groups is the feature that caused a significant interest in them. In [17], Charney and Davis construct an Eilenberg-MacLane space for each partially commutative group, which is a compact, non-positively curved, piecewise-Euclidean cube complex. Bestvina and Brady [11] have effectively applied geometric methods to the study of partially commutative groups. These groups can parametrized by finite simplicial complexes Σ satisfying a certain flag condition. There is heavy dependence of the Artin group associated to Σ on the combinatorial structure of Σ, not only in topology. Nevertheless, Bestvina and Brady show that the cohomological finiteness properties of the kernel of the canonical map onto $\mathbb{Z}$ are determined by the topology of Σ alone.

From Koberda [49] one knows that a partially commutative group is the universal group with specified commutation and noncommutation among its vertices. "For any subset $S \subset G$ of a group, we build the **commutation graph** of S, written $Comm(S)$, as follows. The vertices of $Comm(S)$ are the elements of S, and two vertices of S are connected by an edge if they commute in G". The following proposition gives the universal property of partially commutative groups.

Proposition 1.0.1. *[49] Let G be a group and let $S \subset G$ be a finite subset. The inclusion $S \subset G$ extends to a unique homomorphism*

$$G_{Comm(S)} \to G$$

which agrees with the identification $V(Comm(S)) \cong S$. In the universal property, we require S to be finite because partially commutative groups are defined to be finitely generated.

A finite generating set for $Aut(G_\Gamma)$ the automorphism group of a partially commutative group has been found by Servatius [69] and Laurence [51]. Over the last few years, significantly more has been discovered: Bux, Charney, Crisp and Vogtmann ([14], [17] and [19] for example) have shown that these automorphism groups are virtually torsion-free and have finite virtual cohomological dimension. Day has shown also that peak reduction techniques may be used on certain subsets of the generators and consequently has given a presentation for the automorphism group of partially commutative groups [24] and [27]. These groups, moreover, have a very rich subgroup structure. In other words, Gutierrez, Piggott and Ruane [40] were able to construct a semi-direct product decomposition for the more general case of automorphism groups of graph products of groups. In addition, Duncan, Remeslennikov and Kazachkov [34] provided a description of several arithmetic subgroups of the automorphism group of a partially commutative group. Noskov [60] also found different arithmetic subgroups. Providing certain conditions have made on the graph Γ, Charney and Vogtmann have shown [20] that the Tits alternative holds for the outer automorphism group of $G(\Gamma)$. Day [25] moreover, has shown that in all cases this group holds either a finite-index nilpotent subgroup or a non-Abelian free subgroup. Minasyan has shown [58] that partially commutative groups are conjugacy separable (loc. cit.) from which it can be shown that their outer automorphism groups are residually finite. Lohrey and Schleimer [53] have studied the compressed word problem and proved that the word problem for $Aut(G_\Gamma)$ is reducible to the compressed word problem for $G(\Gamma)$, i.e., the word problem for $Aut(G_\Gamma)$ has polynomial time complexity.

Charney and Farber [18], and then Day [26], have studied automorphism groups of partially commutative groups associated to random graphs, of Erdos-Renyi type. They have shown that if the edge probability (p) lies between 0.2929 and 1 and is constant then as the number of vertices (n) tends to ∞, the probability that the partially commutative group has finite outer automorphism group tends to 1.

Duncan, Remeslennikov and Remeslennikov [35] have defined several standard subgroups of the automorphism group $Aut(G_\Gamma)$ of a partially commutative group using the notion of admissible subset of a graph (see Section 4.1). The automorphism group of a partially commutative group G_Γ with commutative graph Γ contains a group $Aut^\Gamma(G_\Gamma)$ induced by isomorphisms of Γ. In Section 4.1 we introduce a particular subgroup $St^{conj}(\mathcal{K})$ and a subgroup $Aut^\Gamma_{comp}(G)$ of $Aut(\Gamma)$ (see Definitions 4.1.5, 4.1.6).

Theorem 1.0.2. *[35] The group $Aut(G)$ can be decomposed into the internal semi-direct product of the subgroup $St^{conj}(\mathcal{K})$ and the finite subgroup $Aut^\Gamma_{comp}(G)$, i.e.*

$$Aut(G) = St^{conj}(\mathcal{K}) \rtimes Aut^\Gamma_{comp}(G).$$

This theorem essentially reduces the problem of studying $Aut(G_\Gamma)$ to the study of the group $St^{conj}(\mathcal{K})$.

A **basis-conjugating** automorphism is one which maps each canonical generator x to x^{g_x}, for some $g_x \in G$. Toinet [70] has constructed a presentation for $Conj(G)$ the group of basis-conjugating automorphisms. Here we consider subgroups $Conj_N(G)$ of **normal conjugating automorphisms** (see Definition 4.1.7) and $Conj_V(G_\Gamma)$ of **vertex conjugating automorphisms** (see Section 4.1). We find a presentation for $Conj_V(G_\Gamma)$ of the automorphism groups of the partially commutative group $Aut(G_\Gamma)$.

Let G be a group with identity e and R be a ring with unit 1 different from 0. Then R is said to be G**-graded ring** if there exist additive subgroups R_g of R such that $R = \oplus \sum_{g \in G} R_g$ and $R_g R_h \subseteq R_{gh}$, for all $g, h \in G$.

Methods used in the study of graded rings have proved to be successful tools in the structure theory of commutative rings. Due to the great importance of grading of rings and modules, the study of this concept attracted wide interest from mathematicians everywhere. One of the mathematicians who studied the properties of grading of rings in general when G is a group or a subgroup was Jespers in [37] and [45]. On the other hand, M. Refai, carried out a number of studies about graded ring theory and graded modules (see for example [64], [66] and [65]).

A **differential graded category** (DG category) over the commutative ring R is a R-category $\mathcal{A}$ whose morphism spaces are differential graded R-modules (Definition 6.2.5) and whose compositions

$$\mathcal{A}(Y,Z) \otimes \mathcal{A}(X,Y) \to \mathcal{A}(X,Z), \quad (f,g) \mapsto fg$$

are morphisms of differential graded R-modules.

DG categories already appear in [47]. In the seventies, they found applications (see [67] and [31]) in the representation theory of finite-dimensional algebras. From B. Keller [46] one knows how the DG categories enhance our understanding of triangulated categories appearing in algebra and geometry. DG categories have been studied extensively since that time. For an introduction to the theory of DG category see [46].

A **differential graded algebra** (DG algebra) over the commutative ring R is a graded algebra, $A = \oplus_{i\in\mathbb{Z}} A_i$ over R together with a differential, that is a R-linear map $d : A \to A$ of degree -1 with $d^2 = 0$, satisfying the Leibniz rule $d(rs) = d(r)s + (-1)^{|r|} r d(s)$, where $r, s \in R$ and r is a graded element of degree $|r|$. We can think of DG algebras as generalisations of rings, so we have just gained more objects to work with. DG algebras, have been the object of considerable study in recent years, and a good picture of their properties has been built up through the work of many different researchers. For example, D. Dugger and B. Shipley [32] have investigated the relationship between DG algebras and topological ring spectra. M. Angel and R. Dlaz [4] have introduced the concept of N-differential graded algebras (N-dga), and study the moduli space of deformations of the differential of an N-dga. J. Jardine [43] has constructed a closed model structure for the category of non-commutative DG algebras over an arbitrary commutative ring with unit. Introductions to the theory of DG algebras can be found in [2], [6], [10] and [63].

Carlsson has studied properties of the **differential graded modules** (DG modules). In fact the **solvable** differential graded R-modules concept already appeared in the 1983's in work of G. Carlsson [15]. Recently, these modules have attracted much interest in ring theory, homological algebra, category theory, algebraic geometry and algebraic topology. For example, L. Avramov and D. Grayson [7] have shown that the duals of infinite projective resolutions of modules over a complete intersection are finitely generated DG modules over a graded polynomial ring. From X. Mao [55] one knows some new results on cone length of DG modules and global dimension of connected DG algebras. K. BECK [9] has investigated the image of the totaling functor, defined from the category of complexes of graded A-modules to the category of differential graded A-modules where A is a DG algebra with a trivial differential over a commutative unital ring. To each Λ_*-differential graded module A. Legrand [52] has associated "characteristic" classes which are invariants of the quasi-isomorphism class of this module and determined the Pontrjagin product by the zeroth and the first homology, where Λ_* is not necessarily a connected DG algebra.

The structure of this thesis is as follows: In Chapter 2, we present a background to partially commutative groups. We then give a description of the generating sets of automorphism groups of partially commutative groups. One of the commonly used generating sets of $Aut(G_\Gamma)$ is the set of Whitehead automorphisms. We describe the Whitehead automorphisms for partially commutative groups and the relations among Whitehead automorphisms. We develop a *GAP* package to find a finite presentation for the automorphism groups of partially commutative groups with a finite simple graph Γ. In order to do this we give a description of $Aut(G_\Gamma)$ according to Day's work in [24].

In Chapter 3, we give a description of the subgroup of basis-conjugating automorphisms $Conj(G_\Gamma)$ of $Aut(G_\Gamma)$ according to Toinet's work, in [70], and Day's work in [24]. We develop an algorithm and written a GAP package that provides a finite presentation for the subgroup $Conj(G_\Gamma)$.

In Chapter 4, we find a presentation for the subgroup $Conj_V$ of $Aut(G_\Gamma)$. We develop a *GAP* package that provides a finite presentation for $Conj_V$.

Chapter 5, contains some basic notions, definitions and results on exact homology sequences. Chapter 6, outlines the general principles of graded rings and some of their properties, as well as the definitions of graded algebras, and differential graded modules over the graded polynomial ring $R = K[x_1, x_2, \ldots, x_n]$.

In Chapter 7, we study composition series and then construct a classification for some types of differential graded R-modules, based on the degree P of the differential graded module and dimension of the module. This classification gives a partial algorithm to test whether such modules are solvable.

In Chapter 8 we give an algorithm implemented in GAP for all the cases covered in Chapter 7. This Chapter also includes a description of each function used in our algorithm.

Chapter 2

Finite Presentation for Automorphism Groups of pc Groups

2.1 Introduction

Partially commutative groups have drawn much attention in geometric group theory, because of their rich subgroup structure and good algorithmic properties, their actions on cubical complexes and their various applications. This chapter is concerned with automorphism groups of partially commutative groups and their finite presentations.

The GAP system will be used to find a finite presentation for the automorphism group of a partially commutative group. In order to do this work we will give a presentation for the automorphism group of a partially commutative group, according to Day's work in [24] and [27].

2.2 Background for pc groups

We will briefly describe the relationship between partially commutative groups, other Artin groups and Coxeter groups.

Definition 2.2.1. A **graph** Γ consists of

(i) a non-empty set $V(\Gamma)$ of **vertices** and

(ii) a set $E(\Gamma)$ of **edges**

such that every edge $e \in E(\Gamma)$ is a multiset $\{a, b\}$ of two vertices $a, b \in V(\Gamma)$.

$\Gamma = (V, E)$ will denote a graph with vertex and edge sets V and E (one or both of which may be infinite)

Vertices a and b are **adjacent** if there exists an edge $e \in E$ with $e = \{a, b\}$. If $e \in E$ and $e = \{c, d\}$ then e is said to be **incident** to c and to d and to **join** c and d. If a and b are vertices joined by edges $e_1, \ldots, e_k$, where $k > 1$, then $e_1, \ldots, e_k$ are called **multiple** edges.

Definition 2.2.2. An edge of the form $\{a, a\}$ is called a **loop**. A graph which has no multiple edges and no loops is called a **simple** graph.

Remark 2.2.3. A graph is **finite** if both its vertex set and edge set are finite. In this study we study only finite graphs, and so the term "graph" always means "finite graph". We call a graph with just one vertex **trivial** and all other graphs **nontrivial**. **All graphs in this thesis are finite and simple**. For an introduction to this class of graphs see [13] and [68].

Definition 2.2.4. [16] An **Artin group** A is a group with presentation of the form

$$A = \langle s_1, \ldots, s_n |\ \underbrace{s_i s_j s_i \ldots}_{m_{ij}} = \underbrace{s_j s_i s_j \ldots}_{m_{ji}} \text{ for all } i \neq j \rangle,$$

where $m_{ij} = m_{ji}$ is an integer ≥ 2 or $m_{ij} = \infty$ in which case we omit the relation between s_i and s_j. If we add to this presentation the additional relations $s_i = s_i^{-1}$ for all i, we obtain a **Coxeter group**

$$\begin{aligned} W &= \langle s_1, \ldots, s_n |\ s_i = s_i^{-1}, s_i s_j s_i \ldots = s_j s_i s_j \ldots \text{ for all } i \neq j \rangle \\ &= \langle s_1, \ldots, s_n |\ (s_i)^2 = 1, (s_i s_j)^{m_{ij}} = 1 \text{ for all } i \neq j \rangle. \end{aligned}$$

$D_\infty = \mathbb{Z}/2\mathbb{Z} * \mathbb{Z}/2\mathbb{Z}$ is an example of Coxeter group.

A **partially commutative group** (right-angled Artin group) is an Artin group in which $m_{ij} \in \{2, \infty\}$ for all i, j. In other words, in the presentation for the Artin group, all relations are commutator relations: $s_i s_j = s_j s_i$. Right-angled Coxeter groups are defined similarly. The easiest way to determine the presentation for a right-angled Coxeter or Artin group is by means of the **defining graph** (also called the **commutation graph**) Γ. This is the graph whose vertices are labeled by the generators $S = \{s_1, \ldots, s_n\}$ and whose edges connect a pair of vertices s_i, s_j if and only if $m_{ij} = 2$. Note that any finite, simple graph Γ is the defining graph for a right-angled Coxeter group W_Γ and a partially commutative groups G_Γ.

Theorem 2.2.5. *[16] Every partially commutative group embeds as a finite index subgroup of a right-angled Coxeter group.*

2.2.1 Partially Commutative Groups

Let Γ be a graph on n vertices, with vertex list V and a list of pairs of vertices E , i.e., $\Gamma = (V, E)$, where

$$V = \{x_1, \ldots, x_n\}$$

and

$$E = \{\{x_{i_1}, x_{i_2}\}, \ldots, \{x_{i_k}, x_{i_{k+1}}\}\}$$

Let G_Γ be the partially commutative group of Γ, defined by

$$G_\Gamma = \langle V | R_\Gamma \rangle$$

where the relations are

$$R_\Gamma = \{[x_i, x_j] \mid x_i, x_j \in V \text{ and } \{x_i, x_j\} \in E\}$$

where $[x_i, x_j] = x_i^{-1} x_j^{-1} x_i x_j$ and (x_i and x_j are **adjacent** if there exists an edge $e \in E$ with $e = \{x_i, x_j\}$). According to this construction we have the following two an important cases:

Firstly, if the graph Γ is the null graph (n vertices and no edges) then G_Γ is **free group** F_n of rank n. Secondly, if Γ is a complete graph on n vertices then G_Γ is the **free abelian group** $\mathbb{Z}^n$ of rank n. In general, G_Γ interpolates between these two extremes. Similarly, the automorphism group $Aut(G_\Gamma)$, the automorphism group of G_Γ interpolates between $Aut(F_n)$, the automorphism group of a free group, and $GL(n, \mathbb{Z})$, the automorphism group of a free abelian group. In fact the automorphism groups of partially commutative groups contain $Aut(F_n)$ and $GL(n, \mathbb{Z})$ and automorphism groups of free and direct products of $Aut(F_n)$ and $GL(n, \mathbb{Z})$. **From now on $Aut(G_\Gamma)$, denotes the automorphism group of G_Γ.**

Example 2.2.1.1

The following are a few examples of partially commutative groups:

(1) If Γ is a square as in Figure 2.1, then G_Γ decomposes as a direct product of two free groups $G_\Gamma \cong F(x,z) \times F(y,w)$.

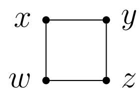

Figure 2.1: A Graph Γ

(2) If $\Gamma = P_3$, the path on three vertices then $G_\Gamma \cong F_2 \times \mathbb{Z}$.

(3) If Γ as in Figure 2.2, then $G_\Gamma \cong \mathbb{Z}^2 * \mathbb{Z}$.

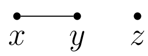

Figure 2.2: $G_\Gamma \cong \mathbb{Z}^2 * \mathbb{Z}$

(4) If Γ is an n-gon for $n \geq 5$, then G_Γ cannot be decomposed as either a direct product or a free product.

Remark 2.2.6. Let $L = V \cup V^{-1}$. For $x \in L$, we define $v(x) \in V$ the vertex of x, to be the unique element of $V \cap \{x, x^{-1}\}$. Hence $e = \{x, y\} = \{v(x), v(y)\}$ for each $x, y \in L$. The **star** of x denoted by $st(x)$ is a set of all the vertices that are connected directly to x by an edge, as well as the vertex x. The inverse of the star of x denoted by $st(x)^{-1}$ is the set of inverses of elements of $st(x)$. The **link** of denoted by $\ell k(x)$ is $st(x) \setminus \{x\}$, and the inverse of the link of x denoted by $\ell k(x)^{-1}$ is the set of inverses of elements of $\ell k(x)$. We set $st_L(x) = st(x) \cup st(x)^{-1}$ and $\ell k_L(x) = \ell k(x) \cup \ell k(x)^{-1}$.

Consider the graph of Γ of Figure 2.3 with $V = \{x, a, b, c, d, e, f, g\}$. Then we have that,

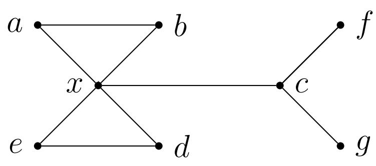

Figure 2.3: Graph of Γ

$L = V \cup V^{-1} = \{x, a, b, c, d, e, f, g, x^{-1}, a^{-1}, b^{-1}, c^{-1}, d^{-1}, e^{-1}, f^{-1}, g^{-1}\}$.
$st(x) = \{x, a, b, c, d, e\}$, $st(x)^{-1} = \{x^{-1}, a^{-1}, b^{-1}, c^{-1}, d^{-1}, e^{-1}\}$, and
$\ell k(x) = \{a, b, c, d, e\}$, $\ell k(x)^{-1} = \{a^{-1}, b^{-1}, c^{-1}, d^{-1}, e^{-1}\}$. Hence,
$st_L(x) = st(x) \cup st(x)^{-1} = \{x, a, b, c, d, e, x^{-1}, a^{-1}, b^{-1}, c^{-1}, d^{-1}, e^{-1}\}$ and
$\ell k_L(x) = \ell k(x) \cup \ell k(x)^{-1} = \{a, b, c, d, e, f, a^{-1}, b^{-1}, c^{-1}, d^{-1}, e^{-1}\}$.

2.3 Combinatorial group theory of partially commutative groups

Let the set of letters L be $V \cup V^{-1}$. Recall that a **word** in L is a finite sequence of elements of L and every word in L represents an element of G_Γ. By a **cyclic word** w we mean the set consisting of w and all cyclic permutations of the sequence of letters of w. For example, xyy is a word and the corresponding cyclic word is $\{xyy, yyx, yxy\}$.

Any two elements of a cyclic word represent group elements that are conjugate to each other, so a cyclic word represents a well-defined conjugacy class, we say a **conjugate** to b denoted $a \sim b$ if there exists g such that $g^{-1}ag = b$. Now, if we pick any two elements of a cyclic word as in our example above then these are conjugate to each other:

$(yy)xyy(yy)^{-1} = yyx$,
$(y^{-1})yyx(y) = yxy$,
$(y^{-1})yxy(y) = xyy$.

If w is a cyclic word, we will use (w) to denote the set of all cyclic permutations of w (it is the image of w under a cyclic permutation.) A word w on L is **graphically reduced** if it contains no subsegments of the form aua^{-1}, where $a \in L$ and u is a word in $\langle \ell k_L(a) \rangle$ (because in this case $aua^{-1} = u$ in G_Γ, so $w_1aua^{-1}w_2 = w_1uw_2$ in G_Γ, for all words w_1, w_2). A cyclic word is **graphically reduced** if all its elements are graphically reduced as words. If we consider the graph Γ of Figure 2.4 then we have that,

$L = V \cup V^{-1} = \{a, x_1, x_2, x_3, x_4, a^{-1}, x_1^{-1}, x_2^{-1}, x_3^{-1}, x_4^{-1}\}$,
$G_\Gamma = \langle V | R_\Gamma \rangle$,
$\ell k_L(a) = \{x_1, x_2, x_3, x_1^{-1}, x_2^{-1}, x_3^{-1}\}$,
$R_\Gamma = \{[a, x_1], [a, x_2], [a, x_3], [x_3, x_4]\}$,

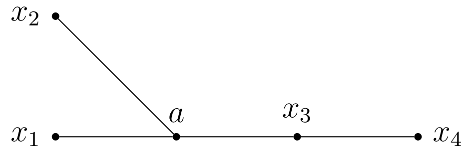

Figure 2.4: Graph of Γ

so we have $ax_1a^{-1} = x_1, ax_2a^{-1} = x_2, ax_3a^{-1} = x_3$, and $x_3x_4x_3^{-1} = x_4$.

Now if we pick any word u in $\langle \ell k_L(a) \rangle$, let we say $u = x_1x_2x_1^{-1}$, then $aua^{-1} = ax_1x_2x_1^{-1}a^{-1} = x_1x_2x_1^{-1}$.

If w is a word in L then the support of w is the set of letters $x \in V$ such that x or x^{-1} accurs in w, denoted $supp(w)$. By Baudisch [8] if w and w' are reduced words representing the same element of G_Γ then $supp(w) = supp(w')$. Therefore we make the following definition.

Definition 2.3.1. For an element g of G_Γ, the **support** of g is

$$supp(g) = supp(v) \text{ where } v \text{ is a reduced word representing } g.$$

The support $supp(w)$ of a k-tuple $W = (w_1, \ldots, w_k)$ of conjugacy classes is $\bigcup_{i=1}^{k} supp(w_i)$.

By Baudisch [8] if w and w' are graphically reduced words and represent the same element of G_Γ then the lengths of w and w' are equal. Therefore we define the **length** of an element g of G_Γ to be the length of any graphically reduced word representing g. We say that an element g in G_Γ is **cyclically reduced** if it can not be written as vhv^{-1} or $v^{-1}hv$ with $v \in V$, and $|g| = |h| + 2$. By [69], Proposition 2, every element of G_Γ is conjugate to a unique (up to cyclic permutation) cyclically reduced element. The **length of a conjugacy class** is defined to be the minimal length of any of its representative elements. Observe that the length of a conjugacy class is equal to the length of a cyclically reduced element representing it. For an n-tuple of conjugacy classes W, we define the length of W, denoted by $|W|$, as the sum of the length of its elements ($n \geq 1$).

2.4 Automorphisms of pc groups

In this section we shall give the definition of Laurence-Servatius generators for $Aut(G_\Gamma)$. We shall also give the definition of Whitehead automorphisms for partially commutative groups. Some other definitions and concepts that are important in our study will be given.

2.4.1 Laurence's generators for $Aut(G_\Gamma)$

We will state some definitions and concepts that are important in our study before we give the definition of Laurence-Servatius generators for $Aut(G_\Gamma)$.

1. There is a reflexive and transitive binary relation on V called the **domination relation**: $x \geq y$ (x **dominates** y) iff $\ell k(y) \subset st(x)$.

2. Domination is clearly reflexive and transitive, since $\ell k(x) \subset st(x)$, so $x \geq x$ and this implies that the domination is reflexive. Now, domination is transitive, because that if we have $x \geq y$ and $y \geq z$ then we have that $\ell k(z) \subset st(y)$ and $\ell k(y) \subset st(x)$. So we have two cases:

   (a) If $y \notin \ell k(z)$, since $\ell k(z) \subset st(y)$ and $y \notin \ell k(z)$, then we will get that $\ell k(z) \subset \ell k(y)$, which implies to $\ell k(z) \subset \ell k(y) \subset st(x)$, implies to $x \geq z$.

   (b) If $y \in \ell k(z)$, as case(1), $\ell k(z) \setminus \{y\} \subset \ell k(y) \subset st(x)$. So if we prove that, $y \in st(x)$ then $\ell k(z) \subset st(x)$. Note that, since $y \in \ell k(z)$ then we have the edge $e_1 = \{z, y\}$, and since $\ell k(y) \subset st(x)$ then we have the edge $e_2 = \{z, x\}$,also since $\ell k(z) \subset st(y)$ then we have the edge $e_3 = \{y, x\}$. Therefore, $y \in st(x)$, and hence $x \geq z$. Thus domination is transitive.

3. For $x, y \in L, say\ x \geq y\ if\ v(x) \geq v(y)$.

4. Write $x \sim y$ when $x \geq y$ and $y \geq x$; the relation $\sim$ is called the **domination equivalence** relation.

5. The **adjacent domination** relation, which holds for x and y if $\{x, y\} \in E$ (or $[x, y] \in R_\Gamma$) and $x \leq y$.

6. The **non-adjacent domination** relation, which holds for x and y if $x \leq y$ $\{x, y\} \notin E$ (or $[x, y] \notin R_\Gamma$).

7. We say that x **strictly dominates** y if $x \geq y$ and $x \not\sim y$.

Definition 2.4.1. [51] and [69] The **Laurence-Servatius** generators for $Aut(G_\Gamma)$ are the following four classes of automorphisms:

1. **Transvections:** For $x, y \in L$ with $x \geq y$ and $v(x) \neq v(y)$, the **transvection** $\tau_{x,y}$ is the map that sends

$$y \mapsto yx$$

and fixes all generators not equal to $v(y)$. A transvection $\tau_{x,y}$ determines an automorphism of G_Γ (see [51], [69]).

2. **Partial Conjugations:** An automorphism $c_{x,Y}$, for $x \in L$ and Y a non-empty union of connected components of $\Gamma \backslash st(x)$, that maps each $y \in Y$ to $x^{-1}yx$ and fixes all generators not in Y is called a **partial conjugation**. The set $Conj(G_\Gamma) = Conj$ of all partial conjugations forms a subgroup of G_Γ. Every partial conjugation determines an automorphism of G_Γ ([51], [69]). For example in the graph of Γ of Figure 2.5 we have a partial conjugation $y_i \mapsto x^{-1}y_ix$, $i = 1, 2$, $b \mapsto b$, $c \mapsto c$, $a \mapsto a$, $d \mapsto d$, $x \mapsto x$.

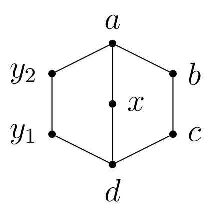

Figure 2.5: Graph of Γ

In particular if $Y = \Gamma \backslash st(x)$ then $c_{x,Y}$ is the **inner automorphism** γ_x sending u to u^x for all $x \in V$.

3. **Inversions:** For $x \in V$, the **inversion** τ_x of x is the map that sends

$$x \mapsto x^{-1}$$

and fixes all other generators. i.e., inversions send a standard generator of G_Γ to its inverse. Every inversion determines an automorphism of G_Γ ([51], [69]).

4. **Graphic Automorphisms:** For π an automorphism of the graph Γ, the **graphic automorphism** of G_Γ is determined by π is the map that sends

$$x \mapsto \pi(x)$$

for each generator $x \in X$, (**An automorphism of a graph** $G = (V, E)$ is a permutation σ of the vertex set V, such that the pair of vertices $\{u, v\}$ forms an edge if and only if the pair $\{\sigma(u), \sigma(v)\}$ also forms an edge.) Every graphic automorphism is an automorphism of G_Γ ([51], [69]) and the set of all graphic automorphisms of $Aut(G_\Gamma)$ is denoted $Aut^\Gamma(G_\Gamma)$.

Theorem 2.4.2. *[51] The group $Aut(G_\Gamma)$ is generated by the finite set consisting of all transvections, partial conjugations, inversions and graphic automorphisms of G_Γ. The subgroup $Conj(G_\Gamma)$ is generated by the partial conjugations.*

A finite presentation for the subgroup $Conj(G_\Gamma)$ of $Aut(G_\Gamma)$ is given in [70].

2.4.2 Whitehead automorphisms for partially commutative groups

Definition 2.4.3. A Whitehead automorphism is an element $\alpha \in Aut(G_\Gamma)$ of one of the following two types:

Type (1): α restricted to $V \cup V^{-1}$ is a permutation of $V \cup V^{-1}$, or

Type (2): there is an element $a \in V \cup V^{-1}$, called the multiplier of α, such that for each $x \in V$ the element $\alpha(x)$ is one of $x, xa, a^{-1}x, a^{-1}xa$.

Let Ω be the set of all Whitehead automorphisms of G_Γ.

Definition 2.4.4. A Whitehead automorphism $\alpha \in \Omega$ is **long-range** if α is of type (1) or if α is of type (2) with multiplier $a \in V \cup V^{-1}$ and α fixes the elements of V adjacent to a in Γ. Let Ω_ℓ be the set of long-range elements of Ω.

A Whitehead automorphism $\alpha \in \Omega$ is **short-range** if α is of type (2) with multiplier $a \in V \cup V^{-1}$ and α fixes the elements of V not adjacent to a in Γ. Let Ω_s be the set of short-range elements of Ω.

By [51] (see Section 2.2), we can conclude that $\Omega_\ell \cup \Omega_s$ is a generating set for $Aut(G_\Gamma)$.

Theorem 2.4.5. *[24] For any graph Γ, the group $Aut(G_\Gamma)$ is finitely presented. Specifically, there is a finite set R of relations among the Whitehead automorphisms Ω such that $Aut(G_\Gamma) = \langle \Omega, R \rangle$.*

There is a special notation for type (2) Whitehead automorphisms. Let $A \subset L$ and $a \in L$, such that $a \in A$ and $a^{-1} \notin A$. If it exists, the symbol (A, a) denotes the Whitehead automorphism satisfying

$$(A, a)(a) = a$$

and for $x \in V \backslash v(a)$:

$$(A, a)(x) = \begin{cases} x & \text{if } x \notin A \text{ and } x^{-1} \notin A \\ xa & \text{if } x \in A \text{ and } x^{-1} \notin A \\ a^{-1}x & \text{if } x \notin A \text{ and } x^{-1} \in A \\ a^{-1}xa & \text{if } x \in A \text{ and } x^{-1} \in A \end{cases}$$

Say that (A, a) is well defined if the formula given above defines an automorphism of G_Γ.

Note:

i. For $\alpha \in \Omega$ of type (2), one can always find a multiplier $a \in L$ and a subset $A \subset L$ such that $\alpha = (A, a)$. There is a little ambiguity in choosing such a representation that comes from the following fact: if $a, b \in L$ with $e = \{a, b\}$, then $(\{a, b, b^{-1}\}, a)$ is the trivial automorphism. In another word if b and $b^{-1} \in \ell k_L$ then we must delete them from the set A, because they cancel each other.

ii. The set of type (1) Whitehead automorphisms is the finite subgroup of $Aut(G_\Gamma)$ generated by the graphic automorphisms and inversions.

iii. The set Ω of Whitehead automorphisms is a finite generating set of $Aut(G_\Gamma)$.

Lemma 2.4.6. *[24] For $A \subset L$ with $a \in A$ and $a^{-1} \notin A$, the automorphism (A, a) is well defined if and only if both of the following hold:*

1. *The set $(V \cap A \cap A^{-1}) \backslash 1k(v(a))$ is a union of connected components of $\Gamma \backslash st(a)$.*

2. *For each $x \in (A \backslash A^{-1})$, we have $a \geq x$.*

Alternatively, (A, a) is well defined if and only if for each $x \in A \backslash st_L(a)$ with $a \not\geq x, (A, a)$ acts on the entire component of $x \in \Gamma \backslash st(a)$ by conjugation.

2.5 Relations among Whitehead automorphisms

In this section we define the set of relations R in Theorem 2.4.5. Note that we use function composition order and automorphisms act on the left with sets. We use the notation $A+B$ for $A \cup B$ when $A \cap B = \emptyset$. Note the shorthand $A - a$ for $A \backslash \{a\}$ and $A + a$ for $A \cup \{a\}$.

Let Φ be the free group generated by the set Ω. We understand the relation "$w_1 = w_2''$ to correspond to $w_1 w_2^{-1} \in \Phi$. Note that if $(A, a) \in \Omega$ with $B \subset \ell k(v(a))$ and $(B \cup B^{-1}) \cap A = \emptyset$, then (A, a) and $(A + B + B^{-1}, a)$ represent the same element of Ω and therefore the same element of Φ. This is why we do not list "$(A, a) = (A + B + B^{-1}, a)''$ in the relations below. We illustrate this by the following example:

Let Γ be a graph of Figure 2.6 with the set of vertices, $V = \{a, b, c, d, e, f, g\}$

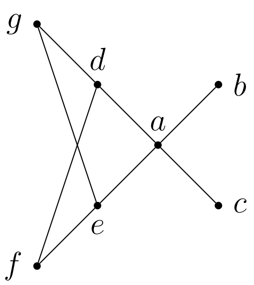

Figure 2.6: Graph of Γ

Let $(A, a) = (\{a, b, b^{-1}\}, a) \in \Omega$. So,
$A = \{a, b, b^{-1}\}$,
$\ell k(v(a)) = \{b, c, d, e\}$.
Let $B = \{d, e\} \subset \ell k(v(a))$ and so $B^{-1} = \{d^{-1}, e^{-1}\}$. From the above we have,
$(A, a)(a) = a, \;\; (A + B + B^{-1}, a)(a) = a$, and for $x \in V \backslash v(a)$, we have
$(A, a)(b) = a^{-1}ba = a^{-1}ab = b$ and $(A + B + B^{-1}, a)(b) = a^{-1}ba = a^{-1}ba = b$,
(since $[a, b] = 1 \Rightarrow ab = ba$),
$(A, a)(c) = c, \qquad (A + B + B^{-1}, a)(c) = c$,
$(A, a)(d) = d, \qquad (A + B + B^{-1}, a)(d) = a^{-1}da = a^{-1}ad = d$,
$(A, a)(e) = e, \qquad (A + B + B^{-1}, a)(e) = a^{-1}ea = a^{-1}ae = e$,

$(A, a)(f) = f, \qquad (A + B + B^{-1}, a)(f) = f,$
$(A, a)(g) = g, \qquad (A + B + B^{-1}, a)(g) = g.$
Hence, $(A, a) = (A + B + B^{-1}, a)$.

Definition 2.5.1. [24] There are ten types of relations as follows:

(R1) $(A, a)^{-1} = (A - a + a^{-1}, a^{-1})$

for $(A, a) \in \Omega$.

(R2) $(A, a)(B, a) = (A \cup B, a)$

for (A, a) and $(B, a) \in \Omega$ with $A \cap B = \{a\}$.

(R3) $(B, b)(A, a)(B, b)^{-1} = (A, a)$

for (A, a) and $(B, b) \in \Omega$ such that $a \notin B, b \notin A, a^{-1} \notin B, b^{-1} \notin A$, and at least one of $(a) A \cap B = \emptyset$ or $(b) b \in 1k_L(a)$ holds. We refer to this relation as (R3a) if condition (a) holds and (R3b) if condition (b) holds.

(R4) $(B, b)(A, a)(B, b)^{-1} = (A, a)(B - b + a, a)$

for $(A, a) \in \Omega$ and $(B, b) \in \Omega$ such that $a \notin B, b \notin A, a^{-1} \notin B, b^{-1} \in A$, and at least one of $(a) A \cap B = \emptyset$ or $(b) b \in \ell k_L(a)$ holds. We refer to this relation as (R4a) if condition (a) holds and (R4b) if condition (b) holds.

(R5) $(A - a + a^{-1}, b)(A, a) = (A - b + b^{-1}, a)\tau_b(a, b)$

where $\tau_b \in I$ and (a, b) is the graphic automorphism transposing a and b; with $(A, a) \in \Omega$, $b \in A$, $b^{-1} \notin A$, $b \neq a$, $b \sim a$.

(R6) There are two types of $R6$ relation which are,

(R6a) $\tau_x(A, a)\tau_x^{-1} = (\tau_x(A), \tau_x(a))$, where $\tau_x \in I$, and

(R6b) $\phi(A, a)\phi^{-1} = (\phi(A), \phi(a))$, where $\phi \in Aut(G_\Gamma)$.

(R7) The entire multiplication table of the type (1) Whitehead automorphisms, which forms a finite subgroup of Aut G_Γ.

(R8) $(A, a) = (L - a^{-1}, a)(L - A, a^{-1})$,

for $(A, a) \in \Omega$.

(R9) $(A, a)(L - b^{-1}, b)(A, a)^{-1} = (L - b^{-1}, b)$,

for $(A, a) \in \Omega$ and $b \in L$ with $b, b^{-1} \notin A$.

(R10) $(A,a)(L-b^{-1},b)(A,a)^{-1} = (L-a^{-1},a)(L-b^{-1},b)$

for $(A,a) \in \Omega$ and $b \in L$ with $b \in A$, $b^{-1} \notin A$ and $b \neq a$.

Let R be the set of elements of Φ corresponding to all relations of the forms $(R1)$, $(R2)$, $(R3)$, $(R4)$, $(R5)$, $(R6)$, $(R7)$, $(R8)$, $(R9)$, $(R10)$. This is the same R in Theorem 3.3.9 and Day [24] proved in Section 5 that:

$$Aut(G_\Gamma) := \langle \Omega | R_\Gamma \rangle .$$

2.5.1 Relations $R5$ and $R6$

In Day's work relations $(R5)$ and$(R6)$ are not the same as the ones in the Definition 2.5.1. Our alternative forms for the relations $(R5)$ and$(R6)$ are more suitable for our algorithm. In this section we show that our relations $(R5)$ and$(R6)$ are equivalent to Day's relations $(R5)$ and$(R6)$. Day's $(R5)$ and$(R6)$ are

(R'5) $(A-a+a^{-1},b)(A,a) = (A-b+b^{-1},a)\sigma_{a,b}$

for $(A,a) \in \Omega$ and $b \in A$ with $b^{-1} \notin A, b \neq a$, and $b \sim a$, where $\sigma_{a,b}$ is the type (1) Whitehead automorphism with $\sigma_{a,b}(a) = b^{-1}, \sigma_{a,b}(b) = a$ and which fixes the other generations.

(R'6) $\sigma(A,a)\sigma^{-1} = (\sigma(A), \sigma(a))$

for $(A,a) \in \Omega$ of type (2) and $\sigma \in \Omega$ of type (1).

First, we will give an example for small graph and after that we will go to the general case.

Example 2.5.1.1
Let $V = \{x_1, x_2, x_3, x_4, x_5\}$ be the set of vertices and Γ be a graph of Figure 2.7:

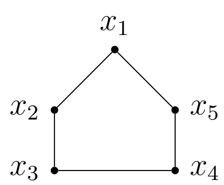

Figure 2.7: A Graph Γ

We have graph isomorphism π such that,

$$\pi = \begin{tikzcd} x_1 \arrow[r] & x_2 \arrow[r] & x_3 \arrow[r] & x_4 \arrow[r] & x_5 \arrow[llll, bend left] \end{tikzcd}$$

and another ρ such that

$$\rho = \begin{cases} x_4 \to x_4 \\ x_3 \to x_5 \\ x_5 \to x_3 \\ x_1 \to x_2 \\ x_2 \to x_1 \end{cases}$$

In this example the isomorphism group of Γ is generated by π and ρ (and is isomorphic to dihedral group D_5).

- $G(\Gamma) = \langle x_1, x_2, x_3, x_4, x_5 \rangle \mid [x_1, x_2] = [x_2, x_3] = [x_3, x_4] = [x_4, x_5]$
  $= [x_5, x_1] = 1\rangle$
- $V \cup V^{-1} = \{x_1, x_2, x_3, x_4, x_5, x_1^{-1}, x_2^{-1}, x_3^{-1}, x_4^{-1}, x_5^{-1}\}$.

Now, let $\theta \in Aut(G_\Gamma)$ be an automorphism of type (1), so θ permutes $V \cup V^{-1}$. Let $x \in V(\Gamma)$ then $\theta(x) = y \in V \cup V^{-1}$. Since $\theta \in Aut(G_\Gamma)$ then $\theta(x^{-1}) = \theta(x)^{-1} = y^{-1}$. Therefore, $\theta(\{x, x^{-1}\}) = \{y, y^{-1}\}$.

Group $V \cup V^{-1}$ into pairs $\{x_1, x_1^{-1}\}, \{x_2, x_2^{-1}\}, \ldots, \{x_n, x_n^{-1}\}$ and then for each i, θ maps $\{x_i, x_i^{-1}\}$ to $\{x_j, x_j^{-1}\}$ for some j. So, θ is a permutation of the set of pairs $\{x_1, x_1^{-1}\}, \ldots, \{x_n, x_n^{-1}\}$. In this case, if we forget the exponent ± 1 of x_i we may use θ to define an automorphism θ_0 of Γ. Namely if $\theta(\{x_i, x_i^{-1}\}) = \{x_j, x_j^{+1}\}$ define $\theta_0(x_i) = x_j$. In this case we say θ **contracts** to θ_0. For example, let θ be such that

$$\begin{tikzcd} x_1 \arrow[r] & x_2 \arrow[r] & x_3 \arrow[r] & x_4 \arrow[r] & x_5 \arrow[llll, bend left] \end{tikzcd}$$

Then θ contracts to the automorphism π above.

Conversely an automorphism α of Γ induces several automorphisms of G_Γ which contract to α. In fact if $\alpha(x_i) = x_j$ then we my define an automorphism θ of G_Γ such that (a) $\theta(x_i) = x_j$ or (b) $\theta(x_i) = x_j^{-1}$. Suppose θ is defined by making such a choice $\forall x_i \in V(\Gamma)$. Since θ is obtained from α by composition with appropriate inversions, it follows that θ determines an automorphism of G_Γ. Moreover, by definition θ contracts to α. As there are two choices for $\theta(x_i)$, for $i = 1, \ldots, 5$, every $\alpha \in Aut(\Gamma)$ induces at most 2^n distinct elements of $Aut(G_\Gamma)$.

In the example above, ρ gives rise to at most 2^5 automorphisms of type (1). So, we have that

$$\bullet\ \{x_1, x_1^{-1}\} \to \{x_2, x_2^{-1}\} \Rightarrow \left\{ \begin{array}{l} \xrightarrow{a} \begin{cases} x_1 \to x_2 \\ x_1^{-1} \to x_2^{-1} \end{cases} \\ \xrightarrow{b} \begin{cases} x_1 \to x_2^{-1} \\ x_1^{-1} \to x_2 \end{cases} \end{array} \right.$$

For each of a,b we have that

$$\bullet\ \{x_2, x_2^{-1}\} \to \{x_1, x_1^{-1}\} \Rightarrow \left\{ \begin{array}{l} \xrightarrow{c} \begin{cases} x_2 \to x_1 \\ x_2^{-1} \to x_1^{-1} \end{cases} \\ \xrightarrow{d} \begin{cases} x_2 \to x_1^{-1} \\ x_2^{-1} \to x_1 \end{cases} \end{array} \right.$$

- If we have a and c:

$$\rho_1 : \begin{cases} x_1 \leftrightarrows x_2 \\ x_1^{-1} \leftrightarrows x_2^{-1} \end{cases}$$

- If we have a and d:

$$\begin{tikzcd} \rho_2 : x_1 \arrow[r] & x_2 \arrow[r] & x_1^{-1} \arrow[r] & x_2^{-1} \arrow[lll, bend left] \end{tikzcd}$$

- If we have b and c:

$$\begin{tikzcd} \rho_3 : x_1 \arrow[r] & x_2^{-1} \arrow[r] & x_1^{-1} \arrow[r] & x_2 \arrow[lll, bend left] \end{tikzcd}$$

- If we have b and d:

$$\rho_4 : \begin{cases} x_1 \leftrightarrows x_2^{-1} \\ x_1^{-1} \leftrightarrows x_2 \end{cases}$$

$$\rho_5 : \{x_3, x_3^{-1}\} \to \{x_5, x_5^{-1}\}$$

For ρ_5 there are two possibilities

$$\bullet\ \{x_3, x_3^{-1}\} \to \{x_5, x_5^{-1}\} \Rightarrow \begin{cases} \xrightarrow{e} \begin{cases} x_3 \to x_5 \\ x_3^{-1} \to x_5^{-1} \end{cases} \\ \xrightarrow{f} \begin{cases} x_3 \to x_5^{-1} \\ x_3^{-1} \to x_5 \end{cases} \end{cases}$$

Also,

$$\bullet\ \{x_5, x_5^{-1}\} \to \{x_3, x_3^{-1}\} \Rightarrow \begin{cases} \xrightarrow{g} \begin{cases} x_5 \to x_5 \\ x_5^{-1} \to x_3^{-1} \end{cases} \\ \xrightarrow{h} \begin{cases} x_5 \to x_3^{-1} \\ x_5^{-1} \to x_3 \end{cases} \end{cases}$$

- Now, we come back to the general case of θ (on page 22):

  Let

$$\begin{aligned} \sigma &= \langle \tau_{x_1}, \ldots, \tau_{x_n} \rangle \\ &= \langle \tau_{x_1} \rangle \oplus \ldots \oplus \langle \tau_{x_n} \rangle \\ &= \langle \tau_{x_1} | \tau_{x_1}^2 \rangle \oplus \ldots \oplus \langle \tau_{x_n} | \tau_{x_n}^2 \rangle. \end{aligned}$$

  Where,

$$\begin{aligned} \tau_{x_1} &: \ x_1 \to x_1^{-1}, x_1^{-1} \to x_1 \ and\ x_j \to x_j,\ if\ j \neq 1, \tau_{x_1}^2 = 1 = (), \\ \tau_{x_2} &: \ x_2 \to x_2^{-1}, x_2^{-1} \to x_2 \ and\ x_j \to x_j,\ if\ j \neq 2, \tau_{x_2}^2 = 1 = (), \\ &\vdots \\ \tau_{x_n} &: \ x_n \to x_n^{-1}, x_n^{-1} \to x_n \ and\ x_j \to x_j,\ if\ j \neq n, \tau_{x_n}^2 = 1 = () \cdot \end{aligned}$$

(There is no need for $\tau_{x_j^{-1}}$ for $j = 1, \ldots n$, because we have that $\tau_{x_j^{-1}} = \tau_{x_j}$).

Suppose that ϕ is any isomorphism of Γ. So for each $x \in V$ and $\phi(x) \in V$ and ϕ maps x bijectively to itself. Then ϕ gives rise to 2^n automorphisms of type (1) (where $|V(\Gamma)| = n$). For each $x \in V$ we have two choices a and b,

$$x \ \begin{array}{l} \xmapsto{a} \phi(x) \\ \xmapsto{b} \phi(x)^{-1} \end{array}$$

- If $x \longmapsto \phi(x)$ then $x^{-1} \longmapsto \phi(x)^{-1}$,
- If $x \longmapsto \phi(x)^{-1}$ then $x^{-1} \longmapsto \phi(x)$, so once these choices have been made we have uniquely determined an automorphism of type (1).

Now let

$$T = \langle \text{automorphisms of type(1)} \rangle \leq Aut(G_\Gamma),$$

$\zeta = Aut(\Gamma)$ the group of automorphism of Γ (elements of which permute V).

$$\begin{aligned} I &= \langle \tau_x : \ x \in V(\Gamma) \text{ and } \tau_x(x) = x^{-1} \rangle, \\ &= \mathbb{Z}_2 \oplus \ldots \oplus \mathbb{Z}_2, \ (|V(\Gamma)| - \text{times}), \\ &\cong (\langle \tau_{x_1} \rangle \oplus \ldots \oplus \langle \tau_{x_n} \rangle). \end{aligned}$$

Any automorphism θ of type (1) permutes the sets $\{x, x^{-1}\}$ such that $x \in V$ so contracts to a graph automorphism ϕ, from which θ can be recovered as above. Now we have the following facts; for θ and ϕ

Fact 1: $\phi^{-1}\tau_x\phi = \tau_{\phi^{-1}(x)}$. That is $\phi\tau_x = \tau_{\phi(x)}\theta$.

If $\phi \in \zeta$ and $\tau_x \in I$ then for each $z \in V$ we have that,

$$\begin{aligned} \phi^{-1}\tau_x\phi(z) &= \begin{cases} \phi^{-1}\phi(z) & if \ x \neq \phi(z) \\ \phi^{-1}(\phi(z))^{-1} & if \ x = \phi(z), \end{cases} \\ &= \begin{cases} z & if \ x \neq \phi(z) \\ \phi^{-1}(\phi(z^{-1})) & if \ x = \phi(z), \end{cases} \\ &= \begin{cases} z & if \ x \neq \phi(z) \\ z^{-1} & if \ x = \phi(z), \end{cases} \\ &= \tau_{\phi^{-1}(z)}. \end{aligned}$$

Fact 2: $\tau_x\tau_y = \tau_y\tau_x$, for each $x \neq y \in V$

Fact 3: Suppose we choose option b for $x = x_1, \ldots, x_r$ and option a for all other $x \in V$. Then we will have the following fact. The resulting map of type (1) is

$$\theta = \tau_{\phi(x_1)} \ldots \tau_{\phi(x_r)}\phi = \phi\tau_{x_1} \ldots \tau_{x_r},$$

and

$$\phi\tau_x(x) \to \phi(x^{-1}) = \phi(x)^{-1}, \ \forall x \in V.$$

- From Fact 3 we have $T = \langle I, \zeta \rangle$ and moreover $T = \zeta I = I\zeta$. From Fact 1, as $\tau_{\phi^{-1}(x)} \in I$ we have $I \lhd T$.

- We show $\zeta \cap I = \{id\}$. Suppose $\alpha \in \zeta \cap I$. Then $\alpha(x) \in V$, $\forall x \in V$, as $\alpha \in \zeta$. Also $\alpha(x) = x$ or x^{-1}, as $\alpha \in I$. Hence (as $x^{-1} \notin V$) $\alpha(x) = x$, $\forall x \in V$. Therefore $\alpha = id$ and so $\zeta \cap I = \{id\}$. Therefore, $T = \zeta \rtimes I$.

Therefore, given a presentation $\langle Gens(\zeta) \cup I \mid Rels(\zeta) \rangle$ for ζ, a presentation for T is,

$T = \langle Gens(\zeta) \cup Gens(I) \mid Rels(\zeta) \cup \{\tau_v^2 : v \in V(\Gamma)\} \cup \{[\tau_v, \tau_u] : u, v \in V(\Gamma), u \neq v\} \cup \{\phi^{-1}\tau_v\phi = \tau_{\phi^{-1}(v)}$ for each $\phi \in Gens(G)$ and $\tau_v \in Gens(I)\}\rangle$.

■ **Day's relation** $R'5$ is:

$(R'5)$ $(A - a + a^{-1}, b)(A, a) = (A - b + b^{-1}, a)\sigma_{a,b}$

for $(A, a) \in \Omega$ and $b \in A$ with $b^{-1} \notin A, b \neq a$, and $b \sim a$, where $\sigma_{a,b}$ is the type (1) Whitehead automorphism with $\sigma_{a,b}(a) = b^{-1}, \sigma_{a,b}(b) = a$ and which fixes the other generations.

$(R'5)$ involves type (1) automorphisms $\sigma_{a,b}$ which we are writing as $\sigma_{a,b} = \tau_b(a, b)$ where $(a, b) \in Aut^{\Gamma}(G)$ is the graphic automorphism induced by the automorphism (a, b) of Γ sending a to b and b to a. Hence, $(R'5)$ becomes $(R5)$ of Definition 2.5.1.

■ **Day's relation** $R'6$ is:

$(R'6)$ $\sigma(A, a)\sigma^{-1} = (\sigma(A), \sigma(a))$

for $(A, a) \in \Omega$ of type (2) and $\sigma \in \Omega$ of type (1).

We have generators of type (1) of the form

I : that is τ_x for $x \in V$, and

ζ : that is graph isomorphisms (permutations of V). However not all type (1) elements appear in our generating set. So we replace the above relation $(R'6)$ with $(R6)$ of Definition 2.5.1

Note that $(R'6)$ follows from $(R6)$, as we may write any σ of type (1) as

$\sigma = \phi\, \tau_{x_1} \dots \tau_{x_r}$ for suitable $\phi \in \zeta$ and $\tau_{x_i} \in I$ (from Fact 3) and then

$$
\begin{aligned}
\sigma(A,a)\sigma^{-1} &= \phi\tau_{x_1}\dots\tau_{x_r}(A,a)\tau_{x_r}^{-1}\dots\tau_{x_1}^{-1}\phi^{-1} \\
&= \phi\tau_{x_1}\dots\tau_{x_{r-1}}(\tau_{x_r}(A),\tau_{x_r}(a))\tau_{x_{r-1}}^{-1}\dots\tau_{x_1}^{-1}\phi^{-1} \\
&= \phi\tau_{x_1}\dots\tau_{x_{r-2}}(\tau_{x_{r-1}}\tau_{x_r}(A),\tau_{x_{r-1}}\tau_{x_r}(a))\tau_{x_{r-2}}^{-1}\dots\tau_{x_2}^{-1}\phi^{-2} \\
&= \phi(\tau_{x_1}\dots\tau_{x_r}(A),\tau_{x_1}\dots\tau_{x_r}(a))\phi^{-1} \\
&= (\phi\tau_{x_1}\dots\tau_{x_r}(A),\phi\tau_{x_1}\dots\tau_{x_r}(a)) \\
&= (\sigma(A),\sigma(a)).
\end{aligned}
$$

2.6 Peak reduction

Peak reduction is a technique in the study of $Aut(F)$ that is a key ingredient in the solution of several important problems. J.H.C. Whitehead invented the technique in the 1930's in [72] to provide an algorithm that takes in two conjugacy classes (or more generally, $k-$tuples of conjugacy classes) from F and determines whether there is an automorphism in $Aut(F)$ that carries one to the other.

Definition 2.6.1. For W a k-tuple of conjugacy classes in G_Γ, we say that a string $\alpha_m\dots\alpha_1$ of elements of $Aut(G_\Gamma)$ is **peak-reduced** with respect to W if for each $i=1,\dots,m-1$, we do not have both

$$|(\alpha_{i+1}\dots\alpha_1)\cdot W| \leq |(\alpha_i\dots\alpha_1)\cdot W|$$

and

$$|(\alpha_i\dots\alpha_1)\cdot W| \geq |(\alpha_{i-1}\dots\alpha_1)\cdot W|$$

unless all three lengths are equal. It is equivalent to that, for some $k_1 \leq k$, the length of $\alpha_k\dots\alpha_1\cdot W$ decreases with k until $k=k_1$, remains constant until $k=k_2$, and then increases with k until $k=m$.

We see that $Aut(G_\Gamma)$ **has peak reduction** with respect to Ω if for any $\alpha \in Aut(G_\Gamma)$ and any tuple of conjugacy classes W, we can find $\alpha_m,\dots,\alpha_1 \in \Omega$ such that $\alpha=\alpha_m,\dots,\alpha_1$ and the string of elements $\alpha_m,\dots,\alpha_1$ is peak-reduced with respect to W.

Theorem 2.6.2. *[24] The finite generating set $\Omega_\ell \cup \Omega_s$ for $Aut(G_\Gamma)$ has the following properties:*

1. *each $\alpha \in Aut(G_\Gamma)$ can be written as $\alpha=\beta\gamma$ for some $\beta \in \langle\Omega_s\rangle$ and some $\gamma \in \langle\Omega_\ell\rangle$,*

2. *the usual representation $Aut(G_\Gamma) \to Aut(H_1(G_\Gamma))$ to the automorphism group of the abelianization $H_1(G_\Gamma)$, (where $H_1(G_\Gamma) = G_\Gamma/[G_\Gamma, G_\Gamma] = (G_\Gamma)_{ab}$ of G_Γ,) restricts to an embedding $\langle \Omega_s \rangle \hookrightarrow Aut\ H_1(G_\Gamma)$; and*

3. *the subgroup $\langle \Omega_\ell \rangle$ has peak reduction by elements of Ω_ℓ with respect to any $k-$tuple W of conjugacy classes in G_Γ.*

Theorem 2.6.3. *[24] The peak-reduction theorem for a free group F_n states that there is a finite generating set Ω for $Aut(F_n)$ (called the Whitehead automorphisms, see [72]) such that $Aut(F_n)$ has peak reduction with respect to any $k-$tuple of conjugacy classes W in F_n by element of Ω. We will give an example to explain this theorem.*

Example 2.6.0.2
For a free group $F_n = F(x, y)$, pick any $\alpha \in Aut(F_n)$ and any k-tuple $(w_1, \ldots, w_k)$ where w_k is a representative of a conjugacy classes of F_n.
Let $W = (x, xy, xy^{-1})$, $|W| = 5$. Suppose that,

$$\alpha : \begin{cases} x \longmapsto y^{-1}xy \\ y \longmapsto x^2 y \end{cases}$$

we can factorise α into Whitehead automorphism, according to Theorem 2.6.2, so that

$$\alpha = \alpha_m \ldots \alpha_1,$$

$$|W| \leq |\alpha_1 W| \leq |(\alpha_2\alpha_1)W| \leq \ldots \leq |(\alpha_m \ldots \alpha_1)W| = |\alpha W|.$$

Now, we can factor α in the following way, $\alpha = \alpha_1\ \alpha_2\ \alpha_3$, where

$$\alpha_2 = \alpha_3 : \begin{cases} x \longmapsto x \\ y \longmapsto yx \end{cases}$$

so written a Whitehead automorphism,

$$\alpha_2 = (\{x, y\}, x),$$

and

$$\alpha_1 : \begin{cases} x \longmapsto y^{-1}xy \\ y \longmapsto y \end{cases}$$

so written as whitehead automorphism,

$$\alpha_1 = (\{x, x^{-1}, y\}, y)$$

We will check that $\alpha = \alpha_1\ \alpha_2\ \alpha_3$:

$\alpha_1\alpha_2\alpha_3(x) = \alpha_1\alpha_2(x) = \alpha_1(x) = y^{-1}xy$

$\alpha_1\alpha_2\alpha_3(y) = \alpha_1\alpha_2(yx) = \alpha_1(yx^2) = yy^{-1}x^2y = x^2y.$

Hence we get that,

$$\alpha = \alpha_1\alpha_2\alpha_3 : \begin{cases} x \longmapsto y^{-1}xy \\ y \longmapsto x^2y \end{cases}$$

$W = (x, xy, xy^{-1}),$

$\alpha_3.W = (x, xyx, y^{-1}), |\alpha_3.W| = 5.$

$\alpha_2\alpha_3.W = (x, xyx^2, x^{-1}y^{-1}), |\alpha_2\alpha_3.W| = 7.$

$\alpha_1\alpha_2\alpha_3.W = (y^{-1}xy, y^{-1}xyx^2y, y^{-1}x^{-1}) \sim (x, xyx^2, y^{-1}x^{-1}), |\alpha_1\alpha_2\alpha_3.W| = 7.$

As we shown above that $\alpha = \alpha_1\alpha_2\alpha_3$, then it is obvious that $\alpha.W = \alpha_1\alpha_2\alpha_3.W$. Hence, the sequence W, $\alpha_1.W$, $\alpha_2\alpha_1.W$, $\alpha_3\alpha_2\alpha_1.W$ has no peak.

Lemma 2.6.4. *[24] Let X be a k-tuple of conjugacy classes whose elements are all the conjugacy classes in G_Γ of length 2, each appearing once. If $(A, a) \in \Omega_\ell$ and $|(A, a) \cdot X| \leq |V|$, then (A, a) is trivial or is the conjugation $(L\backslash\{a^{-1}\}, a)$.*

Lemma 2.6.5. *[24] Suppose α, $\beta \in \Omega_\ell$ and $[W]$ is a k-tuple of conjugacy classes of G_Γ. If $\beta\alpha^{-1}$ forms a peak with respect to $[W]$, there exist $\delta_1, \ldots, \delta_k\Omega_\ell$ such that $\beta\alpha^{-1} = \delta_k \ldots \delta_1$ and for each i, $1 \leq i < k$, we have:*

$$|(\delta_i \ldots \delta_1) \cdot [W]| < |\alpha^{-1} \cdot [W]|$$

*A factorization of $\beta\alpha^{-1}$ is **peak-lowering** if it satisfies the conclusions of the Lemma, so Lemma 2.6.5 states that every peak has a peak-lowering factorization.*

2.7 GAP Presentation for the $Aut(G_\Gamma)$

First we will give a small example to find a finite presentation of automorphism groups of partially commutative group $Aut(G_\Gamma)$.

Example 2.7.0.3

Let $\Gamma = (V, E)$ be the following graph:

Then $V = \{x_1, x_2\}$ and $E = \emptyset$. It is a free group with two generators $\{x_1, x_1\}$. Thus,

(1) $st(x_1) = \{x_1\}$,

$\ell k(x_1) = \phi$,

$Comps1 = \Gamma \backslash st(x_1) = \{x_2\} =$ connected components of $\Gamma \backslash st(x_1)$.

(2) $st(x_2) = \{x_2\}$,

$\ell k(x_1) = \phi$,

$Comps2 = \Gamma \backslash st(x_2) = \{x_1\} =$ connected components of $\Gamma \backslash st(x_2)$.

(3) A list $Y(x)$, for each x in V of these vertices y in V such that y less then x, and we call this list by Y, so

$$Y = \{\{x_2\}, \{x_1\}\}.$$

(4) Now, we will find the generators of type (2) of the Whitehead automorphisms of the subgraph $E_1 = \Gamma \backslash st(x_1)$:

$L_1 = Comps1 \cup \{\{x_2\}, \{x_2^{-1}\}\}$

$= \{x_2, x_2^{-1}\} \cup \{\{x_2\}, \{x_2^{-1}\}\}$

$= \{\{x_2\}, \{x_2^{-1}\}, \{x_2, x_2^{-1}\}\}.$

Hence, the whitehead automorphisms of the subgraph $E_1 = \Gamma \backslash st(x_1)$ are:

$C_1 = \{\{\{x_2, x_1\}, x_1\}, \{\{x_2, x_1^{-1}\}, x_1^{-1}\}, \{\{x_2^{-1}, x_1\}, x_1\},$
$\{\{x_2^{-1}, x_1^{-1}\}, x_1^{-1}\}, \{\{x_2, x_2^{-1}, x_1\}, x_1\}, \{\{x_2, x_2^{-1}, x_1^{-1}\}, x_1^{-1}\}\}.$

(5) Now, we will find the generators of type (2) of whitehead automorphisms of the subgraph $E_2 = \Gamma \backslash st(x_2)$:

$L_2 = \ Comps2 \ \cup \{\{x_1\}, \{x_1^{-1}\}\}$

$= \{\{x_1, x_1^{-1}\} \cup \{\{x_1\}, \{x_1^{-1}\}\}$

$= \{\{x_1\}, \{x_1^{-1}\}, \{x_1, x_1^{-1}\}\}.$

Hence, the whitehead automorphisms of the subgraph $E_2 = \Gamma \backslash st(x_2)$ are:

$$C_2 = \{\{\{x_1, x_2\}, x_2\}, \{\{x_1, x_2^{-1}\}, x_2^{-1}\}, \{\{x_1^{-1}, x_2\}, x_2\}, \{\{x_1^{-1}, x_2^{-1}\}, x_2^{-1}\}, \{\{x_1, x_1^{-1}, x_2\}, x_2\}, \{\{x_1, x_1^{-1}, x_2^{-1}\}, x_2^{-1}\}\}.$$

- Therefore, the generators of type (2) whitehead automorphisms of the graph Γ are the following set A:

  $A = C_1 \cup C_2,$

  $$A = \{A_1 = \{\{x_2, x_1\}, x_1\},\ A_2 = \{\{x_2, x_1^{-1}\}, x_1^{-1}\},\ A_3 = \{\{x_2^{-1}, x_1\}, x_1\},\ A_4 = \{\{x_2^{-1}, x_1^{-1}\}, x_1^{-1}\},\ A_5 = \{\{x_2, x_2^{-1}, x_1\}, x_1\},\ A_6 = \{\{x_2, x_2^{-1}, x_1^{-1}\}x_1^{-1}\},\ A_7 = \{\{x_1, x_2\}, x_2\},\ A_8 = \{\{x_1, x_2^{-1}\}, x_2^{-1}\}, A_9 = \{\{x_1^{-1}, x_2\}, x_2\}, A_{10} = \{\{x_1^{-1}, x_2^{-1}\}, x_2^{-1}\},\ A_{11} = \{\{x_1, x_1^{-1}, x_2\}, x_2\},\ A_{12} = \{\{x_1, x_1^{-1}, x_2^{-1}\}, x_2^{-1}\}\}.$$

- Now, we will find type (1) of generators of the whitehead automorphisms of the graph Γ:

  (1) The graph isomorphisms of Γ are that,
  $\zeta = \{F_1 = (1, 2),\ \textit{identity}\ \}$ (*permutation of vertices*).
  $I = \langle g_x : x \in V(\Gamma) \text{ and } g_x(x) = x^{-1}\rangle,$
  $= \{g_{x_1}(x_1) = x_1^{-1}, g_{x_2}(x_2) = x_2^{-1}\}$
  Thus, the generators of type (1) of the whitehead automorphisms are the following set T:
  $T = \zeta \cup I = \{F_1, g_{x_1}, g_{x_2}\}$

- Therefore, the generators set $Gens$ of the automorphism groups of PCG of the graph Γ is that,

  $Gens = A \cup T = \{A_1, A_2, A_3, A_4, A_5, A_6, A_7, A_8, A_9, A_{10}, A_{11}, A_{12}, F_1, g_{x_1}, g_{x_2}\}.$

- The relations (Rels) between these generators as follows:

(1) $R_1 = \{A_1 * A_2, A_3 * A_4, A_5 * A_6, A_7 * A_8, A_9 * A_{10}, A_{11} * A_{12}\}.$

(2) $R_2 = \{A_1 * A_3 * A_5^{-1}, A_2 * A_4 * A_6^{-1}, A_3 * A_1 * A_5^{-1}, A_4 * A_2 * A_6^{-1}, A_7 * A_9 * A_{11}^{-1}, A_8 * A_{10} * A_{12}^{-1}, A_9 * A_7 * A_{11}^{-1}, A_{10} * A_8 * A_{12}^{-1}\}.$

(3) $R_3 = \emptyset$.

(4) $R_4 = \emptyset$.

(5) $R_5 = \{A_9 * A_1 * g_{x_2} * A_3^{-1}, A_7 * A_2 * g_{x_2} * A_4^{-1}, A_{10} * A_3 * g_{x_2} * A_1^{-1}, A_8 * A_4 * g_{x_2} * A_2^{-1}, A_3 * A_7 * g_{x_1} * A_9^{-1}, A_1 * A_8 * g_{x_1} * A_{10}^{-1}, A_4 * A_9 * g_{x_1} * A_7^{-1}, A_2 * A_{10} * g_{x_1} * A_8^{-1}\}$.

(6) $R_6 + R_7 = \{g_{x_1}^2, g_{x_2}^2, g_{x_1}^{-1} * g_{x_2}^{-1} * g_{x_1} * g_{x_2}, g_{x_2}^{-1} * g_{x_1}^{-1} * g_{x_2} * g_{x_1}, F_1^{-1} * g_{x_1} * F_1 * g_{x_2}, F_1^{-1} * g_{x_2} * F_1 * g_{x_1}\}$.

(7) $R_8 = \{A_1 * A_4^{-1} * A_5^{-1}, A_2 * A_3^{-1} * A_6^{-1}, A_3 * A_2^{-1} * A_5^{-1}, A_4 * A_1^{-1} * A_6^{-1}, A_5 * Id * A_5^{-1}, A_6 * Id * A_6^{-1}, A_7 * A_{10}^{-1} * A_{11}^{-1}, A_8 * A_9^{-1} * A_{12}^{-1}, A_9 * A_8^{-1} * A_{11}^{-1}, A_{10} * A_7^{-1} * A_{12}^{-1}, A_{11} * Id * A_{11}^{-1}, A_{12} * Id * A_{12}^{-1}\}$.

(8) $R_9 = \emptyset$.

(9) $R_{10} = \{A_1 * A_{11} * A_1^{-1} * A_{11}^{-1} * A_5^{-1}, A_2 * A_{11} * A_2^{-1} * A_{11}^{-1} * A_6^{-1}, A_3 * A_{12} * A_3^{-1} * A_{12}^{-1} * A_5^{-1}, A_4 * A_{12} * A_4^{-1} * A_{12}^{-1} * A_6^{-1}, A_7 * A_5 * A_7^{-1} * A_5^{-1} * A_{11}^{-1}, A_8 * A_5 * A_8^{-1} * A_5^{-1} * A_{12}^{-1}, A_9 * A_6 * A_9^{-1} * A_6^{-1} * A_{11}^{-1}, A_{10} * A_6 * A_{10}^{-1} * A_6^{-1} * A_{12}^{-1}\}$.

(10) We have one relation for the automorphisms of graph ($F_1 = (1, 2)$), which is F_1^2.

Therefore, the relations set $Rels$ among the generators $Gens$ is that,

$$Rels = R1 \cup R2 \cup R3 \cup R4 \cup R5 \cup R6 \cup R7 \cup R8 \cup R9 \cup R10 \cup \{F_1^2\}.$$

Hence, the finite presentation for automorphism groups of G_Γ is that,

$$Aut(G_\Gamma) = \langle Gens | Rels \rangle .$$

We have developed $AutParCommGrp$ (**Finite Presentations of Automorphism Groups of Partially Commutative Groups and Their Subgroups**) a package using the GAP system for computation of a finite presentation for the automorphism group of a partially commutative group $Aut(G_\Gamma)$ and their subgroups $Conj(G_\Gamma)$ and $Conj_V$ which are described in Chapters 3 and 4 respectively see [1].

This package *AutParCommGrp* mainly installs new method to provide a finite presentation for the groups $Aut(G_\Gamma)$, $Conj(G_\Gamma)$ and $Conj_V$. The process involves the computation of other objects/values which may be useful in their own right. These are defined for a graph $\Gamma = (V, E)$ on n vertices, with vertices V and edge set E, where E is a list of pairs of vertices. They are the star $St(v)$ and the link $Lk(v)$ for each vertex v of V, the list $Y(v)$ of those vertices u in V such that u is less than v, the subgraphs $\Gamma\backslash St(v)$, the connected components of a graph, the unions of the connected components of a graph, the equivalence classes for each vertex v of V under equivalence relation $\sim$ ($St(v)$ and $Lk(v)$ are used to define a partial order on V which induces equivalence relation $\sim$). In addition, it can be used to apply Tietze transformations to simplify the presentation of the groups it finds by using a *GAP* function.

To write an algorithm to produce a finite presentation for the automorphism group of a partially commutative group $Aut(G_\Gamma)$ first we find Ω the Whitehead generators set of this group based on Laurence's generators as defined in Section 2.4 and then find the set of relations R as defined in Definition 2.5.1.

The input of the main function `FinitePresentationOfAutParCommGrp`(V, E) that provides finite presentation for the group $Aut(G_\Gamma)$ is a simple graph $\Gamma = (V, E)$. A graph with vertex set V of size n always has vertices $\{1, \dots, n\}$ and E is a list of pairs of elements of V. For example if Γ is a simple graph with vertex set $V = \{x_1, x_2, x_3\}$ and edge set $E = \{[x_1, x_2], [x_1, x_3], [x_2, x_3]\}$ (where $[x, y]$ denotes an edge joining x to y) then Γ will be represented as $([1, 2, 3], [[1, 2], [1, 3], [2, 3]])$. The output of `FinitePresentationOfAutParCommGrp` consists of two sets *gens* and *rels*, where *gens* is the list of the Whitehead generators of $Aut(G_\Gamma)$ defined in Section 2.4 and *rels* is the list of the relators R.

This section describes the functions from the package *AutParCommGrp* which we have written for computing a finite presentation for $Aut(G_\Gamma)$ as follows.

2.7.1 `IsSimpleGraph` Function

A simple graph, is an unweighted, undirected graph containing no graph loops or multiple edges. A simple graph may be either connected or disconnected. `IsSimple-Graph` tests whether the graph Γ fulfills these conditions. The input of the function `IsSimpleGraph`(V, E) is a graph $\Gamma = (V, E)$, where V and E represents the list of vertices and the list of edges respectively. The algorithm carries out the following instructions:

IsSimpleGraph(V, E)

if V is empty list
    then return error message
if V or E are not lists
    then return error message
if Γ has loops
    then return error message
if $E \not\subset V \times V$
    then return error message
$M \leftarrow$ Size(E)
for i **in** $\{1, \ldots, M\}$
    do if E has multiple edges
        then return error message
return *true*

2.7.2 `StarLinkDominateOfVertex` Function

The input of the function `StarLinkDominateOfVertex`(V, E) is a simple graph $\Gamma = (V, E)$. It computes the star $St(v)$ and the link $Lk(v)$ and concatenates them in two separate lists St and Lk respectively. Also it calculates a list $Y(v)$, for each vertex v in V of those vertices u in V such that u is less than v, and we call the list of all such $Y(v)$, YY. In addition, it calculates sV, the size of the list of vertices V and M, the size of the list of edges E. The algorithm carries out the following instructions:

StarLinkDominateOfVertex(V, E)

for v **in** $V(\Gamma)$
    do for e **in** $E(\Gamma)$
        do if e is adjacent v
            then Add "end point" of e to $Lk[v]$
$St[v] = Lk[v] \cup \{v\}$
for v **in** $St[v]$
    do for u **in** $Lk[v]$
        do if $St[u] \subseteq Lk[v]$
            then Add u to $Y(v)$

Append $Y(v)$ to YY
$L \leftarrow V \cup (-V)$
return $[St, Lk, YY, sV, M, L, sL]$

2.7.3 `DeleteVerticesFromGraph` Function

The input of the function `DeleteVerticesFromGraph`(St, V, E) is the list of stars St, the list of vertices V, and the list of edges E. It computes graphs $\Gamma \backslash St(v)$, for all v in V, with NV the list of all lists of vertices of $\Gamma \backslash St(v)$ and NE the list of all lists of edges of $\Gamma \backslash St(v)$. The algorithm carries out the following instructions:

DELETEVERTICES(St, V, E)
$sV \leftarrow$ SIZE(V)
$M \leftarrow$ SIZE(E)
for v **in** $V(\Gamma)$
  do for e **in** $E(\Gamma)$
    do if e is not adjacent to $u \in St(v)$
      then ADD e to $H1$
        ADD vertices incident to edges in $H1$ to $H2$
Append $H1$ to NE and $H2$ to NV
return $[NV, NE, sNV, sNE]$

2.7.4 `ConnectedComponentsOfGraph` Function

The input of the function `ConnectedComponentsOfGraph`$(G1, G2)$ is the list of vertices $G1$ and the list of edges $G2$ of a graph B. It computes the list of connected components *AllComps* of the graph B and its size *sAllComps*. Also it computes the list of non-isolated connected components *NonIsolatedComps* and the list of isolated connected components *IsolatedComps* of the graph B. In addition it computes the lists D and F the list of vertices of *NonIsolatedComps* and *IsolatedComps* respectively. The algorithm carries out the following instructions:

CONNECTEDCOMPONENTSOFGRAPH$(G1, G2)$
$M \leftarrow$ LENGTH$(G2)$ $\triangleright$ $G2$ is edge list of a simple graph B.
for i **in** $\{1, \ldots, M\}$
  do $D \leftarrow$ COMPUTEVERTEXLISTOFNON-ISOLATED COMPONENTS(B)

```
sD ← Size(D)
for i in {1, ..., M}
      do W ← ComputeAdjacencyMatrix(B)
for i in {1, ..., sD}
      do if color[s] = 0    ▷ color is a list of size sD with entries the
                            ▷ numbers of non-isolated components.
            then count ← count + 1
                            ▷ count is a specific number representing
                            ▷ the vertices of each component.
                  color[i] ← count
                  NonIsolatedComps ← DfsVisit(i, W, sD, count, color)
for k in {1, ..., count}
      do for i in {1, ..., sD}
            do Add non-isolated component with its inverse to new list P
                  Append P to the list NonIsolatedComps
F ← Difference(G1, D)    ▷ F is vertices of isolated components
sF ← Size(F)
for i in {1, ..., sF}
      do IsolatedComps ← ComputeIsolated components(B)
AllComps ← ComputeAllcomponents(B)
return [AllComps, sAllComps, NonIsolatedComps, D, IsolatedComps, F]
```

2.7.5 `DFSVisit` Function

The input to `DFSVisit`$(i, W, sD, count, color)$ is a vertex i of graph B, the weight matrix W of B, the size sD of the vertex list of the graph B, an index *count*, corresponding to a connected component of B and a list *color*. The s^{th} item of *color* is the (number of the) component of B to which the s^{th} vertex of B belongs (or is zero if s has not yet been processed). The function implements the depth search algorithm to construct the connected components (having more than one vertex) of the graph B. On input a vertex i with *count* $j > 0$, the algorithm checks to see if there is a vertex s, joined to i by an edge, with $color[s] = 0$. On finding such an s the algorithm sets $color[s] = count$ and calls itself with input $(s, W, sV, count, color)$.

DfsVisit($i, W, sD, count, color$)

for s **in** $\{1, \dots, sD\}$
  do if $color[s] = 0$ and $W[i][s] = 1$
    then $color[s] = count$
      DfsVisit($s, W, sD, count, color$)
End

2.7.6 `WhiteheadAutomorphismsOfSecondType` Function

The inputs of the function `WhiteheadAutomorphismsOfSecondType`(NV, NE, St, YY) are the lists of vertices NV and the list of edges NE of the subgraphs $\Gamma \backslash St(v) = (NV(v), NE(v))$ for all v in V, the list of stars $St(v)$, and the list YY defined in `StarLinkDominateOfVertex` above. It computes the list A of type (2) Whitehead automorphisms which forms the first part of the set of generators of $Aut(G_\Gamma)$. Also it computes a list T of names of elements of A (the i^{th} element of T is the name of the i^{th} element of A). The algorithm carries out the following instructions:

WhiteheadAutomorphismsOfSecondType(NV, NE, St, YY)

$sNE \leftarrow$ Size(NE)
for h **in** $\{1, \dots, sNE\}$ $\quad \triangleright\ h \in V$
  do $G \leftarrow$ NE(h)
    $R3 \leftarrow$ ConnectedComponentsOfGraph($G1, G2$)
    $Comps \leftarrow$ R3(3) $\quad \triangleright$ $Comps$ is non-isolated components
    $sComps \leftarrow$ Size($Comps$)
    $D \leftarrow$ R3(4)
    $sD \leftarrow$ Size(D)
    $S \leftarrow$ St(h)
    $DYY \leftarrow YY(v) \cup YY(V)^{-1}$
    $sDYY \leftarrow$ Size(DYY)
    $Ls \leftarrow [[]]$
    for t **in** $\{1, \dots, sDYY\}$
      do $xn \leftarrow$ DYY(t)
        $Ls \leftarrow$ UnionElement(Ls, xn, S)
    $sAQ \leftarrow$ Size(Ls)
    for i **in** $\{1, \dots, sAQ\}$
      do Add the non empty elements of Ls to new list $L3$

$sMV \leftarrow \mathrm{MV}(h)$
for j **in** $\{1, \ldots, sMV\}$
do if $\mathrm{MV}(h)(j) \notin D$ and $sMV \neq 1$ and $MV(h) \neq YY(h)$
then ADD $[MV(h)(j)]$ and $[MV(h)(j)]$ to $Ls3$
ADD $[MV(h)(j), MV(h)(j)^{-1}]$ to $Ls3$
for each list W **in** $L3$
do ADD $W \cup \{h\}$ to new list $L4$
for X **in** $L4$
do ADD $(X \backslash \{h\}) \cup \{h^{-1}\}$ to new list $L5$
$AA \leftarrow$ CONCATENATION$(L4, L5)$
ADD the non empty elements of AA to new list A
$sA \leftarrow$ SIZE(A)
for i **in** $\{1, \ldots, sA\}$
do ADD A_i the name of the i^{th} element of A to new list T
return $[A, T, sA]$

2.7.7 `WhiteheadAutomorphismsOfFirstType` Function

The input of the function `WhiteheadAutomorphismsOfFirstType`(E, sV, sA, T) is the list of edges E, the size of the list of vertices sV, the size of the list A of type (2) Whitehead automorphism of Γ, defined above, and the list T, also defined earlier. It computes the list *Gens* of the type (1) Whitehead automorphisms which forms the second part of the set of generators of the automorphism group of G_Γ, and then computes the list of the generators *gens* of $Aut(G_\Gamma)$ with its size *sgens*. The subgroup $Aut^\Gamma(G_\Gamma)$ of $Aut(G_\Gamma)$ consists of graph automorphism: that is, elements $\pi \in Aut(G_\Gamma)$ such that $\pi|_\Gamma$ is a graph automorphism. The algorithm carries out the following instructions:

WHITEHEADAUTOMORPHISMSOFFIRSTTYPE(E, sV, sA, T)
$Gr \leftarrow$ GRAPHAUTOMORPHISMGROUP(E)
$HH \leftarrow$ ASGROUP(Gr)
$GHH \leftarrow$ GENERATORSOFGROUP(HH)
$KK \leftarrow$ ISOMORPHISMFPGROUPBYGENERATORS(HH, GHH)
$HHH \leftarrow$ IMAGE(KK)
$rels2 \leftarrow$ RELATORSOFFPGROUP(HHH)

```
srels2 ← RelatorsOfFpGroup(rels2)
F ← GeneratorsOfGroup(HHH)
SF ← Size(F)
for each R in rels2
      do zz ← ExtRepOfObj(R)
            Add zz to new list Rels1
sRels1 ← Size(Rels1)
for i in {1, ..., sF}
      do Add f_i the name of the i^th element of F to new list Gens3
relvalofF ← GeneratorsOfGroup(HH)
srelvalofF ← Size(relvalofF)
for v in V
      do I2 ← ComputeInversionAutomorphismOfEachVertex
            Add I2 to new list I1
for A in {1, ..., I1}
      do Add A_i the name of the i^th element of I1 to new list Gens2
sGens2 ← Size(Gens2)
Gens ← Concatenation(Gens2, Gens3)
sGens ← Size(Gens)
for i in {1, ..., sGens}
      do Add Gens(i) to new list gens
genss ← Concatenation(T, Gens2)
gens ← Concatenation(T, Gens)
sgenss ← Size(genss)
sgens ← Size(gens)
return [gens, sgens, sgenss, Gens3, relvalofF, srelvalofF, Rels1, sRels1, sGens2]
```

Remark 2.7.1. We have an important notes before we start describe the functions that compute the set of relations as follows:

(1) The relators are represented using sequences of the form $R = [p, \epsilon_1 n_1, \ldots, \epsilon_k n_k]$, where p,ϵ_i,n_i are integers, $\epsilon_i = \pm 1$, $0 \leq p \leq 2$ and $1 \leq n_i$. If $p = 0$ *or* 1 then the sequence R corresponds to the word $W_R = ((A_{n_1}^{\epsilon_1})^{p+1} * \ldots * (A_{n_k}^{\epsilon_k})^{p+1})$, and R is called the index of W_R. For example relators of type $(R1)$ have form $(A, a) * (A - a + a^{-1}, a^{-1}) = 1$ and have indices of form $[0, idx1, idx2]$ where

$idx1 = (A, a)$ and $idx2 = (A - a + a^{-1}, a^{-1})$. Sequences with $p = 1$ occur only in Section 2.7.8 below.

(2) If $p = 2$ then the sequence R corresponds to a relator of type $(R5)$. These have the form $W_R = 1$ where $W_R = (A - a + a^{-1}, b) * (A, a) * \sigma_{a,b} * (A - b + b^{-1}, a)^{-1}$, and the corresponding sequence is $[2, idx1, idx2, -idx3, idx4, a, b, a]$ where,

$idx1 = (A - a + a^{-1}, b)$,

$idx2 = (A, a)$,

$idx3 = (A - b + b^{-1}, a)^{-1}$. In this case R is called the index of W_R.

(3) One type of graph isomorphisms of Γ is an **inversion**, $g_x : x \in V(\Gamma)$ given by $g_{x(x)} = x^{-1}$ and $g_{y(y)} = y$ for each $y \in V(\Gamma) \setminus \{x\}$. All inversions are type (1) Whitehead automorphisms. The subgroup $\langle g_x : x \in V(\Gamma) \rangle$ is denoted I. The inversions satisfy the relations of the form:

$$R11 = \{g_x^2 = 1 : x \in V(\Gamma)\}$$

2.7.8 `RelationsOfGraphAutomorphisms` Function

The inputs of the function `RelationsOfGraphAutomorphisms`$(sA, sgenss, relvalo-fF, sV, sGens2)$ are the size sA of the list A of definition of the second type of generator, the size of the list $genss$ defined above which is called $sgenss$, the list of generators of the graph automorphism $relvalofF$ from above, sV and $sGens2$ of lists V and $Gens2$. Compute the row matrix of indices $Rels$ of the generators which forms the relations of this type, that related to the graph automorphism with its size $sRels$. The algorithm carries out the following instructions:

RelationsOfGraphAutomorphisms$(sA, sgenss, relvalofF, sV, sGens2)$

for i **in** $\{sA + 1, \ldots, sgenss\}$
  do Add $[1, i]$ to new list $Rels$ $\triangleright$ 1 means the generators of power two
for i **in** $\{sA + 1, \ldots, sgenss\}$
  do for j **in** $\{sA + 1, \ldots, sgenss\}$
    do if $i \neq j$
      then Add $[0, -i, -j, i, j]$ to the list $Rels$
        $\triangleright$ 0 means generators here of power one
$srelvalofF \leftarrow$ Size$(relvalofF)$

```
for i in {1, ..., srelvalofF}
    do d ← RELVALOFF([i])
        F1 ← d^{-1}
        ADD F1 to new list FF
for i in {1, ..., srelvalofF}
    do for j in {1, ..., sV}
        do PP ← ONPOINTS(j, FF[i])
            idx1 ← i + sA + sGens2
            idx2 ← sA + j
            idx3 ← sA + PP
            ADD [0, -idx1, idx2, idx1, idx3] to the list Rels

sRels ← Rels
return [Rels, sRels]
```

2.7.9 `APCGRelationR1` Function

The inputs of the function `APCGRelationR1`($sV, A, T, Rels$) are the size of the list of vertices sV, the list A defined earlier, the list of generators T from Section 2.7.6, and the list of row matrices of indices of the generators $Rels$. It computes the list of indices $[0, idx1, idx2]$ of relators of type $(R1)$ of Definition 2.5.1 and adds them to the list $Rels$. We can replace $Rels$ by empty list if we want just the list of row matrices of indices of $(R1)$. In addition it calculates the size of the list $Rels$. It returns $[Rels, sRels]$.

2.7.10 `APCGRelationR2` Function

The inputs of the function `APCGRelationR2`($A, T, Rels, St$) are the list A is defined earlier, list of the generators T of $\mathrm{Aut}(\mathrm{G}_\Gamma)$ from Section 2.7.6, the list of row matrix of the indices of the generators $Rels$, and the list of stars St. It computes the list of indices of the generators $[0, idx1, idx2, -idx3]$ of relators of type $(R2)$ of Definition 2.5.1 and adds them to the list $Rels$. We can replace $Rels$ by empty list if we want just the list of row matrices of indices of $(R2)$. In addition it calculates the size of the list $Rels$. It returns $[Rels, sRels]$.

2.7.11 `APCGRelationR3` Function

The inputs of the function **`APCGRelationR3`**$(A, T, Lk, Rels)$ are the list A is defined earlier, the list of the generators T of $\mathrm{Aut}(\mathrm{G}_\Gamma)$ from Section 2.7.6, the list of links Lk, and the list of row matrix of the indices of the generators $Rels$. It computes the list of the indices $[0, idx1, idx2, -idx1, -idx2]$ of relators of type $(R3)$ of Definition 2.5.1 and $(R3a)$ and adds them to the list $Rels$. We can replace $Rels$ by empty list if we want just the list of row matrices of indices of $(R3)$. In addition it calculates the size of the list $Rels$. It returns $[Rels, sRels]$.

2.7.12 `APCGRelationR4` Function

The inputs of the function **`APCGRelationR4`**$(A, T, Lk, Rels)$ are the list A is defined earlier, the list of the generators T of $\mathrm{Aut}(\mathrm{G}_\Gamma)$ from Section 2.7.6, the list of links Lk, and the list of row matrix of the indices of the generators $Rels$. It compute the list of indices $[0, idx1, idx2, -idx1, -idx3, -idx2]$ of relators of type $(R4)$ and $(R4a)$ of Definition 2.5.1 and adds them to the list $Rels$. We can replace $Rels$ by empty list if we want just the list of row matrices of indices of $(R4)$. In addition it calculates the size of the list $Rels$. It returns $[Rels, sRels]$.

2.7.13 `APCGRelationR5` Function

The inputs of the function **`APCGRelationR5`**$(A, St, Lk, Rels, T)$ are the list A is defined earlier, the list of stars St, the list of links Lk, the list of row matrix of the indices of the generators $Rels$, and the list of the generators T of $\mathrm{Aut}(\mathrm{G}_\Gamma)$ from Section 2.7.6. It computes the list of indices $[2, idx1, idx2, idx4, -idx3, j, k, j]$ of relators of type $(R5)$ of Definition 2.5.1, where 2 means that the $idx4$ refers to the location of A's (which are start at $sA + 1$ and end at $sA + sGens2$), j and k refer to the vertex or its inverse, and adds them to the list $Rels$. We can replace $Rels$ by empty list if we want just the list of row matrices of indices of $(R5)$. In addition it calculates the sizes of the list $Rels$. It returns $[Rels, sRels]$.

2.7.14 `APCGRelationR8` Function

The inputs of the function **`APCGRelationR8`**$(V, A, T, Lk, Rels)$ are the list of vertices V, the list A is defined earlier, the list of the generators T of $\mathrm{Aut}(\mathrm{G}_\Gamma)$ from Section 2.7.6, the list of links Lk, and the list of row matrix of the indices of the generators

Rels. It computes the lists of indices $[0, idx1, -idx3, -idx2]$, $[0, idx1, -idx2]$, and $[0, idx1]$ of relators of type $(R8)$ of Definition 2.5.1 and adds them to the list *Rels*. We can replace *Rels* by empty list if we want just the list of row matrices of indices of $(R8)$. In addition it calculates the sizes of the list *Rels*. It returns $[Rels, sRels]$.

2.7.15 `APCGRelationR9` Function

The inputs of the function `APCGRelationR9`$APCGRelationR9(V, A, T, Lk, Rels)$ are the list of vertices V, the list A is defined earlier, the list of the generators T of $\mathrm{Aut(G_\Gamma)}$ from Section 2.7.6, the list of links Lk, and the list of row matrix of the indices of the generators *Rels*. It computes the list of indices $[0, idx1, idx2, -idx1, -idx2]$ of relators of type $(R9)$ of Definition 2.5.1 and adds them to the list *Rels*. We can replace *Rels* by empty list if we want just the list of row matrices of indices of $(R9)$. In addition it calculates the sizes of the list *Rels*. It returns $[Rels, sRels]$.

2.7.16 `APCGRelationR10` Function

The inputs of the function `APCGRelationR10`$(V, A, T, Lk, Rels)$ are the list of vertices V, the list A is defined earlier, the list of the generators T of $\mathrm{Aut(G_\Gamma)}$ from Section 2.7.6, the list of links Lk, the list of row matrix of the indices of the generators *Rels*. It computes the list of indices $[0, idx1, idx2, -idx1, -idx2, -idx3]$ of relators of type $(R10)$ of Definition 2.5.1 and adds them to the list *Rels*. We can replace *Rels* by empty list if we want just the list of row matrices of indices of $(R10)$. In addition it calculates the sizes of the list *Rels*. It returns $[Rels, sRels]$.

2.7.17 `APCGFinalReturn` Function

The input of `APCGFinalReturn`$(gens, Rels, sRels, sRels1, Rels1, sgenss)$ are the list of generators *gens*, the list of the indices of the relators *Rels*, its size *sRels*, the list of the matrices indices of the relators $Rels1$, it size $sRels1$ and *sgenss* the size of the list *genss* defined in Section 2.7.7. It forms the list of relations *rels* from the list *Rels* (computed in the functions `RelationsOfGraphAutomorphisms`, `APCGRelationR1`, `APCGRelationR2`,..., `APCGRelationR10`). For each index R of one of these lists the relator W_R is added to *rels*. It also forms the list of relations $rels1$ from the list $Rels1$ (computed in the functions `WhiteheadAutomorphismsOfFirstType`) and adds them to the list $rels1$, and then adds it to the list of relations *rels*. At the same time it computes the sizes of *rels* and *rels*1. It computes the free group F on *gens* defined in Section 2.7.7. Also it computes the finitely presented group $GGG = F/rels$ where F is the free group on the generators *gens* defined in Section 2.7.7 and *rels* is the list of relations which are defined on the generators *gens*. Finally, it returns $[F, gens, rels, GGG, sgens, srels]$. In fact this function forms the output of one of the main functions which is `FinitePresentationOfAutParCommGrp` in our package *AutParCommGrp*. The algorithm carries out the following instructions:

```
APCGFinalReturn(gens, Rels, sRels, sRels1, Rels1, sgenss)
F ← FreeGroup(gens)
gens ← GeneratorsOfGroup(F)
sgens ← Size(gens)
for i in {1, ..., sRels1}
    do GHK ← Size(Rels1[i])
       GHK1 ← GHK/2        ▷ Find real length of each single relation
       for j in {1, ..., GHK1}
           do Form rels1 the list of relators of graph group from Rels1
              srels1 ← Size(rels1)
for i in {1, ..., sRels}
    do GHK ← Size(Rels[i])
       Form rels the list of relators of the group from Rels
for i in {1, ..., srels1}
    do Add the list rels1 to the list rels
       srels ← Size(rels)
GGG ← F/rels
return [F, gens, rels, GGG, sgens, srels]
```

2.7.18 `FinitePresentationOfAutParCommGrp` Function

The function `FinitePresentationOfAutParCommGrp`(V, E) is the first main function in our algorithm. It provides a finite presentation for automorphism group $Aut(G_\Gamma)$ of G_Γ. The input of this function is a simple graph $\Gamma = (V, E)$, where V and E represent the set of vertices and the set of edges respectively. It returns $[gens, rels, GGG]$. The algorithm carries out the following instructions:

```
FinitePresentationOfAutParCommGrp(V, E)
if Γ is simple graph
   then Call The Function StarLinkDominateOfVertex
        Call The Function DeleteVerticesFromGraph
        Call Function WhiteheadAutomorphismsOfSecondType
        Call Function WhiteheadAutomorphismsOfFirstType
        Call The Function RelationsOfGraphAutomorphisms
        Call The Function APCGRelationR5
        Call The Function APCGRelationR1
        Call The Function APCGRelationR2
        Call The Function APCGRelationR3
        Call The Function APCGRelationR4
        Call The Function APCGRelationR8
        Call The Function APCGRelationR9
        Call The Function APCGRelationR10
        Call The Function APCGFinalReturn
   else return "The graph must be a simple graph"
return [gens, rels, GGG]
```

Where,

(i) *gens*: is a list of free generators of the automorphism group $Aut(G_\Gamma)$ of G_Γ.

(ii) *rels*: is a list of relations in the generators of the free group. Note that relations are entered as relators, i.e., as words in the generators of the free group.

(iii) $GGG := F/rels$: is the automorphism group $Aut(G_\Gamma)$ of G_Γ given as a finitely presented group with generators *gens* and relators *rels*.

For example,

```
gap> B:=FinitePresentationOfAutParCommGrp([1,2],[[1,2]]);

[ [ A1, A2, A3, A4, A5, A6, A7, A8, A9, A10, f1 ], [A9^2, A10^2,
A9^-1*A10^-1*A9*A10, A10^-1*A9^-1*A10*A9, f1^-1*A9*f1*A10,
f1^-1*A10*f1*A9,A7*A1*A10*A3^-1,A5*A2*A10*A4^-1,A8*A3*A10*A1^-1,
A6*A4*A10*A2^-1, A3*A5*A9*A7^-1, A1*A6*A9*A8^-1, A4*A7*A9*A5^-1,
A2*A8*A9*A6^-1, A1*A2, A3*A4, A5*A6, A7*A8, A1*A3, A2*A4, A3*A1,
```

```
A4*A2, A5*A7, A6*A8, A7*A5, A8*A6, A1*A4^-1, A2*A3^-1, A3*A2^-1,
A4*A1^-1, A5*A8^-1, A6*A7^-1, A7*A6^-1, A8*A5^-1, f1^2 ],
<fp group on the generators [A1, A2, A3, A4, A5, A6, A7, A8, A9,
A10, f1 ]> ]

gap> B:=FinitePresentationOfAutParCommGrp([1,2],[]);
[ [ A1, A2, A3, A4, A5, A6, A7, A8, A9, A10, A11, A12, A13, A14,
f1], [ A13^2, A14^2, A13^-1*A14^-1*A13*A14, A14^-1*A13^-1*A14*A13,
f1^-1*A13*f1*A14,f1^-1*A14*f1*A13,A9*A1*A14*A3^-1,A7*A2*A14*A4^-1,
A10*A3*A14*A1^-1,A8*A4*A14*A2^-1,A3*A7*A13*A9^-1,A1*A8*A13*A10^-1,
A4*A9*A13*A7^-1, A2*A10*A13*A8^-1, A1*A2, A3*A4, A5*A6, A7*A8,
A9*A10, A11*A12, A1*A3*A5^-1, A2*A4*A6^-1,A3*A1*A5^-1,A4*A2*A6^-1,
A7*A9*A11^-1, A8*A10*A12^-1, A9*A7*A11^-1, A10*A8*A12^-1,
A1*A4^-1*A5^-1, A2*A3^-1*A6^-1, A3*A2^-1*A5^-1,A4*A1^-1*A6^-1,
<identity ...>, <identity ...>, A7*A10^-1*A11^-1, A8*A9^-1*A12^-1,
A9*A8^-1*A11^-1, A10*A7^-1*A12^-1, <identity ...>, <identity ...>,
A1*A11*A1^-1*A11^-1*A5^-1, A2*A11*A2^-1*A11^-1*A6^-1,
A3*A12*A3^-1*A12^-1*A5^-1, A4*A12*A4^-1*A12^-1*A6^-1,
7*A5*A7^-1*A5^-1*A11^-1, A8*A5*A8^-1*A5^-1*A12^-1,
A9*A6*A9^-1*A6^-1*A11^-1, A10*A6*A10^-1*A6^-1*A12^-1, f1^2 ],
<fp group on the generators [ A1, A2, A3, A4, A5, A6, A7, A8, A9,
A10, A11, A12, A13, A14, f1 ]> ]
```

Remark 2.7.2. We use the standard GAP function `AssignGeneratorVariables`(G) to makes our generators readable by *GAP*. If G is a group, whose generators are represented by symbols this function assigns these generators to global variables with the same names. The aim of this function is to make the generators work interactively and more conveniently with *GAP*; for more information see (37.2.3) of the *GAP* Manuals.

For example from the output of `FinitePresentationOfAutParCommGrp`($[1,2]$, $[[1,2]]$) above we have:

```
gap> G:=B[3];
<fp group on the generators [ A1, A2, A3, A4, A5, A6, A7, A8, A9,
A10, f1 ]>
```

```
gap> AssignGeneratorVariables(G);
#I  Assigned the global variables [ A1, A2, A3, A4, A5, A6,A7, A8,
A9, A10, f1 ]
```

2.7.19 TietzeTransformations Function

The aim of the function `TietzeTransformations`(G) is to simplify the presentation of the finitely presented group G, i.e., to reduce the number of generators, the number of relators and the relator lengths. The input of the function `TietzeTransformations` is a finite presentation of G. The operation returns a group H isomorphic to G, so that the presentation of H has been simplified using Tietze transformations. The algorithm carries out the following instructions:

TietzeTransformations(G)
1 $hom \leftarrow$ IsomorphismSimplifiedFpGroup(G)
2 $H \leftarrow$ Image(hom)
3 $R \leftarrow$ RelatorsOfFpGroup(H)
4 **return** $[H, R]$

For example, using the output of `FinitePresentationOfAutParCommGrp`($[1,2]$, $[[1,2]]$) in Section 2.7.18 we have that,

```
gap> G:=B[3];
<fp group on the generators [ A1, A2, A3, A4, A5, A6, A7, A8, A9,
A10, f1 ]>
gap> D:=TietzeTransformations(G);
[ <fp group on the generators [ A1, A10, f1 ]>, [ A10^2, f1^2,
A10*f1*A10*f1*A10*f1*A10*f1, A10*A1^-1*f1*A10*f1*A1^-1*A10*A1^-1 ] ]
```

Chapter 3

Finite Presentation for the Subgroup $Conj(G_\Gamma)$

3.1 Introduction

The subgroup of $Aut(G_\Gamma)$, which we consider here, plays an important role in the structure of $Aut(G_\Gamma)$: see for example [34], [35], [38], [57] and [61]. Recall that the set of all basis conjugating automorphisms forms a subgroup $Conj(G_\Gamma)$ generated by partial conjugations (see Chapter 2). A finite presentation for the the subgroup $Conj(G_\Gamma)$ is given in [70].

Our aim in this chapter is to develop an algorithm using *GAP* system that provides a finite presentation for the subgroup $Conj(G_\Gamma)$. In addition, we find Tietze transformations to simplify the presentation of $Conj(G_\Gamma)$; using a *GAP* function. In order to do this work we will give a description of the presentation of the subgroup $Conj(G_\Gamma)$ according to Toinet's work [70].

Note that amongst the partial conjugations we have the inner automorphisms; so some of the generators of $Conj(G_\Gamma)$ are inner automorphisms.

3.2 Finite Presentation for $Conj(G_\Gamma)$

In [70], Toinet computed a finite presentation for the subgroup $Conj(G_\Gamma)$ of $Aut(G_\Gamma)$ generated by partial conjugations. In this section we will describe this presentation following Toinet's paper.

Let Ω be the set of Whitehead automorphisms. We set Ω_1 to be the set of Whitehead automorphisms of type (1), and Ω_2 to be the set of Whitehead automorphisms of type (2). We also denote by Ω_ℓ the set of long-range Whitehead automorphisms.

Note that, as we have mentioned in Chapter 2, Day in [24] proved that $Aut(G_\Gamma)$ is generated by the Whitehead automorphisms, with the relations $(R1)$ to $(R10)$ given in Definition 2.5.1.

In following we will apply the definition of peak reduced (see 2.6.1).

Theorem 3.2.1. *[70] The subgroup $Conj(G_\Gamma)$ has a presentation $\langle S|R\rangle$ where S is the set of partial conjugations $c_{x,Y}$, for $x \in L$ and Y a non-empty union of connected components of $\Gamma\backslash st(x)$), and R is the finite set of relations:*

(C1) $(c_{x,Y})^{-1} = c_{x^{-1},Y}$,

(C2) $c_{x,Y}c_{x,Z} = c_{x,Y\cup Z}$ *if* $Y \cap Z = \emptyset$,

(C3) $c_{x,Y}c_{y,Z} = c_{y,Z}c_{x,Y}$ *if* $x \notin Z$, $y \notin Y$, $x \neq y, y^{-1}$, *and at least one of* $Y \cap Z = \emptyset$ *or* $y \in \ell k_L(x)$ *holds,*

(C4) $\gamma_y c_{x,Y}\gamma_y^{-1} = c_{x,Y}$ *if* $y \notin Y$, $x \neq y, y^{-1}$.

Proof. The proof is based on arguments developed by McCool in [56] and [57] (similar arguments were used in [24]). Let S denote the set of partial conjugations $c_{x,Y}$ where $x \in L$. Let R denote the set of relations given in the statement of Theorem 3.2.1. We shall construct a finite connected 2-complex K with fundamental group

$$Conj(G_\Gamma) = \langle S \mid R\rangle.$$

We identify a partial conjugation with any of its representatives in Ω_2. Note that, for every $(A,a) \in \Omega_2$, $(A,a) \in S$ if and only if $(A-a)^{-1} = A-a$.

Set $\mathcal{V} = \{v_1,\ldots,v_n\}(n \geq 1)$. Let W denote the n-tuple $(v_1,\ldots,v_n)$. The set of vertices $K^{(0)}$ of K is the set of n-tuples $\alpha \cdot W$, where α ranges over and set Ω_1 of type (1) Whitehead automorphisms. For and $\alpha, \beta \in \Omega_1$, the vertices $\alpha \cdot W$ and $\beta\alpha \cdot W$ are joined by a Directed edge $(\alpha\cdot W, \beta\alpha\cdot W;\beta)$ labelled β. Note that, at this stage, K is just the Cayley graph of Ω_1. Next, for any $\alpha \in \Omega_1$, and $(A,a) \in S$, we add a loop $(\alpha\cdot W,\alpha\cdot W;(A,a))$ labelled (A,a) at $\alpha\cdot W$. This defines the 1-skeleton $K^{(1)}$ of K.

We shall define the 2-cells of K. These 2-cells will derive from the relations (R1)-(R10) of Definition 2.5.1. First, let K_1 be the 2-complex obtained by attaching

2-cells corresponding to relation $(R7)$ of Definition 2.5.1 to $K^{(1)}$. Note that, if C is the 2-complex obtained from K_1 by deleting the loops $(\alpha \cdot W, \alpha \cdot W; (A, a))(\alpha \in \Omega_1, (A, a) \in S)$, then C is just the Cayley complex of Ω_1, and therefore is simply connected. We now explore the relations (R1)-(R5) and (R8)-(R10) of Definition 2.5.1 to determine which of these will give rise to relations on the elements of S. Relation (R1) of Definition 2.5.1 will give rise to the following:

$$(A, a)^{-1} = (A - a + a^{-1}, a^{-1}) \tag{3.2.1}$$

for $(A, a) \in S$.

Relation (R2) of Definition 2.5.1 will give rise to

$$(A, a)(B, a) = (A \cup B, a) \tag{3.2.2}$$

for $(A, a), (B, a) \in S$, with $A \cap B = \{a\}$.

Relation (R3) of Definition 2.5.1 will give rise to

$$(A, a)(B, b) = (B, b)(A, a), \tag{3.2.3}$$

for $(A, a), (B, b) \in S$, such that $a \notin B, a^{-1} \notin B, b \notin A$, $b^{-1} \notin A$, and at least one of (a) $A \cap B = \emptyset$ or (b) $b \in \ell k_L(a)$ holds.

From relation (R4) of Definition 2.5.1, no relations arise. Indeed, suppose that $(A, a), (B, b)$ are in S with $a^{-1} \notin B, b \notin A$, and $b^{-1} \in A$. Then $b^{-1} = a$ (because $(A - a)^{-1} = A - a)$. But then $a^{-1} = b \in B$, leading to a contradiction with our assumption on a.

From relation (R5) of Definition 2.5.1, no relations arise (by the same argument as above).

From relation (R8) of Definition 2.5.1, we obtain a relation which is a direct consequence of (3.2.1) and (3.2.2).

Relation (R9) of Definition 2.5.1 will give rise to the following:

$$(A, a)(L - \ell k_L(b) - b^{-1}, b)(A, a)^{-1} = (L - \ell k_L(b) - b^{-1}, b) \tag{3.2.4}$$

for $(A, a) \in S$, and $b \in L$ such that $b \notin A$, and $b^{-1} \notin A$.

From relation $(R10)$ of Definition 2.5.1, no relations arise (by the same argument as above).

We rewrite the relations (3.2.1)-(3.2.4) in the form

$$\sigma_k^{\epsilon_k} \dots \sigma_1^{\epsilon_1} = 1$$

where $\sigma_1, \dots, \sigma_k \in S$ and $\epsilon_1, \dots, \epsilon_k \in \{-1, 1\}$. Let K_2 be the 2-complex optioned from K_1 by attaching 2-cells corresponding to the relations (3.2.1)-(3.2.4). Note that the boundary of each of these 2-cells has the from

$$(\alpha \cdot W, \alpha \cdot W; \sigma_1)^{\epsilon_1} (\alpha \cdot W, \alpha \cdot W; \sigma_2)^{\epsilon_2} \dots (\alpha \cdot W, \alpha \cdot W; \sigma_k)^{\epsilon_k},$$

for $\alpha \in \Omega_1$.

Finally, relation relation (R6) of Definition 2.5.1, will give rise to the following:

$$\alpha(A, a)\alpha^{-1} = (\alpha(A), \alpha(a)), \tag{3.2.5}$$

for $(A, a) \in S$, and $\alpha \in \Omega_1$. Then K is obtained from K_2 by attaching 2-cells corresponding to the relations (3.2.5). Observe that the boundary of each of these 2-cells has the form

$(\beta \cdot W, \beta \cdot W; (\alpha(A), \alpha(a)))^{-1} (\beta \cdot W, \alpha^{-1}\beta \cdot W; \alpha)^{-1} (\alpha^{-1}\beta \cdot W, \alpha^{-1}\beta \cdot W; (A, a)) (\alpha^{-1}\beta \cdot W, \beta \cdot W; \alpha)$, for $\beta \in \Omega_1$.

It remains to show that $\pi_1(K, W) = Conj(G_\Gamma) = \langle S \mid R \rangle$.

Let T be a maximal tree in the 1-skeleton $K^{(1)}$ of K. Note that T is in fact a maximal tree in the 1-skeleton $C^{(1)}$ of C (i.e., the Cayley graph of Ω_1). We compute a presentation of $\pi_1(K, W)$ using T. For every vertex V in K, there exists a unique reduced path pv from W to V in T. To each edge $(V_1, V_2; \alpha)$ of K, we associate the element $\pi_1(K, W)$ represented by the loop $pv_1(V_1, V_2; \alpha)p_{V_2}^{-1}$. We again denote this by $(V_1, V_2; \alpha)$. Evidently these elements generate $\pi_1(K, W)$. Now, since C is simply connected, we have

$$(\alpha \cdot W, \beta\alpha \cdot W; \beta) = 1 \ \ (in\ \pi_1(K, W)), \tag{3.2.6}$$

for all $\alpha, \beta \in \Omega_1$.

Let $\mathcal{P}$ be the set of combinatorial in the 1-skeleton $K^{(1)}$ of K. We define a map $\widehat{\varphi} : \mathcal{P} \to Aut(G_\Gamma)$ as follows. For an edge $e = (V_1, V_2; \alpha)$, we set $\widehat{\varphi}(e) = \alpha$, and for a path $p = e_k^{\epsilon_k} \dots e_1^{\epsilon_1}$, we set $\widehat{\varphi(p)} = \widehat{\varphi}(e_k)^{\epsilon_k} \dots \widehat{\varphi}(e_1)^{\epsilon_1}$. Clearly, if p_1 and p_2 are loops at W such that $p_1 \sim p_2$, then $\widehat{\varphi}(p_1) = \widehat{\varphi}(p_2)$. Hence, $\widehat{\varphi}$ induces a map $\varphi : \pi_1(K, W) \to Aut(G_\Gamma)$. It is easily seen that φ is a homomorphism. Then we see from (3.2.6) that φ maps $\pi_1(K, W)$ to $Conj(G_\Gamma)$. It follows immediately from the construction of K that $\varphi : \pi_1(K, W) \to Aut(G_\Gamma)$ is surjective. Thus, it suffices to show that φ is injective. Let p be a loop at W such that $\varphi(p) = 1$. We have to show that $p \sim 1$. Write $p = e_k^{\epsilon_k} \dots e_1^{\epsilon_1}$, where $k \geq 1$ and $\epsilon_i \in \{-1, 1\}$ for all $i \in \{1, \dots, k\}$. Using the 2-cells arising from (3.2.1) and the fact that $\Omega_1^{-1} = \Omega_1$, we can restrict our attention to the case where $p = e_k \dots e_1$. Set $\alpha_i = \varphi(e_i)$ for all $i \in \{1, \dots, k\}$. Note that $\alpha_i \in S \cup \Omega_1 \subset \Omega_\ell$ for all $i \in \{1, \dots, k\}$.

Let Z be a tuple containing each conjugacy class of length 2 of G_Γ, each appearing once. We prove the following:

claim. We have $p \sim e'_1 \dots e'_1$, such that, if we set $\alpha'_i = \varphi(e_i)$ for all $i \in \{1, \dots, l\}$, then $(\alpha'_i \in \Omega_1$ or $(\alpha'_i \in \Omega_2 \cap Inn(G_\Gamma)$ for each $i \in \{1, \dots, l\}$.

First, we examine the case where $\alpha_k \dots \alpha_1$ is peak-reduced with respect to Z. We claim that the sequence

$$\mid Z \mid, \mid \alpha_1 \cdot Z \mid, \mid \alpha_2\alpha_1 \cdot Z \mid, \dots, \mid \alpha_{k-1} \dots \alpha_1 \cdot Z \mid, \mid \alpha_k \dots \alpha_1 \cdot Z \mid = \mid Z \mid$$

is a constant sequence. Suppose the contrary. By Lemma 2.6.4, $\mid Z \mid$ is the least element of the set $\{\mid \alpha \cdot Z \mid \mid \alpha \in \langle \Omega_\ell \rangle\}$. Hence we can find $i \in \{1, \dots, k-1\}$ such that we have

$$\mid \alpha_{i-1} \dots \alpha_1 \cdot Z \mid \leq \mid \alpha_i \dots \alpha_1 \cdot Z \mid,$$

$$\mid \alpha_{i+1} \dots \alpha_1 \cdot Z \mid \leq \mid \alpha_i \dots \alpha_1 \cdot Z \mid,$$

and at least one of these inequalities is strict, which contradicts the fact that the product $\alpha_k \dots \alpha_1$ is peak-reduced. Therefore we have

$$\mid \alpha_i \dots \alpha_1 \cdot Z \mid = \mid Z \mid,$$

for all indices $i \in \{1, \dots, k\}$. We argue by induction on $i \in \{1, \dots, k\}$ to prove that $\{\alpha_i \dots \alpha_1\} \cdot Z$ is a tuple containing each conjugacy class of length 2 of G_Γ, each appearing once. The result holds for $i = 0$ by assumption. Suppose that $i \geq 1$, and that the result holds for $i - 1$. Observe that a type (1) Whitehead automorphism does not change the length of a conjugacy class. Thus, we can assume that α_i is a type (2) Whitehead automorphism. Since $\mid \alpha_i\alpha_{i-1} \dots \alpha_1 \cdot Z \mid = \mid \alpha_{i-1} \dots \alpha_1 \cdot Z \mid, \alpha_i$ is trivial, or an inner automorphism by Lemma 2.6.4. Thus, the result holds for i. In this case, p has already the desired from.

We now turn to prove the claim. We define

$$h_p = max\{| \alpha_i \dots \alpha_1 \cdot Z || i \in \{0, \dots, k\}\}$$

and

$$N_p =| \{i \mid i \in \{0, \dots, k\} and \mid \alpha_i \dots \alpha_1 \cdot Z |= h_p\} \mid .$$

We argue by induction on h_p. The base of induction is $\mid Z \mid$, i.e. the smallest possible value for h_p by Lemma 2.6.4. If $h_p =\mid Z \mid$, then the product $\alpha_k \dots \alpha_1$ is peak-reduced and we are done. Thus, we can assume that $h_p >\mid Z \mid$ and that the result has been proved for all loop p' with $h_{p'} < h_p$. Let $i \in \{1, \dots, k\}$ be such that α_i is a peak of height h_p. An examination of the proof of Lemma 2.6.5 shows that $e_{i+1}e_i \sim f_j \dots f_1$ such that, if we set $\beta_k = \varphi(f_k)$ for all $k \in \{1, \dots, j\}$, then

$$\mid \beta_k \dots \beta_1 \alpha_{i-1} \dots \alpha_1 \cdot Z \mid < \mid \alpha_i \alpha_{i-1} \dots \alpha_1 \cdot Z \mid \tag{3.2.7}$$

for all $k \in \{1, \dots, j-1\}$. Therefore, we get

$$p \sim e_k \dots e_{i+2} f_j \dots f_1 e_{i-1} \dots e_1 = p',$$

and a new product $\alpha_k \dots \alpha_{i+2} \beta_j \dots \beta_1 \alpha_{i-1} \dots \alpha_1$. We argue by induction on N_p. If $N_p = 1$, then (3.2.7) implies that $h_{p'} < h_p$ and $N_{p'} < N_p$, and we can apply the induction hypothesis on n_p. This proves the claim.

Hence, using the 2-cells arising from the relations (3.2.5), we obtain

$$p \sim h_S \dots h_1 g_r \dots g_1,$$

where, if we set

$$\gamma_i = \varphi(g_i) \text{ for all } i \in \{1, \dots, r\} \text{ and } \delta_j = \varphi(h_j) \text{ for all } j \in \{1, \dots, s\},$$

then $\delta_i \in \Omega_1$ for all $i \in \{1, \dots, s\}$ and $\gamma_i \in \Omega_2 \cap Inn(G_\Gamma)$ for all $j \in \{1, \dots, r\}$. Using relation (3.2.6), we obtain $p \sim g_r \dots g_1$. Set $\mathcal{Z} = \cap_{v \in \mathcal{V}} st(v)$. It follows from Servatius' Centralizer Theorem (see [69]) that the center $Z(G_\Gamma)$ of G_Γ is the special subgroup of G_Γ generated by $\mathcal{Z}$. Let Γ' be the full subgraph of Γ spanned by $\mathcal{V} \setminus \mathcal{Z}$. We have

$$G_{\Gamma'} \simeq Inn(G_\Gamma),$$

$\gamma_i = \varphi(g_i)$ for all $i \in \{1, \dots, r\}$ and $\delta_j = \varphi(h_j)$ for all $j \in \{1, \dots, S\}$,

where the isomorphism is given by $v \mapsto w_v$ (see, for example, [2, Lemma 5.3]). Write

$$\gamma_i = (L - \ell k_L(c_i) - c_i^{-1}, c_i),$$

where $c_i \in (\mathcal{V} \setminus \mathcal{Z}) \cup (\mathcal{V} \setminus \mathcal{Z})^{-1} (i \in \{1, \dots, r\})$. Since $\gamma_r \dots \gamma_1 = 1$ (in $Inn(G_\Gamma)$), we have $c_r \dots c_1 = 1$ (in $G_{\Gamma'}$). Therefore $c_r \dots c_1$ is a product of conjugates of defining relators of G_Γ. Using the 2-cells corresponding to the relations (3.2.1) and (3.2.3)(b), we deduce that $p \sim 1$. We conclude that φ is injective, and thus

$$Conj(G_\Gamma) = \pi_1(K, W).$$

Now, using the 2-cells arising from the relations (3.2.5) (with $\alpha = \beta$), we obtain

$$(\alpha \cdot W, \alpha \cdot W; (\alpha(A), \alpha(a))) = (\alpha \cdot W, W; \alpha^{-1})(W, W; (A, a))(W, \alpha \cdot W; \alpha),$$

and then, using (3.2.6)

$$(\alpha \cdot W, \alpha \cdot W; (\alpha(A), \alpha(a))) = (W, W; (A, a)), \tag{3.2.8}$$

for all $\alpha \in \Omega_1$, and $(A, a) \in S$. It then follows that $Conj(G_\Gamma)$ is generated by the $(W, W; (A, a))$, for $(A, a) \in S$. We identify $(W, W; (A, a))$ with (A, a) for all $(A, a) \in S$. Any relation in $Conj(G_\Gamma) = \pi_1(K, W)$ will be a product of conjugates of boundary lables of 2-cells of K. Then, using relation (3.2.8) and identifying $(W, W; (A, a))$ with (A, a), we see that these relations (3.2.1)-(3.2.4) aboe are equivalent to those of R. We have shown that $Conj(G_\Gamma)$ has the presentation $\langle S \mid R \rangle$.
$\square$

Now we will give a small example to find a finite presentation of a subgroup $Conj(G_\Gamma)$ of $Aut(G_\Gamma)$,

Example 3.2.0.1

Consider the graph Γ of Figure 3.1

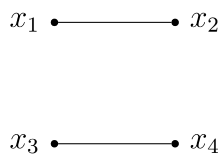

Figure 3.1: A Graph Γ

Then $V = \{x_1, x_2, x_3, x_4\}$ *and* $E = \{\{x_1, x_2\}, \{x_3, x_4\}\}$. Let $Conj(G_\Gamma)$ be a subgroup of $Aut(G_\Gamma)$. Then,

(1) $St(x_1) = \{x_1, x_2\}$,

$Lk\{x_1\} = \{x_2\}$,

$Comps1 = \{x_4^{-1}, x_3^{-1}, x_3, x_4\}$ = the connected components of $\Gamma \backslash St(x_1)$.

(2) $St(x_2) = \{x_2, x_1\}$

$Lk(x_2) = \{x_1\}$,

$Comps2 = \{x_4^{-1}, x_3^{-1}, x_3, x_4\}$ = the connected components of $\Gamma \backslash St(x_2)$.

(3) $St(x_3) = \{x_3, x_4\}$,

$Lk(x_3) = \{x_4\}$,

$Comps3 = \{x_2^{-1}, x_1^{-1}, x_1, x_2\}$ = the connected components of $\Gamma \backslash St(x_3)$.

(4) $St(x_4) = \{x_4, x_3\}$,

$Lk(x_4) = \{x_3\}$

$Comps4 = \{x_2^{-1}, x_1^{-1}, x_1, x_2\}$ = the connected components of $\Gamma \backslash St(x_4)$.

- We find Y which is a non-empty union of connected components of $\Gamma \backslash st(x)$, where $x \in L$:

  $Y = \{Y_1 = \{x_4^{-1}, x_3^{-1}, x_3, x_4\}, Y_2 = \{x_2^{-1}, x_1^{-1}, x_1, x_2\}\}$

- Now, we find $c_{x,Y}$, the partial conjugations that form the first part of the set of the generators of $Conj(G_\Gamma)$:

  $C_{x,Y} = \{c_{x_2^{-1},Y_1} = \{\{x_4^{-1}, x_3^{-1}, x_3, x_4, x_2^{-1}\}, x_2^{-1}\},$

$$
\begin{aligned}
c_{x_1^{-1},Y_1} &= \{\{x_4^{-1}, x_3^{-1}, x_3, x_4, x_1^{-1}\}, x_1^{-1}\},\\
c_{x_1,Y_1} &= \{\{x_4^{-1}, x_3^{-1}, x_3, x_4, x_1\}, x_1\},\\
c_{x_2,Y_1} &= \{\{x_4^{-1}, x_3^{-1}, x_3, x_4, x_2\}, x_2\},\\
c_{x_4^{-1},Y_2} &= \{\{x_2^{-1}, x_1^{-1}, x_1, x_2, x_4^{-1}\}, x_4^{-1}\},\\
c_{x_3^{-1},Y_2} &= \{\{x_2^{-1}, x_1^{-1}, x_1, x_2, x_3^{-1}\}, x_3^{-1}\}\\
c_{x_3,Y_2} &= \{\{x_2^{-1}, x_1^{-1}, x_1, x_2, x_3\}, x_3\},\\
c_{x_4,Y_2} &= \{\{x_2^{-1}, x_1^{-1}, x_1, x_2, x_4\}, x_4\}\}.
\end{aligned}
$$

- We find w, the inner automorphisms that form the second part of the set of the generators of $Conj(G_\Gamma)$ (since every inner automorphism is a partial conjugation):

$$
\begin{aligned}
W = \{&w_{x_2^{-1}} = \{\{x_4^{-1}, x_3^{-1}, x_2^{-1}, x_3, x_4\}, x_2^{-1}\},\ w_{x_1^{-1}} = \{\{x_4^{-1}, x_3^{-1}, x_1^{-1}, x_3, x_4\}, x_1^{-1}\},\\
&w_{x_1} = \{\{x_4^{-1}, x_3^{-1}, x_1, x_3, x_4\}, x_1\},\ w_{x_2} = \{\{x_4^{-1}, x_3^{-1}, x_2, x_3, x_4\}, x_2\},\\
&w_{x_4^{-1}} = \{\{x_4^{-1}, x_2^{-1}, x_1^{-1} x_1, x_2\}, x_4^{-1}\},\ w_{x_3^{-1}} = \{\{x_3^{-1}, x_2^{-1}, x_1^{-1} x_1, x_2\}, x_3^{-1}\},\\
&w_{x_3} = \{\{x_2^{-1}, x_1^{-1}, x_1, x_2, x_3\}, x_3\},\ w_{x_4} = \{\{x_2^{-1}, x_1^{-1}, x_1, x_2, x_4\}, x_4\}\}.
\end{aligned}
$$

- We find S, the set of the generators of $Conj(G_\Gamma)$, which is equal to the union of $C_{x,Y}$ and W:

$$
\begin{aligned}
S = \{&c_{x_2^{-1},Y_1} = \{\{x_4^{-1}, x_3^{-1}, x_3, x_4, x_2^{-1}\}, x_2^{-1}\}, c_{x_1^{-1},Y_1} = \{\{x_4^{-1}, x_3^{-1}, x_3, x_4, x_1^{-1}\}, x_1^{-1}\},\\
&c_{x_1,Y_1} = \{\{x_4^{-1}, x_3^{-1}, x_3, x_4, x_1\}, x_1\}, c_{x_2,Y_1} = \{\{x_4^{-1}, x_3^{-1}, x_3, x_4, x_2\}, x_2\},\\
&c_{x_4^{-1},Y_2} = \{\{x_2^{-1}, x_1^{-1}, x_1, x_2, x_4^{-1}\}, x_4^{-1}\}, c_{x_3^{-1},Y_2} = \{\{x_2^{-1}, x_1^{-1}, x_1, x_2, x_3^{-1}\}, x_3^{-1}\}\\
&c_{x_3,Y_2} = \{\{x_2^{-1}, x_1^{-1}, x_1, x_2, x_3\}, x_3\},\ c_{x_4,Y_2} = \{\{x_2^{-1}, x_1^{-1}, x_1, x_2, x_4\}, x_4\}\}.
\end{aligned}
$$

- We find R, the set of relations according to the relations that are defined in Theorem 4.2.3:

$R = \{c_{x_2^{-1},Y_1} * c_{x_2,Y_1}, c_{x_1^{-1},Y_1} * c_{x_1,Y_1}, c_{x_1,Y_1} * c_{x_1^{-1},Y_1}, c_{x_2,Y_1} * c_{x_2^{-1},Y_1}, c_{x_4^{-1},Y_2} * c_{x_4,Y_2}, c_{x_3^{-1},Y_2} * c_{x_3,Y_2}, c_{x_3,Y_2} * c_{x_3^{-1},Y_2}, c_{x_4,Y_2} * c_{x_4^{-1},Y_2}, c_{x_2^{-1},Y_1} * c_{x_1^{-1},Y_1} * (c_{x_2^{-1},Y_1})^{-1} * (c_{x_1^{-1},Y_1})^{-1}, c_{x_2^{-1},Y_1} * c_{x_1,Y_1} * (c_{x_2^{-1},Y_1})^{-1} * (c_{1,Y_1})^{-1}, c_{x_1^{-1},Y_1} * c_{x_2,Y_1} * (c_{x_1^{-1},Y_1})^{-1} * (c_{x_2,Y_1})^{-1}, c_{x_1,Y_1} * c_{x_2,Y_1} * (c_{x_1,Y_1})^{-1} * (c_{x_2,Y_1})^{-1}, c_{x_4^{-1},Y_2} * c_{x_3^{-1},Y_2} * (c_{x_4^{-1},Y_2})^{-1} * (c_{x_3^{-1},Y_2})^{-1}, c_{x_4^{-1},Y_2} * c_{x_3,Y_2} * (c_{x_4^{-1},Y_2})^{-1} * (c_{x_3,Y_2})^{-1}, c_{x_3^{-1},Y_2} * c_{x_4,Y_2} * (c_{x_3^{-1},Y_2})^{-1} * (c_{x_4,Y_2})^{-1}, c_{x_3,Y_2} * c_{x_4,Y_2} * (c_{x_3,Y_2})^{-1} * (c_{x_4,Y_2})^{-1},\ c_{x_1^{-1},Y_1} * c_{x_2^{-1},Y_1} * (c_{x_1^{-1},Y_1})^{-1} *$

$(c_{x_2^{-1},Y_1})^{-1}, c_{x_1,Y_1} * c_{x_2^{-1},Y_1} * (c_{x_1,Y_1})^{-1} * (c_{x_2^{-1},Y_1})^{-1}, c_{x_2^{-1},Y_1} * c_{x_1^{-1},Y_1} * (c_{x_2^{-1},Y_1})^{-1} * (c_{x_1^{-1},Y_1})^{-1}, c_{x_2,Y_1} * c_{x_1^{-1},Y_1} * (c_{x_2,Y_1})^{-1} * (c_{x_1^{-1},Y_1})^{-1}, c_{x_2^{-1},Y_1} * c_{x_1,Y_1} * (c_{x_2^{-1},Y_1})^{-1} * (c_{x_1,Y_1})^{-1}, c_{x_2,Y_1} * c_{x_1,Y_1} * (c_{x_2,Y_1})^{-1} * (c_{x_1,Y_1})^{-1}, c_{x_1^{-1},Y_1} * c_{x_2,Y_1} * (c_{x_1^{-1},Y_1})^{-1} * (c_{x_2,Y_1})^{-1}, c_{x_1,Y_1} * c_{x_2,Y_1} * (c_{x_1,Y_1})^{-1} * (c_{x_2,Y_1})^{-1}, c_{x_3^{-1},Y_2} * c_{x_4^{-1},Y_2} * (c_{x_3^{-1},Y_2})^{-1} * (c_{x_4^{-1},Y_2})^{-1}, c_{x_3,Y_2} * c_{x_4^{-1},Y_2} * (c_{x_3,Y_2})^{-1} * (c_{x_4^{-1},Y_2})^{-1}, c_{x_4^{-1},Y_2} * c_{x_3^{-1},Y_2} * (c_{x_4^{-1},Y_2})^{-1} * (c_{x_3^{-1},Y_2})^{-1}, c_{x_4,Y_2} * c_{x_3^{-1},Y_2} * (c_{x_4,Y_2})^{-1} * (c_{x_3^{-1},Y_2})^{-1}, c_{x_4^{-1},Y_2} * c_{x_3,Y_2} * (c_{x_4^{-1},Y_2})^{-1} * (c_{x_3,Y_2})^{-1}, c_{x_4,Y_2} * c_{x_3,Y_2} * (c_{x_4,Y_2})^{-1} * (c_{x_3,Y_2})^{-1}, c_{x_3^{-1},Y_2} * c_{x_4,Y_2} * (c_{x_3^{-1},Y_2})^{-1} * (c_{x_4,Y_2})^{-1}, c_{x_3,Y_2} * c_{x_4,Y_2} * (c_{x_3,Y_2})^{-1} * (c_{x_4,Y_2})^{-1}\}.$

- Hence, the finite presentation for the group of $Conj(G_\Gamma)$ is

  $Conj(G_\Gamma) = \langle S|R \rangle$

3.3 GAP Presentation for $Conj(G_\Gamma)$

This section describes the functions available from the *AutParCommGrp* package which we have written for computing a finite presentation for the subgroup $Conj(G_\Gamma)$ of $Aut(G_\Gamma)$ with commuting graph Γ generated by partial conjugations.

To write an algorithm to produce this presentation we first construct the set S of generators $c_{x,Y}$ (Laurence's generators), and then find the set R of relations defined in Theorem 3.2.1. The input of the main function `FinitePresentationOfSubgroup-Conj` that provides finite presentation for the subgroup $Conj(G_\Gamma)$ is a simple graph $\Gamma = (V, E)$. A graph with vertex set V of size n always has vertices $\{1, \ldots, n\}$ and E is a list of pairs of elements of V. For example if Γ is a simple graph with vertex set $V = \{x_1, x_2, x_3\}$ and edge set $E = \{[x_1, x_2], [x_1, x_3], [x_2, x_3]\}$ (where $[x, y]$ denotes an edge joining x to y) then Γ will be represented as $([1, 2, 3], [[1, 2], [1, 3], [2, 3]])$. The output of `FinitePresentationOfSubgroupConj` consists of two sets *gens* and *rels*, where *gens* is the list of the generators of the automorphism $c_{x,Y}$ defined above and *rels* is the list of the relators.

In addition, to the functions `IsSimpleGraph`, `DeleteverticesFromGraph` and `ConnectedComponentsOfGraph` which we have described in Sections 2.7.1, 2.7.3 and 2.7.4 respectively the function `FinitePresentationOfSubgroupConj` runs the following functions:

3.3.1 `StarLinkOfVertex` Function

The input of the function `StarLinkOfVertex`(V, E) is a simple graph $\Gamma = (V, E)$, where V and E represents the list of vertices and the list of edges respectively. It computes the star $St(v)$ and the link $Lk(v)$ and concatenates them in two separate lists St and Lk respectively. The algorithm carries out the following instructions:

```
StarLinkOfVertex(V, E)
sV ← Size(V)
M ← Size(E)
St ← NullMat(sV, 1, 0)
for v in V(Γ)
      do Add v to St[v]
         for e in E(Γ)
              do if e is adjacent v
                     then Add "end point" of e to St[v]
for v in V(Γ)
      do Y2 ← Set(St[v])
         Y3 ← RemoveSet(Y2, v)
         Add Y3 to new list Lk
return [St, Lk]
```

3.3.2 `CombinationsOfConnectedComponents` Function

The input of the function `CombinationsOfConnectedComponents`$(Comps)$ is the list of connected components $Comps$ of the specified graph B. The output is the set of all combinations $Y4$ of the multiset $Comps$ (a list of objects which may contain the same object several times) (see *GAP* manual (16.2.1). The algorithm carries out the following instructions:

```
CombinationsOfConnectedComponents(Comps)
C1 ← Combinations(Comps)
sC1 ← Size(C1)
for q in {1, ..., sC1}
      do L2 ← Concatenation(C1[q])
         U2 ← SSortedList(L2)
         Add L2 to new list Y2 and U2 to new list Y3
```

```
sY3 ← Size(Y3)
for i in {1, ..., sY3}
      do if Y3[i] ≠ ∅
            Add Y3[i] to new list Y4
sY4 ← Size(Y4)
return [Y3, Y4, sY4]
```

3.3.3 `GeneratorsOfSubgroupConj` Function

The input of the function `GeneratorsOfSubgroupConj`(NE, NV, V) is the list NE of all lists of edges of $\Gamma\backslash St(v)$, the list NV of all lists of vertices of $\Gamma\backslash St(v)$, and and the list of vertices V. It computes the list $gens1$ which form the type (1) generators of $Conj(G_\Gamma)$. The algorithm carries out the following instructions:

```
GeneratorsOfSubgroupConj(NE, NV, V)
sNE ← Size(NE)
invV ← ComputeTheInveres(V)
L ← Concatenation(V, invV)
for h in {1, ..., sNE}                    ▷ h ∈ V
      do G2 ← NE(h)
         G1 ← NV(h)
         R3 ← ConnectedComponentsOfGraph(G1, G2)
         Comps ← R3(1)                    ▷ Comps is the list of all components
         sComps ← R3(2)
         R4 ← CombinationsOfConnectedComponents(Comps)
         Y3 ← R4(1)
         Y4 ← R4(2)
         sY4 ← R4(3)
         for i in {1, ..., sY4}
              do diff2 ← Difference(L, Y4[i])
                 Add diff2 to new list xs1
         for i in {1, ..., sY4}
              do sz ← Size(xs1[i])
                 for j in {1, ..., sz}
                      do KK ← Concatenation(Y4[i], [xs1[i][j]])
                         HH ← [KK, xs1[i][j]]
```

```
                        ADD HH to new list Y5
            sY5 ← SIZE(Y5)
            ADD Y5 to new list Y6
ADD xs1 to new list xs2
ADD Bs to new list Y3
sY6 ← SIZE(Y6)
Y7 ← CONCATENATION(Y6)
sY7 ← SIZE(Y7)
xs3 ← CONCATENATION(xs2)
sxs3 ← SIZE(xs3)
for i in {1, . . . , sxs3}
      do ADD the non-empty element of xs3 to new list xs
sxs ← SIZE(xs)
Uxs ← UNION(xs)
Uxs ← SIZE(Uxs)
for i in {1, . . . , sY7}
      do ADD the non-empty element of Y7 to new list CxY1
sCxY1 ← SIZE(CxY1)
for j in {1, . . . , sCxY1}
      do COMPUTE CxY a list of the definitions of the partial conjugations
sCxY ← SIZE(CxY)
Y8 ← CONCATENATION(Bs)
sBs ← SIZE(Bs)
sY8 ← SIZE(Y8)
for i in {1, . . . , sY8}
      do ADD the non-empty element of Y8 to new list Y
sY ← SIZE(Y)
for k in {1, . . . , sCxY}
      do CONSTRUCT a list f such that f(n) = CxY(n), n ∈ N
sf ← SIZE(f)
for j in {1, . . . , sf}
      do ADD f_i the name of the i^th element of f to new list gens1
sgens1 ← SIZE(gens1)
return [CxY, sCxY, Y, sY, f, sf, gens1, sgens1]
```

Remark 3.3.1. The relators on the generators of $Conj(G_\Gamma)$ are represented using sequences of the form $R = [p, \epsilon_1 n_1, \ldots, \epsilon_k n_k]$, where p,ϵ_i,n_i are integers, $\epsilon_i = \pm 1$, $0 \leq p \leq 1$ and $1 \leq n_i$. Each sequence R determines a word W_R, in the generators S, as follows, and R is called the **index** of W_R. If $p = 0$ then the sequence R corresponds to a word $W_R = c_{v,Y} * c_{v^{-1},Y}$ of length 2. For example relators of type $(C1)$ have form $c_{v,Y} * c_{v^{-1},Y} = 1$ and have indices of form $[0, idx1, idx2]$ where $idx1 =$$c_{v,Y}$ and $idx2 =$$c_{v^{-1},Y}$. If $p = 1$ then R corresponds to a word $W_R = w_u * c_{v,Y} * w_u^{-1} * c_{v^{-1},Y}$ of length 4. For example relators of type $(C4)$ have form $w_u * c_{v,Y} * w_u^{-1} * c_{v^{-1},Y} = 1$ if $u \notin Y, v \neq u, u^{-1}$ and have indices of form $[1, idx1, idx2, idx3, idx4]$ where $idx1 =$$w_u$, $idx2 =$$c_{v,Y}$, $idx3 =$$w_u^{-1}$, and $idx4 =$$c_{v^{-1},Y}$. Sequences with $p = 1$ occur only in Section 3.3.7.

3.3.4 `APCGRelationRConj1` Function

The inputs of the function `APCGRelationRConj1`(CxY, Y, f) are CxY the list of the definitions of partial conjugations of $Conj(G_\Gamma)$ defined in Section 3.3.3, Y the list of the non-empty union of connected components of $\Gamma \backslash St(v)$ defined in Section 3.3.3, f the list of the names of the definitions of partial conjugations defined in Section 3.3.3. It computes the list of indices $[0, idx1, idx2]$ of relations of type $(C1)$ and adds each of them to the list $R2a$. In addition it calculates the size of the list $R2a$. It returns $[R2a, sR2a]$.

3.3.5 `APCGRelationRConj2` Function

The inputs of the function `APCGRelationRConj2`$(CxY, Y, Lk, f, R2a)$ are CxY the list of the definitions of partial conjugations of $Conj(G_\Gamma)$ defined in Section 3.3.3, Y the list of the non-empty union of connected components of $\Gamma \backslash St(v)$ defined in Section 3.3.3, the list of links Lk, f the list of the names of the definitions of partial conjugations defined in Section 3.3.3 and the list $R2a$ computed in Section 3.3.4. It computes the list of indices $[0, idx1, idx2, idx3]$ of relations of type $(C2)$ and adds each of them to the list $R2a$ (we replace $R2a$ by [] if we need just the indices $[0, idx1, idx2, idx3]$ of relations of type $(C2)$. In addition it calculates the size of the list $R2a$. It returns $[R2a, sR2a]$.

3.3.6 `APCGRelationRConj3` Function

The inputs of the function `APCGRelationRConj3`$(CxY, Y, Lk, f, R2a)$ are CxY the list of the definitions of partial conjugations of $Conj(G_\Gamma)$ defined in Section 3.3.3, Y the list of the non-empty union of connected components of $\Gamma \backslash St(v)$ defined in Section 3.3.3, the list of links Lk, f the list of the names of the definitions of partial conjugations defined in Section 3.3.3 and the list $R2a$ computed in Section 3.3.5. It computes the list of indices $[0, idx1, idx2, idx3, idx4]$ of relations of type $(C3)$, and adds each of them to the list $R2a$ (we can replace $R2a$ by [] if we need just the indices $[0, idx1, idx2, idx3]$ of relations of type $(C3)$. In addition it calculates the size of the list $R2a$. It returns $[R2a, sR2a]$.

3.3.7 `APCGRelationRConj4` Function

The inputs of the function `APCGRelationRConj4`$(CxY, V, Lk, gens1, Y, f, R2a)$ are CxY the list of the definitions of elementary partial conjugations of $Conj(G_\Gamma)$ defined in Section 3.3.3, the list of vertices V, the list of links Lk, the list $gens1$ from Section 3.3.3, Y the list of the non-empty union of connected components of $\Gamma \backslash St(v)$ defined in Section 3.3.3, f the list of the names of the definitions of partial conjugations defined in Section 3.3.3 and the list $R2a$ computed in Section 3.3.6. Firstly, it computes the list of inner automorphisms W, then $gens4$ the list of the generators of $Conj(G_\Gamma)$. This is the concatenation of the lists $gens1$ and W but; without repeating generators that appear in $gens1$. Secondly, it computes the list of indices $[1, idx1, idx2, idx3, idx4]$ of relations of type $(C4)$, and adds each of them to the list $R2a$ (we can replace $R2a$ by [] if we need just the indices $[1, idx1, idx2, idx3, idx4]$ of relations of type $(C4)$. It returns $[W, gens4, R2a, sW, sgens4, sR2a]$ where sW, $sgens4$ and $sR2a$ are the sizes of W, $gens4$ and $R2a$ respectively.

3.3.8 `APCGConjLastReturn` Function

The inputs of the function `APCGConjLastReturn`$(gens4, R2a, sR2a)$ are the list of generators $gens4$ of the subgroup $Conj(G_\Gamma)$, the list of the indices of the relators $R2a$ and its size $sR2a$. It forms the list of relations $rels$ from the list $R2a$. For each element R of $R2a$ the relator W_R is added to a new list $rels$. It computes the free group F on $gens4$ (defined in Section 3.3.7). Also it computes a finite presentation of the groups $GGG = F/rels$. Finally, it returns the final return $[gens, rels, GGG]$ of the functions `FinitePresentationOfSubgroupConj` below. The algorithm carries out the following instructions:

APCGConjLastReturn($gens4, R2a, sR2a$)
$F \leftarrow$ FreeGroup($gens4$)
$gens \leftarrow$ GeneratorsOfGroup(F)
$sgens \leftarrow$ Size($gens$)
for i **in** $\{1, \ldots, sR2a\}$
    do Form $rels$ the list of relators of the subgroup from $Rels$
$srels \leftarrow$ Size($rels$)
$GGG \leftarrow F/rels$
return $[gens, rels, GGG]$

3.3.9 `FinitePresentationOfSubgroupConj` Function

The function `FinitePresentationOfSubgroupConj`(V, E) provides finite presentation for the subgroup $Conj(G_\Gamma)$. The input of this function is a simple graph $\Gamma = (V, E)$. It returns $[gens, rels, GGG]$. The algorithm carries out the following instructions:

FinitePresentationOfSubgroupConj(V, E)
if Γ is simple graph
    then Call The Function StarLinkOfVertex
        Call The The Function DeleteVerticesFromGraph
        Call The Function GeneratorsOfSubgroupConj
        Call The Function APCGRelationRConj1
        Call The Function APCGRelationRConj2
        Call The Function APCGRelationRConj3
        Call The Function APCGRelationRConj4
        Call The Function APCGConjLastReturn
    else **return** "The graph must be a simple graph"
return $[gens, rels, GGG]$

Where,

(i) $gens$ is a list of free generators of the subgroup $Conj(G_\Gamma)$ of the automorphism group $Aut(G_\Gamma)$ of G_Γ .

(ii) $rels$ is a list of relations in the generators of the free group F. Note that relations are entered as relators, i.e., as words in the generators of the free group.

(iiii) $GGG := F/rels$ is a finitely presented of the subgroup $Conj(G_\Gamma)$ of the automorphism group $Aut(G_\Gamma)$ of G_Γ.

For example:

```
gap> A:=FinitePresentationOfSubgroupConj([1,2,3,4],[[1,2],[3,4]]);
[ [f1, f2, f3, f4, f5, f6, f7, f8], [f1*f4, f2*f3, f3*f2, f4*f1,
f5*f8,f6*f7, f7*f6, f8*f5, f1*f2*f4*f3,f1*f3*f4*f2, f2*f4*f3*f1,
f3*f4*f2*f1, f5*f6*f8*f7, f5*f7*f8*f6, f6*f8*f7*f5, f7*f8*f6*f5,
f2*f1*f3*f4, f3*f1*f2*f4, f1*f2*f4*f3, f4*f2*f1*f3, f1*f3*f4*f2,
f4*f3*f1*f2, f2*f4*f3*f1, f3*f4*f2*f1, f6*f5*f7*f8, f7*f5*f6*f8,
f5*f6*f8*f7, f8*f6*f5*f7, f5*f7*f8*f6, f8*f7*f5*f6, f6*f8*f7*f5,
f7*f8*f6*f5 ], <fp group on the generators [ f1, f2, f3, f4, f5,
f6, f7, f8 ]>]
```

Remark 3.3.2. We can simplify the presentation of $Conj(G_\Gamma)$ above by applying the function TietzeTransformations(G) which is described in Section 2.7.19 as follows:

```
gap> G:=A[3];
<fp group on the generators [ f1, f2, f3, f4, f5, f6, f7, f8 ]>
gap> TietzeTransformations(G);
[ <fp group of size infinity on the generators [ f1, f2, f5, f6 ]>,
[ f1*f2*f1^-1*f2^-1, f5*f6*f5^-1*f6^-1 ] ]
```

Chapter 4

Finite Presentation for the Subgroup $Conj_V$

4.1 Introduction and Background for $Conj_V$

Let Γ be a finite graph and let $G = G_\Gamma$ be the corresponding partially commutative group. Recall that a **basis-conjugating** automorphism is one which maps each canonical generator x to x^{g_x}, for some $g_x \in G$. A presentation for the subgroup of basis-conjugating automorphisms $Conj(G_\Gamma)$ is constructed in [70] as we saw that in Chapter 3. Further subgroups of $Aut(G_\Gamma)$ are discussed in [35], using the notion of admissible subset of a graph, defined as follows. Let $V = V(\Gamma)$ and let $x \in V$. Recall that the **star** of x is $st(x) = \{y \in V : [y, x] = 1\}$. If $Y \subset V$ then the star of Y is $Y^\perp = \cap_{x\in Y} st(x)$. The **closure** of Y is $cl(Y) = \cap_{z\in Y^\perp} st(z)$. For $x \in V$, the **link** of x is $\ell k(x) = st(x)\backslash\{x\}$. The **admissible** set of Y is $\mathfrak{a}(Y) = \cap_{y\in Y}(st(y))^\perp$ and $\mathfrak{a}(x) = \cap_{y\in \ell k(x)} st(y)$.

An element $\phi \in Conj(G)$ is said to be a **Vertex Conjugating automorphism** if, for every element $x \in V$ there exists $f_x \in G$ such that $\phi(y) = y^{f_x}$, for all $y \in [x]$ the equivalence class of the vertex x under the domination equivalence relation. The subgroup of all vertex conjugating automorphism is denoted $Conj_V$.

Our aim in this chapter is to find a finite presentation for the subgroup $Conj_V$ of $Aut(G_\Gamma)$ generated by partial conjugations. Moreover, we develop an algorithm using GAP system that provides a finite presentation for the subgroup $Conj_V$ of $Aut(G_\Gamma)$ with commutative graph Γ. In addition, to find the Tietze transformations of a copy of the presentation of the given finitely presented subgroup $Conj_V$ by using a *GAP* function.

The work in this chapter is motivated by the work of Duncan and Remeslennikov in [35], and we have used terminology and notation of that chapter wherever possible. Note that in some places there are differences between that notation and that of other authors we have followed; in particular [35] has used the terms "conjugation" or "elementary conjugation" to mean "partial conjugation", we may occasionally use those terms too.

Lemma 4.1.1. *[35] For all $x \in V$,*

1. *the set $\mathfrak{a}(x) = \{y \in V : \ell k(x) \subseteq st(y)\}$ and*

2. *$y \in \mathfrak{a}(x)$ if and only if $cl(y) \subseteq \mathfrak{a}(x)$, for all $y \in Y$.*

Proof. (1) $y \in \mathfrak{a}(x)$ if and only if $[y, v] = 1$, for all $v \in \ell k(x)$, if and only if $\ell k(x) \subseteq st(y)$.

(2) For all $y \in V$ we have $y \in cl(y)$, so the "if" clause follows. On the other hand if $y \in \mathfrak{a}(x)$ then, from (i), $\ell k(x) \subseteq st(y)$; so $(st(y))^{\perp} \subseteq (\ell k(x))^{\perp}$, as required. □

Example 4.1.0.1
In the graph Γ of Figure 4.1

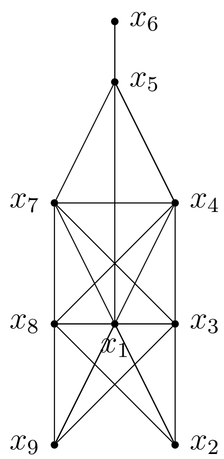

Figure 4.1: A Graph Γ

- $\mathfrak{a}(x_1) = \{x_2, x_3, x_4, x_5, x_7, x_8, x_9\}^{\perp} = \{x_1\} = cl(x_1)$;

- $st(x_4) = st(x_7) = \{x_1, x_3, x_4, x_5, x_7, x_8\}$ and $\mathfrak{a}(x_4) = \mathfrak{a}(x_7) = \{x_1, x_4, x_7\} = cl(x_4) = cl(x_7)$;
- $cl(x_2) = \{x_1, x_2, x_3, x_8\}^{\perp} = \{x_1, x_2\}, cl(x_9) = \{x_1, x_3, x_8, x_9\}^{\perp} = \{x_1, x_9\}$, $\ell k(x_9) = \ell k(x_2)$ and $\mathfrak{a}(x_9) = \mathfrak{a}(x_2) = \{x_1, x_3, x_8\}^{\perp} = \{x_1, x_2, x_4, x_7, x_9\} = cl(x_2) \cup cl(x_4) \cup cl(x_9)$;
- $cl(x_3) = \{x_1, x_3\}, cl(x_8) = \{x_1, x_8\}, \ell k(x_3) = \ell k(x_8)$ and $\mathfrak{a}(x_3) = \mathfrak{a}(x_8) = \{x_1, x_3, x_8\} = cl(x_3) \cup cl(x_8)$;
- $\mathfrak{a}(x_5) = \{x_1, x_4, x_6, x_7\}^{\perp} = \{x_5\} = cl(x_5)$ and
- $cl(x_6) = \{x_5, x_6\}^{\perp} = \{x_5, x_6\}$ and $\mathfrak{a}(x_6) = \{x_5\}^{\perp} = \{x_1, x_4, x_5, x_6, x_7\} = cl(x_4) \cup cl(x_6)$.

For sets U, X we write $U < X$ to indicate that $U \subseteq X$ and $U \neq X$. A subset Y of V is called a **simplex** if the full subgraph of Γ with vertices Y is isomorphic to a complete graph.

Lemma 4.1.2. *[35] For $x \neq z \in X$ and subsets U and X of V the following hold.*

(i) If $U \subseteq X$ then $\mathfrak{a}(X) \subseteq \mathfrak{a}(U)$.

(ii) $\mathfrak{a}(U) \cap \mathfrak{a}(X) = \mathfrak{a}(U \cup X)$.

(iii) $cl(x) = \mathfrak{a}(x) \cap st(x)$ *so* $\mathfrak{a}(x) = cl(x)$ *if and only if* $\mathfrak{a}(x) \subseteq st(x)$.

(iv) $st(x) \subseteq \mathfrak{a}(x)$ *if and only if* $st(x)$ *generates a complete subgraph.*

(v) If $\ell k(x) \subseteq \ell k(z)$ then $\mathfrak{a}(z) \subseteq \mathfrak{a}(x)$.

(vi) If $st(x) \subseteq st(z)$ then $\mathfrak{a}(z) \subseteq \mathfrak{a}(x)$.

(vii) $\mathfrak{a}(z) \subseteq \mathfrak{a}(x)$ *if and only if* $\ell k(x) \subset st(z)$.

(viii) $\mathfrak{a}(x) = \mathfrak{a}(z)$ *if and only if either* $st(x) = st(z)$ *or* $\ell k(x) = \ell k(z)$.

(ix) If $z \in \mathfrak{a}(x)$ then $\mathfrak{a}(z) \subseteq \mathfrak{a}(x)$.

(x) $\mathfrak{a}(U) = \cup_{y \in \mathfrak{a}(U)} \mathfrak{a}(y)$.

(xi) If $cl(x) = \mathfrak{a}(x)$ then $cl(y) = \mathfrak{a}(y)$, *for all* $y \in \mathfrak{a}(x)$.

(xii) If $[x, z] = 1$ *then* $[G(\mathfrak{a}(x)), G(\mathfrak{a}(z))] = 1$.

Proof. Statements (i) to (v) follow directly from the definitions and the fact that if $S \subseteq T$ the $T^\perp \subseteq S^\perp$, for all subsets S, T of X. For (vi) note that in this case $z \in st(x)$, so as $x \neq z$, $\mathfrak{a}(x) = (\ell k(x))^\perp = ((st(x)\backslash\{x, z\}) \cup \{z\})^\perp = (st(x)\backslash\{x, z\})^\perp \cap st(z) \supseteq (st(z)\backslash\{x, z\})^\perp \cap st(x) = \mathfrak{a}(z)$.

The right to left implication of (vii) is a consequence of (v) and (vi), and the fact that if $\ell k(x) \subseteq st(z)$ then $st(x) \subseteq st(z)$ or $\ell k(x) \subseteq \ell k(z)$. To see the opposite implication: if $\mathfrak{a}(z) \subseteq \mathfrak{a}(x)$ then, as $z \in \mathfrak{a}(z)$, we have $z \in \mathfrak{a}(x)$, so $\ell k(x) \subseteq st(z)$, from Lemma 4.1.1.

To see $(viii)$ suppose first that $\mathfrak{a}(x) = \mathfrak{a}(z)$. Then, from (vii), we have $\ell k(x) \subseteq st(z)$ and $\ell k(z) \subseteq st(x)$. If $x \in st(z)$ then $z \in st(x)$, and in this case $st(x) = st(z)$. Otherwise $x \notin st(z)$ and $z \notin st(x)$ in which case $\ell k(x) = \ell k(z)$. Conversely, if either $st(x) = st(z)$ or $\ell k(x) = \ell k(z)$ then it follows, from (v) and (vi), that $\mathfrak{a}(x) = \mathfrak{a}(z)$.

Statement (ix) follows immediately from (vii) and Lemma 4.1.1. Statement (x) follows from (ix) as if $y \in \mathfrak{a}(U)$ then $\mathfrak{a}(y) \subseteq \mathfrak{a}(U)$.

To see statement (xi) observe that $cl(x)$ is a simplex so if $cl(x) = \mathfrak{a}(x)$ and $y \in \mathfrak{a}(x)$ then $\mathfrak{a}(y) \subseteq \mathfrak{a}(x)$ implies that $\mathfrak{a}(y)$ is a simplex. Therefore $\mathfrak{a}(y) \subseteq st(y)$ and the result follows from (iii).

For (xii) suppose that $u \in \mathfrak{a}(x)$ and $v \in \mathfrak{a}(z)$. Since $z \in \ell k(x)$ we have $u \in st(z)$ and similarly $v \in st(x)$. Since $[u, y] = 1$ for all $y \in st(x)$, except possibly x, it follows that u commutes with v, unless $v = x$. However if $v = x$ then, since $v \in (\ell k(z))^\perp$, v commutes with all elements of $st(z)$, including u. $\square$

Remark 4.1.3. Let $\sim_{st}$ be the relation on V given by $x \sim_{st} y$ if and only if $st(x) = st(y)$ and let $\sim_{\ell k}$ be the relation given by $x \sim_{\ell k} y$ if and only if $\ell k(x) = \ell k(y)$. These are equivalence relation and the equivalence classes of x under $\sim_{st}$ and $\sim_{\ell k}$ are denoted by $[x]_{st}$ and $[x]_{\ell k}$, respectively. Note that if $|\, [x]_{st} \,| > 1$ then $[x]_{\ell k} = \{x\}$ and the same is true on interchanging st and ℓk. Therefore the relation $\sim$, given by $x \sim y$ if and only if $x \sim_{st} y$ or $x \sim_{\ell k} y$, is an equivalence relation. Denote the equivalence class of x under $\sim$ by $[x]$. Then $x \sim y$ if and only if $x \sim_{st} y$ or $x \sim_{\ell k} y$, and $[x] = [x]_{st} \cup [x]_{\ell k}$. It follows that $x \sim y$ if and only if $st(x)\backslash\{x, y\} = st(y)\backslash\{x, y\}$.

Example 4.1.0.2

In the graph Γ of Figure 4.2:

$st(x) = \{x, b, e, y, d, l\}$ and $st(y) = \{y, b, e, x, d, l\}$.

So, $st(x) = st(y)$ and $st(x)\backslash\{x, y\} = st(y)\backslash\{x, y\}$. Hence, $x \sim y$.

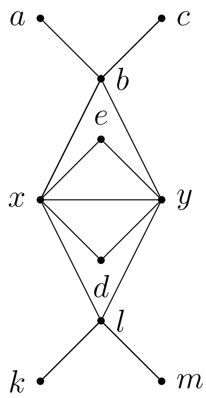

Figure 4.2: Graph of Γ

Definition 4.1.4. [35] Let $x \in V$ and let C be a connected component of the full subgraph $\Gamma \backslash st(x)$

Then the automorphism $\beta_{C,x}$ given by

$$y\beta_{C,x} = \begin{cases} y^x, & if\ y \in C \\ y, & otherwise \end{cases}$$

is called an **aggregate conjugating automorphism**. The subgroup of $Conj(G)$ generated by all aggregate automorphisms is denoted $Conj_A(G)$.

Definition 4.1.5. [35] Let $\mathcal{K} = \mathcal{K}(\Gamma)$ denote the set of admissible subsets of X and define

$$St(\mathcal{K}) = \{\phi \in Aut(G) \mid \phi(G(Y)) = G(Y), \text{for all } Y \in \mathcal{K}\}.$$

$$St^{conj}(\mathcal{K}) = \{\phi \in Aut(G) \mid (G(Y))^{\phi} = G(Y)^{f_Y}, \text{for some } f_Y \in G, \text{for all } Y \in \mathcal{K}\}.$$

Definition 4.1.6. [35] Let $Aut(G)$ be the automorphism group of the partially commutative group G_Γ with commutation graph Γ. An element $\phi \in Aut(G)$ is

(i) a graph automorphism if the restriction $\phi|_X$ of ϕ to X is an element of $Aut(\Gamma)$; and

(ii) a compressed graph automorphism if $\phi|_X$ is an element of $Aut_{comp}(\Gamma)$.

(iii) Denote by $Aut^{\Gamma}(G)$ and $Aut^{\Gamma}_{comp}(G)$ the subgroups of $Aut(G_\Gamma)$ consisting of graph automorphisms and compressed graph automorphisms, respectively.

(iv) For $v \in X$, denote by $S_{[v]}(G)$ the subgroup of $Aut^{\Gamma}(G)$ consisting of elements ϕ such that $\phi|_X \in S_{[v]}$.

(v) Denote by $Aut^{\Gamma}_{symm}(G_{j,*})$ the subgroup of automorphisms ϕ of $Aut(G_{\Gamma})$ such that $\phi|_X$ is an element of $Aut_{symm}(\Gamma_{j,*})$; and

(vi) by $Aut^{\Gamma}_{comm}(G_{j,k})$ the subgroup of automorphisms ϕ such that $\phi|_X$ is an element of $Aut_{comm}(\Gamma_{j,k})$.

Definition 4.1.7. [35] An element $\phi \in Conj(G)$ is said to be a normal conjugating automorphism if, for ever element $x \in V$, there exists $f_x \in G$ such that $\phi(y) = y^{f_x}$, for all $y \in \mathfrak{a}(x)$. The subgroup of all normal conjugating automorphisms is denoted $Conj_N(G)$.

Definition 4.1.8. [35] An elementary conjugating automorphism $\alpha_{C,u}$, where $u = x^{\pm 1}$, for some $x \in V$ is called an **elementary singular conjugating automorphism** if $C = \{y\}$, for some $y \in V$, and the set of all such elementary conjugating automorphisms is denoted $LInn_S = LInn_S(G)$. The subgroup of $Conj(G)$ generated by $LInn_S(G)$ is called **singular** and denoted $Conj_S(G)$.

Definition 4.1.9. Let $Tr_{st} = \{\tau_{V^{\epsilon},y^{\delta}} \in Tr \mid x \in st(y), \epsilon, \delta = \pm 1\}$ and $Tr_{\ell k} = \{\tau_{V^{\epsilon},y^{\delta}} \in Tr \mid x \notin st(y), \epsilon, \delta = \pm 1\}$.

Definition 4.1.10.

- If x and y are vertices of V such that $st(x) \cap st(y) = \ell k(y)$ then we say that x **dominates** y.
- The set of all vertices dominated by x is denoted $Dom(x) = \{u \in V \mid x$ dominates $u\}$.
- The set of all dominated vertices is denoted $Dom(\Gamma) = \cup_{x \in V} Dom(x)$.
- For fixed $y \in V$ the set of all x such that $y \in Dom(x)$ and $[y] \neq [x]$ is the **outer admissible** set of y, denoted $out(y)$.

  From the definition and Lemma 4.1.2 (vii) it follows that x dominates y if and only if $[x, y] \neq 1$ and $\mathfrak{a}(x) \subseteq \mathfrak{a}(y)$. Thus $out(y) = \{x \in \mathfrak{a}(y) : x \notin [y] \cup st(y)\}$.

  If $\alpha_{C,x} \in LInn_S(G)$ then $C = \{y\}$ is a connected component of $\Gamma_{st(x)}$ so $\ell k(y) \subseteq st(x)$ and $y \notin st(x)$. Therefore x dominates y and $\tau_{y,x} \in Tr_{\ell k}$ and $\alpha_{C,x} = \tau_{y,x}\tau_{y^{-1},x}$. Hence $Conj_S$ is the subgroup of $Aut(G_{\Gamma})$ generated by the set $\{\tau_{y,x}\tau_{y^{-1}}, x \mid x \text{ dominates } y\} = LInn_S$.

Definition 4.1.11. [35] Let $x, u \in V$ such that x dominates u and let $[u] \backslash \{x\} = \{v_1, \ldots, v_n\}$. The conjugating automorphism

$$\alpha_{[u],x} = \prod_{i=1}^{n} \alpha_{\{v_i\},x}$$

is called a **basic collected conjugating automorphism** and the set of all basic collected conjugating automorphisms is denoted $LInn_C = LInn_C(G)$. The subgroup of $Conj(G)$ generated by $LInn_C(G)$ is denoted $Conj_C = Conj_C(G)$.

Definition 4.1.12. [35]

- The set of regular elementary conjugating automorphisms is

  $LInn_R = LInn_R(G) = (LInn_G \cap Conj_V(G)) \backslash LInn_S(G)$.

- The set of basic vertex conjugating automorphisms is $LInn_V = LInn(G) = LInn_R(G) \cup LInn_C(G)$.

Not that, an element $\alpha_{y,x} \in LInn_R$ iff

(i) $\mid y \mid \geq 2$; and,

(ii) $\forall\, y \in Y,\ [y] \subseteq Y \cup st(x)$.

Lemma 4.1.13. *[35] Let Γ be a group.*

(i) (a) Γ has an isolated vertex then $Inn = Conj_N$ and

(b) if Γ has no isolated vertex then $Conj_A \leq Conj_N$.

In all cases

$$Inn \leq Conj_A \leq Conj_V \leq Conj$$

and

$$Inn \leq Conj_N \leq Conj_V \leq Conj.$$

(ii) $LInn(V) \leq Conj_V$.

(iii) If $\phi \in Conj_S$ then $\phi(x) = x^{f_x}$, where $v(f_x) \subseteq \mathfrak{a}(x)$, for all $x \in V$.

Proof. (i) It is immediate from the definitions that $Inn \leq Conj_A, Inn \leq Conj_N$ and $Conj_V \leq Conj$. That $Conj_A \leq Conj_V$ follows from the fact that, if

$x, y \in V$ then $[y] \subseteq C \cup x$, for some connected component C of Γ_x. As $[x] \subseteq \mathfrak{a}(x)$, for all x, it follows that $Conj_N \leq Conj_V$.

If x is an isolated vertex then $\mathfrak{a}(x) = X$, so for $\phi \in Conj_N$ there exists $f_x \in G$ such that $\phi(y) = y^{f_x}$, for all $y \in V$. Hence, in this case $Conj_N = Inn$. Assume then that Γ has no isolated vertex. In this case, for all $x \in X$, the connected component of Γ containing x also contains $\mathfrak{a}(x)$. To see that $Conj_A \leq Conj_N$ suppose that $u \in V$ and consider the aggregate conjugating automorphism $\beta = \beta_{C,x}$, where $x \in V$. If $x \in \ell k(u)$ then $v\beta = v$, for all $v \in \mathfrak{a}(u)$, so assume that this is not the case. If $x \in \mathfrak{a}(u)$ then $x \notin \ell k(u)$ implies that $\mathfrak{a}(u) \subseteq C' \cup \{x\}$, for some component C' of Γ_x, so we may also assume that $x \notin \mathfrak{a}(u)$.

Now let v and w be distinct elements of $\mathfrak{a}(u)$ and r be any element of $\ell k(u)$. Then the path v, r, w does not contain x; so v and w are either both in C or both outside C. Hence $\beta_{C,x}$ either fixes every element of $\mathfrak{a}(u)$, or acts as conjugation by x on every element of $\mathfrak{a}(u)$. Thus all elements of $Conj_A$ are normal, as required.

(ii) Follow directly from the definition and the fact that the sets $[x]$ partition X, so that $LInn_C \subseteq Conj_V$.

(iii) An induction on the length of ϕ as a word in the generators $LInn_S$ is used. If ϕ is trivial there is nothing to be proved, so assume inductively that the result holds for words of length at most $n-1$ and that $\phi = \phi_0\phi_1$, where ϕ_0 has length $n-1$ as a word in $LInn_S^{\pm 1}$ and $\phi_1 \in LInn_S^{\pm 1}$, say $\phi_1 = \alpha_{C,z}$, for some $z \in V^{\pm 1}$ and $C = \{y\}$. Then $\phi_0(x) = x^{f_x}$, where $\nu(f_x) \subseteq \mathfrak{a}(x)$, for all $x \in V$. Let $x \in X$ and $u \in \mathfrak{a}(x)^{\pm 1}$. Then $\phi_1(u) = u$ unless $u = y^{\pm 1}$. In the latter case $y \in \mathfrak{a}(x)$ so $z \in \mathfrak{a}(y)^{\pm 1} \subseteq \mathfrak{a}(x)^{\pm 1}$ and $\phi_1(u) = u^z$ implies $\nu(\phi_1(u)) \subseteq \mathfrak{a}(x)$. Thus we have $\nu(\phi_1(f_x)) \subseteq \mathfrak{a}(x)$. Now $\phi(x) = (\phi_1(x))^{\phi_1(f_x)}$ and since $\phi_1(x) = x^z$ if and only if $x = y^{\pm 1}$ it follows that $\nu(\phi(x)) \subseteq \mathfrak{a}(x)$, as required.

$\square$

Definition 4.1.14. [51] Let ϕ be a conjugating automorphism and for each $x \in V$ let $g_x \in G$ be such that $\phi(x) = g_x^{-1} \circ x \circ g_x$. The length $\mid \phi \mid$ of ϕ is $\sum_{x \in X} lg(g_x)$.

Lemma 4.1.15. *([51] [Lemma 2.5 and Lemma 2.8]). Let ϕ be a non-trivial element of $Conj$ and, for each $x \in V$, let $g_x \in G$ such that $\phi(x) = g_x^{-1} \circ x \circ g_x$. Then* *(i) there exist $x, y \in V$ and $\epsilon \in \{\pm 1\}$ such that $x^\epsilon g_x$ is a right divisor of g_y, and*

(ii) if $y, z \in V \backslash st(x)$ such that $[y, z] = 1$ and $x^\epsilon g_x$ is a right divisor of g_y then $x^\epsilon g_x$ is a right divisor of g_z.

(As can be seen from the example $\phi = \alpha_{C,x}^{-1}$ the variable ϵ taking values ± 1 is a necessary part of the lemma.)

Lemma 4.1.16. *[35] Let $\phi \in Conj_V$ and for each $y \in V$ let $g_y \in G$ be such that $\phi(y) = g_y^{-1} \circ y \circ g_y$.*

(i) If $[y] = [y]_{st}$ then $g_u = g_y$, for all $u \in [y]$.

(ii) If $[y] = [y]_{\ell k}$ and $\mid [y] \mid \geq 2$ then there exist $v \in [y]$ and $m_y \in \mathbb{Z}$ such that $g_u = v^{m_y} \circ g_v$, for all $u \in [y] \backslash \{v\}$. Moreover if $m_y \neq 0$ then v is the unique element of $[y]$ with this property and, setting $\epsilon = -m_y / \mid m_y \mid$, $S = [y] \backslash \{v\}$ and $\alpha = \prod_{u \in S} \alpha_{\{u\}, v^\epsilon}$ we have $\alpha \in LInn_C^{\pm 1}$ and $\mid \alpha\phi \mid < \mid \phi \mid$.

Proof. Since $\phi \in Conj_V$, for all $y \in V$, there exists $f_y \in G$ such that $\phi(u) = u^{f_y}$, for all $u \in [y]$, and we may choose an f_y of minimal length with this property. Fix $y \in V$. Then $u^{f_u} = \phi(u) = u^{g_u}$ so $g_u f_y^{-1} \in C_G(u)$, for all $u \in [y]$. Therefore there are $a, b, c \in G$ such that $g_u = a \circ b$, $f_y = c \circ b$ and $g_u f_y^{-1} = a \circ c^{-1} \in C_G(u)$. As g_u has no left divisor in $C_G(u)$ this means that $a = 1$ and so $f_y = c_u \circ g_u$, for $c = c_u \in C_G(u)$.

If $[y] = [y]_{st}$ then $C_G(u) = C_G(y)$, for all $u \in [y]$, so in this case $g_y = f_y = g_u$, for all $u \in [y]$.

Assume then that $[y] = [y]_{\ell k}$, with $\mid [y] \mid \geq 2$, and let $u, v \in [y], v \neq u$, so $[u, v] \neq 1$. Suppose $v \in v(f_y)$. Then $f_y = c_v \circ g_v = c'_v \circ v^m \circ g_v$, where $c'_v G(\ell k(v))$ and $m \in \mathbb{Z}$. Then $u^{f_y} = u^{v^m g_v}$, since $\ell k(v) = \ell k(u)$. As g_v has no left divisor in $C_G(v)$ and $[v, u] \neq 1$ we have $u^{v^m g_v} = g_v^{-1} \circ v^{-m} \circ u \circ v^m \circ g_v$, so $g_u = v^m \circ g_v$. By choice of f_y we have $c'_v = 1$, and if $m \neq 0$ then no element $u \in [y], u \neq v$, can be a left divisor of $v^m \circ g_v$, so the first statement of (ii) as well as the uniqueness of v follow. Moreover v dominates u, for all $u \in [y]$, so the final statement of (ii) also holds. □

Proposition 4.1.17. *[35] $Conj_V$ is generated by $LInn_V = LInn_R \cup LInn_C$.*

Proof. Note that, from Lemma 4.1.13 (ii) we have that $\langle LInn_V \rangle \leq Conj_V$. So we need to prove the opposite inclusion; $Conj_V \leq \langle LInn_V \rangle$. Suppose that $\phi \in Conj_V$ be an automorphism. By using the induction on the length of ϕ we will do this direction. Assume that $\mid \phi \mid= k$, so if $\mid \phi \mid= 0$ then $\phi = 1$ and there is nothing to prove. Hence, suppose $k > 1$ and assume that if $\varphi \in Conj_V$ with $\mid \varphi \mid< k$ then $\varphi \in \langle LInn_V \rangle$ (by induction assumption). If there exists $y \in V$ such that, $[y] = [y]_{\ell k}$, $\mid [y] \mid\geq 2$ and by using Lemma 4.1.16, suppose $m_y \neq 0$. Set $\alpha = \Pi_{u=y,y_2,\dots,y_n}\alpha_{\{u\},v^\epsilon} \in LInn_C(\epsilon = 1$ if $m < 0$ and $\epsilon = -1$ if $m > 0)$ and $\mid \alpha\phi \mid<\mid \phi \mid$. We have $\phi = \alpha^{-1}\alpha\phi$. Now, $\alpha \in LInn_C$, so $\alpha^{-1} \in \langle LInn_C\rangle \subseteq \langle LInn_V\rangle$. As $\phi \in Conj_V$ and $\alpha \in LInn_C \subseteq Conj_V$ we have $\alpha\phi \in Conj_V$. Write $\alpha\phi = \psi \in Conj_V$, with $\mid \psi \mid<\mid \phi \mid$; so by the assumption of induction we have that $\psi \in \langle LInn_V\rangle$ which implies that $\alpha^{-1}\psi \in \langle LInn_V\rangle$, so $\phi \in \langle LInn_V\rangle$, as claimed.

Hence we assume that either $[y] = [y]_{st}$ or $m_y = 0$, and so $g_y = g_u$, for all $u \in [y]$ and for all $y \in V$. From Lemma 4.1.15(i) there exist $x, y \in V, \epsilon \in \{\pm 1\}$ such that $\phi(x) = g_x^{-1} \circ x \circ g_x, \phi(y) = g_y^{-1} \circ y \circ g_y$ and $x^\epsilon g_x$ is a right divisor of g_y. Suppose that $[x, y] = 1$. Then $[\phi(x), \phi(y)] = 1$; that is $[x^{g_x}, y^{g_y}] = 1$. If $g_y = a \circ x^\epsilon \circ g_x$, for some $a \in G$, then this implies that $[x, y^{ax^\epsilon}] = 1$, from which it follows that $[x, a] = 1$. However, in this case y^{g_y} is not reduced, a contradiction. Therefore $y \notin st(x)$, and so $u \notin st(x)$, for all $u \in [y]$.

Let $[y] = \{v_1, \dots, v_r\}$ and let $C_1, \dots, C_s$ be the components of $\Gamma_{st(x)}$ containing $v_1, \dots, v_r$. Then, from Lemma 4.1.15(ii), $x^\epsilon g_x$ is a right divisor of g_c for all $c \in C_1 \cup \ldots \cup C_s$. Let $\alpha = \prod_{i=1}^s \alpha_{C_i,x^{-\epsilon}}$. Then $\mid \phi(x) \mid<\mid \phi \mid$. We claim that $\alpha \in Conj_V$. Suppose not, so there is some $z \in V$ and elements $u, v \in [z]$ such that $u \in C_i$, for some i, but $v \notin \cup_{i=1}^s C_i \cup \{st(x)\}$. This implies that $\ell k(u) = \ell k(v) \subseteq st(x)$ and, as $u \in C_i$ implies $x \notin st(u)$, so x dominates u. Then $C_i = \{u\}$ so $u \in [y]$ and $[z] = [y] \subseteq \cup_{i=1}^s C_i$, a contradiction. Thus no such z exists and $\alpha \in Conj_V$.

If $s = 1$ and $\mid C_1 \mid\geq 2$ then $\alpha \in LInn_R^{\pm 1}$. If $s = 1$ and $\mid C_1 \mid= 1$ then x dominates y and $\alpha \in LInn_C^{\pm 1}$. If $s > 1$ then $st(x) \supseteq \ell k(y)$ and x dominates every element of $[y]$. In this case $\alpha \in LInn_C^{\pm 1}$ again. Hence by induction $\phi \in \langle LInn_R \cup LInn_C\rangle$. □

4.2 Whitehead Automorphisms and Day's Relations

If (A, a) is a Whitehead automorphism which is a partial conjugation automorphism then for each $y \in X$ either y is mapped to y^a or y is fixed. Thus for all $y \in V$ with $y \neq a^{\pm 1}$, either y and y^{-1} belong to A or $\{y, y^{-1}\} \cap A = \emptyset$. Thus, for such Whitehead automorphisms we can write $A = C \cup C^{-1} \cup \{a\}$ where $C \subseteq V$ and

$a^{\pm 1} \notin C$. Moreover, we may assume that $A \cap \ell k_L(a) = \emptyset$, since if $y \in st_L(a)$ then $y^a = y$. As (A, a) induces an automorphism of G, it follows now that C is a union of vertices of connected components of $\Gamma \backslash st(a)$. Suppose that $\Gamma \backslash st(a)$ has connected components $C_1, \ldots C_n$ and $C = \cup_{i \in T} C_i$, where T is a non-empty subset of $\{1 \ldots n\}$. Then from the union of these connected components above we define $\alpha_{C,a} = (A, a)$ so

$$\alpha_{C,a}(v) = \begin{cases} v^a & if\ v \in C \\ v & otherwise. \end{cases}$$

On the other hand for $y \in V$, if $x_1, \ldots, x_r$ are such that $\ell k(x_i) \subseteq st(y)$, let $D = \{x_1, \ldots, x_r\}$ and we define that $\tau_{D,y} = \tau_{x_1,y} \circ \ldots \circ \tau_{x_r,y}$. Then, written as a Whitehead automorphism $\tau_{D,y}$ is $(D \cup \{y\}, y)$. Conversely, if (A, a) is a Whitehead automorphism, and for all $x \in V \backslash \{a\}$ we have $x \in A$ if and only if $x^{-1} \notin A$ then setting $D = A \backslash \{a\}$ we have $(A, a) = \tau_{D,a}$.

Now in general if (A, a) is a Whitehead automorphism then let $C_0 = \{x \in A \backslash \{a\} : x^{-1} \notin A \backslash \{a\}\}$ and let $C_1 = \{x \in V : x \in A\ and\ x^{-1} \in A\}$. Then $\tau_{C_0,a}$ is an automorphism and $\alpha_{C_1,a}$ is an automorphism and $(A, a) = \tau_{C_0,a}\alpha_{C_1,a}$ (and $\tau_{C_0,a}\alpha_{C_1,a} = \alpha_{C_1,a}\tau_{C_0,a}$).

We now translate relations (R1) to (R10) of Day, from the terminology of Whitehead automorphisms to the terminology used here.

Let $\alpha = (A, a)$ and $\beta = (B, b)$ be a Whitehead automorphisms and write $\alpha = \tau_{C_0,a}\alpha_{C_1,a}$ and $\beta = \tau_{D_0,b}\alpha_{D_1,b}$ where $C_0 \cap C_1 = \emptyset$ with $A \backslash \{a\} = C_0 \cup C_1 \cup C_1^{-1}$ and $D_0 \cap D_1 = \emptyset$ with $B \backslash \{b\} = D_0 \cup D_1 \cup D_1^{-1}$ respectively and $C_1, D_1 \subseteq V$, and $C_0 \cap C_0^{-1} = D_0 \cap D_0^{-1} = \emptyset$.

In the following relations (R1) to (R10) when we consider sets A_0 and A_1 we always assume $A_0 \cap A_1 = \emptyset$ (and similarly for B_0, B_1, or C_0, C_1, etc, and we assume all automorphisms $\alpha_{A_1,a}$, $\tau_{A_0,a}$ mentioned, are well defined.) Now we can replace (A, a) in each of (R1) to (R10) in Section 2.5 of Chapter two by $\tau_{C_0,a}\alpha_{C_1,a}$, with $C_0 \cap C_1 = \emptyset$ and $A \backslash \{a\} = C_0 \cup C_1 \cup C_1^{-1}$, such that $\tau_{C_0,a}$ is one of $\tau_{D,y}$ and $\alpha_{C_1,a}$ is one of $\alpha_{C,a}(v)$ as defined above. Therefore,

(R1) $(\tau_{C_0,a}\alpha_{C_1,a})^{-1} = \tau_{C_0,a^{-1}}\alpha_{C_1,a^{-1}}$, where $\tau_{C_0,a}, \alpha_{C_1,a}$ are of type (2) Whitehead automorphisms.

(R2) $(\tau_{C_0,a}\alpha_{C_1,a})(\tau_{D_0,a}\alpha_{D_1,a}) = \tau_{C_0 \cup D_0,a}\alpha_{C_1 \cup D_1,a}$ when $(C_0 \cup D_0) \cap (C_1 \cup D_1) = \emptyset$.

(R3) $(\tau_{C_0,a}\alpha_{C_1,a})(\tau_{D_0,b}\alpha_{D_1,b}) = (\tau_{D_0,b}\alpha_{D_1,b})(\tau_{C_0,a}\alpha_{C_1,a})$ if $v(a) \notin (D_0 \cup D_1)$, $v(b) \notin (C_0 \cup C_1)$, $a \neq b, b^{-1}$ and at least one of (a) $(C_0 \cup C_1) \cap (D_0 \cup D_1) = \emptyset$ or (b) $b \in \ell k_L(a)$ holds. We refer to this relation as (R3a) if condition (a) holds and (R3b) if condition (b) holds.

(R4) $(\tau_{D_0,b}\alpha_{D_1,b})(\tau_{C_0,a}\alpha_{C_1,a})(\tau_{D_0,b}\alpha_{D_1,b})^{-1} = (\tau_{C_0,a}\alpha_{C_1,a})(\tau_{D_0,a}\alpha_{D_1,a})$, such that $a, a^{-1} \notin D_0 \cup D_1$, $b^{-1} \in C_0$ and at least one of (a) $(C_0 \cup C_1) \cap (D_0 \cup D_1) = \emptyset$ or (b) $b \in \ell k_L(a)$. We refer to this relation as (R4a) if condition (a) holds and (R4b) if condition (b) holds.

(R5) $(\tau_{C_0',b}\alpha_{C_1,b})(\tau_{C_0,a}\alpha_{C_1,a}) = (\tau_{C_0'',a}\alpha_{C_1,a})\pi_{a,b}$ where $C_0' = C_0 \cup \{a^{-1}\}$ and $C_0'' = (C_0 \backslash \{b\}) \cup \{b^{-1}\}$ such that $b \in C_0$, $b^{-1} \notin C_0$ with $a \neq b$ and $b \sim a$, where $\pi \in Aut(G_\Gamma)$ with $\pi_{a,b}(a) = b^{-1}$, $\pi_{a,b}(b) = a$ and which fixes the other generators.

(R6) $\pi(\tau_{C_0,a}\alpha_{C_1,a})\pi^{-1} = \tau_{\pi(C_0),\pi(a)}\alpha_{\pi(C_1),\pi(a)}$ for $\pi \in Aut(G_\Gamma)$ which is a graph automorphism.

(R7) The entire multiplication table of the type (1) Whitehead automorphisms, which forms a finite subgroup of $Aut(G_\Gamma)$.

Note that $L\backslash\{a^{-1}\} = (V \cup V^{-1})\backslash\{a^{-1}\} = (V\backslash st_V(a))^{\pm 1} = D$, so $(L\backslash\{a^{-1}\}, a)$ corresponds to $\alpha_{D,a}$. But, if $D = (V\backslash st_V(a)$ then $\alpha_{D,a} =$ inner automorphism of conjugation by a say (γ_a). Hence the relations (R8) to (R10) are that:

(R8) $(\tau_{C_0,a}\alpha_{C_1,a}) = \gamma_a(\tau_{E_0,a^{-1}}\alpha_{E_1,a^{-1}})$ where $\tau_{C_0,a}, \alpha_{C_1,a}$ are of type (2) Whitehead automorphisms, and $E_1 = V\backslash[C_1 \cup C_0 \cup C_0^{-1} \cup st_V(v(a))]$ with $E_0 = C_0^{-1}$ and $\gamma_a = \alpha_{V\backslash st_V(v(a)),a}$.

(R9) $(\tau_{C_0,a}\alpha_{C_1,a})\gamma_b = \gamma_b(\tau_{C_0,a}\alpha_{C_1,a})$ if $b \in L$ with $b, b^{-1} \notin C_0 \cup C_1$ and $\gamma_b = \alpha_{V\backslash st_V(v(b)),b}$.

(R10) $(\tau_{C_0,a}\alpha_{C_1,a})\gamma_b = \gamma_a\gamma_b(\tau_{C_0,a}\alpha_{C_1,a})$ if $b \in C_0$ such that $\gamma_a = \alpha_{V\backslash st_V(v(a)),a}$ and $\gamma_b = \alpha_{V\backslash st_V(v(b)),b}$.

4.3 A Presentation for $Conj_V$

Note that, if $(A, a) \in Conj_V$ then we have $A_0 = \emptyset$ and $A = A_1 \cup A_1^{-1} \cup \{a\}$. Moreover, as above since (A, a) is a partial conjugation we may assume $A \cap \ell k_L(a) = \emptyset$ so also $A_1 \cap \ell k_L(a) = \emptyset$. So (A, a) can be written as $\alpha_{C,a}$ where $C = A_1$.

In [35] it is shown that $Conj_V$ is generated by a set called $LInn_V$ as we saw in Section 4.1. Here we use different generators which are more convenient. If we use Whitehead automorphisms we need to combine them. So we could have $\alpha_{C,x} \in LInn_R$ (which is already a Whitehead automorphism), and $\beta = \prod_{y\in[u]\setminus\{x\}} \alpha_{\{y\},x} \in LInn_C$, where $[u]$ is an equivalent class of u for all $u \in V$, which is also a Whitehead automorphism. After we combine them we will get a new generator $\alpha_{Z,x} = \alpha_{C,x}\,\beta \in Conj_V$ which is also a Whitehead automorphism and one of Toinet's generators. For example, consider the graph of Γ of Figure 4.3.

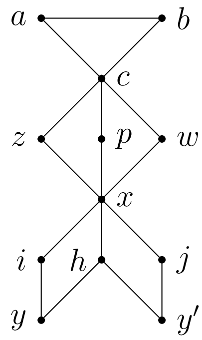

Figure 4.3: Graph of Γ

So, we have $[y] = \{y, \acute{y}\}$, $\beta = \alpha_{\{y\},x}\alpha_{\{\acute{y}\},x}$ and $[c] = \{c\}$ and $[a] = \{a, b\}$. The subgraph $\Gamma\setminus st(x)$ is shown in Figure 4.4.

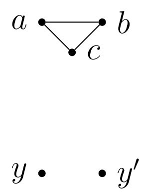

Figure 4.4: Subgraph $\Gamma\setminus st(x)$

Set $Y = \{a, b, c\}$ then $\alpha_{Y,x} \in LInn_R$. Also setting $Z = \{a, b, c, y, \acute{y}\}$ then $\alpha_{Z,x} = \alpha_{Y,x}\beta \in Conj_V$. It is a Whitehead automorphism and one of Toinet's generators.

Therefore, we want a generating set for $Conj_V$ consisting of elements that belong to Toinet's generating set for $Conj$. To this end, we make the following definition.

Definition 4.3.1. Define W_V to be the set of partial conjugations $\alpha_{C,x}$, where $x \in L = V \cup V^{-1}$ and (as well as being a union of connected components of $\Gamma\setminus st(x)$) the set C satisfies the condition that, for all $z \in V$ either

(i) $[z] \cap C = \phi$; *or*

(ii) $[z] \subseteq C \cup st(x)$. (1)

Lemma 4.3.2. *The following two properties hold on W_V:*

(a) Every element of W_V belongs to $Conj_V$ and

(b) $LInn_V \subseteq W_V$.

Proof. (a) Note that, $\alpha_{C,x} \in Conj_V \Leftrightarrow \forall z \in V\ \exists\ g_z$ such that $u\alpha_{C,x} = u^{g_z}\ \forall\ u \in [z]$. But,

$$z\alpha_{C,x} = \begin{cases} z^x = x^{-1}zx & if\ z \in C \\ z & if\ z \notin C, \end{cases}$$

for each $z \in Z$.

If $\alpha_{C,x} \in W_V$ then suppose $z \in Z$. By definition of W_V either (i) or (ii) of (1) holds. If (i) holds then, for each $u \in [z]$ we have $u \notin C$ so $u\alpha_{C,x} = u$. If (b) holds then either, $u \in C$ and hence $u\alpha_{C,x} = u^x$, or $u \in st(x)$, so $u\alpha_{C,x} = u = u^x$ because $[u, x] = 1$. So in both cases $u\alpha_{C,x} = u^x$ and we have $u\alpha_{C,x} = u^x$ for all $u \in [z]$. This means $\alpha_{C,x} \in Conj_V$. Hence, every element of W_V belongs to $Conj_V$.

(b) Let $\beta_{C,x} \in LInn_V$. This implies that $\beta_{C,x} \in LInn_R$ or $\beta_{C,x} \in LInn_C$. (Since $LInn_V = LInn_R \cup LInn_C$). Note that, if $\beta_{C,x} \in LInn_R$ then we have that

(a) $|C| \geq 2$ and

(b) $\forall y \in C, [y] \subseteq C \cup st(x)$ (def. of $LInn_R$).

Thus, $\beta_{C,x} \in W_V$ (since $\beta_{C,x}$ satisfies the conditions of W_V). Hence, $LInn_R \subseteq W_V$. If $\beta_{C,x} \in LInn_C$ then $\beta_{C,x}$ is a basic collected conjugating automorphism (by def. of $LInn_C$). This implies that for some $x, z \in L$ we have x dominates z (i.e., $\ell k(z) \subseteq st(x)$ and $z \notin st(x)$) and $[z]\backslash\{x\} = \{\vartheta_1, \ldots, \vartheta_n\}$ with $\beta_{C,x} = \prod_{i=1} \beta_{\{\vartheta_i\},x} \in LInn_C$. So $\beta_{C,x} = \alpha_{C,x}$ where $C = \{\vartheta_1, \ldots, \vartheta_n\}$.

Now (i) if $u \in V$ and $[u] \cap C \neq \phi$ then $\vartheta_i \in [u]$, for some i so $[u] = [\vartheta_i] = [z]$ so $[u] \subseteq C \cup \{x\} \subseteq C \cup st(x)$, so the second condition of W_V holds. On the other hand if (ii) $u \in V$ and $[u] \cap C = \phi$ then the first condition of W_V holds. So in all cases either the first or the second condition of W_V holds. This implies $\beta_{C,x} \in W_V$. Hence, $LInn_V \subseteq W_V$. Therefore, $LInn_V = LInn_R \cup LInn_C \subseteq W_V$.

□

Lemma 4.3.3. *If* $\alpha_{C,x} \in W_V$ and $D = V \backslash (C \cup st(x))$ then $\alpha_{D,x^\epsilon} \in W_V$ for $\epsilon = \pm 1$.

Proof. To prove this it is necessary only to check that condition (1) on C above holds when C is replaced by D. First note that, for all $z \in V$, either $[z] \cap C = \phi$; or $[z] \subseteq C \cup st(x)$, by definition of W_V.

(i) To show that if $[z] \subseteq C \cup st(x)$ then $[z] \cap D = \phi$.

$$\begin{aligned}
[z] \cap D &= [z] \cap (V \backslash (C \cup st(x))) \\
&= [z] \cap (V \cap (C \cup st(x))^c) \quad (\text{since } A \backslash B = A \cap B^c) \\
&= ([z] \cap (C \cup st(x))^c) \cap V \\
&= \phi \cap V \quad (\text{since we have that } [z] \subseteq C \cup st(x) \text{ which implies that} \\
&\qquad\qquad [z] \cap (C \cup st(x))^c = \phi\ ) \\
&= \phi.
\end{aligned}$$

(ii) To show that if $[z] \cap C = \phi$ then $[z] \subseteq D \cup st(x) = V \backslash (C \cup st(x)) \cup st(x)$.

Note that, by assumption $[z] \cap C = \phi$, so if $u \in [z]$ then $u \in V \backslash C$ and if also $u \notin st(x)$ then $u \in V \backslash (C \cup st(x)) = D$. Hence $[z] \subseteq D \cup st(x)$.

□

Given $\alpha = (A, a)$ Day defines $\bar{\alpha} = (A', a^{-1})$, where $A' = L \backslash (A \cup \ell k_L(a))$. In our terminology, $\bar{\alpha} = \tau_{A'_0, a^{-1}} \alpha_{A'_1, a^{-1}}$ where $A'_0 = \{x \in A' \backslash \{a^{-1}\} : x^{-1} \notin A' \backslash \{a^{-1}\}\} = \{x^{-1} \in V^{\pm 1} : x \in A_0\}$ and $A'_1 = \{x \in V : x \in A' \textit{ and } x^{-1} \in A'\} = \{x \in V : x^{\pm 1} \notin A'_0, x \notin st_L(a) \textit{ and } x \notin A'_1\}$.

In the case of $(A, a) \in W_V$ we have $A_0 = \emptyset$ and $A = A_1 \cup A_1^{-1} \cup \{a\}$. In this case if $\alpha = \alpha_{C,x}$ then $\bar{\alpha} = \alpha_{D,x^{-1}}$, where $D = V \backslash (C \cup st_L(x)) = \{y \in V : y \notin C \cup st_L(x)\}$.

Lemma 4.3.4. *If* $\pi \in Aut(\Gamma)$ and $\alpha_{C,x} \in W_V$ then $\alpha_{\pi(C),\pi(x)} \in W_V$.

Proof. Let $\pi \in Aut(\Gamma)$ and $\alpha_{C,x} \in W_V$. Note that, to show $\alpha_{\pi(C),\pi(x)} \in W_V$ we need only to check the condition (1) on C holds when C is replaced by $\pi(C)$ and x is replaced by $\pi(x)$.

Suppose $z \in V$. We show that either $[z] \cap \pi(C) = \phi$ or $[z] \subseteq \pi(C) \cup st(\pi(x))$. As $\pi \in Aut(\Gamma)$ there exists $y \in V$ such that $\pi(y) = z$. Suppose that $[z] \cap \pi(C) \neq \phi$; and let $u \in [z] \cap \pi(C)$. Since $\pi \restriction_V$ is a graph automorphism we have $\pi[a] = [\pi(a)]$, for each $a \in V$. Hence $[z] = [\pi(y)] = \pi[y]$. Now $u = \pi(v)$ where $v \in C$, since $u \in \pi(C)$, so $\pi(v) \in [z] = \pi[y]$. Thus $\pi(v) = \pi(v')$, for some $v' \in [y]$ and since π is one-one this implies $v = v'$; that is $v \in C$ and $v \in [y]$ so $v \in [y] \cap C$. But, since $\alpha_{C,x} \in W_V$ we have $[y] \cap C = \phi$ or $[y] \subseteq C \cup st(x)$. Hence, as $v \in [y] \cap C$ we have $[y] \subseteq C \cup st(x)$. Hence $\pi[y] \subseteq \pi(C) \cup \pi(st(x))$ which implies that $[z] \subseteq \pi(C) \cup st(\pi(x))$ (as $st(\pi(x)) = \pi(st(x))$). Therefore, either $[z] \cap \pi(C) = \phi$ or $[z] \subseteq \pi(C) \cup st(\pi(x))$. This implies that $\alpha_{\pi(C),\pi(x)} \in W_V$. □

Lemma 4.3.5. *If $\alpha_{C,x}, \alpha_{D,x} \in W_V$ then $\alpha_{C\cap D,x} \in W_V$.*

Proof. Note that, to prove this it is necessary only to check that condition (1) on C above holds when C is replaced by $C \cap D$. Now fix $\alpha_{C,x}, \alpha_{D,x} \in W_V$ and let $z \in Z$.

If $[z] \cap C = \phi$ then $[z] \cap (C \cap D) = ([z] \cap C) \cap D = \phi \cap D = \phi$. Similarly if $[z] \cap D = \phi$ then $[z] \cap (C \cap D) = \phi$.

Hence we may assume that $[z] \subseteq C \cup st(x)$ and $[z] \subseteq D \cup st(x)$. Note that, $(C \cap D) \cup st(x) = (C \cup st(x)) \cap (D \cup st(x))$ (distributive laws). But, $[z] \subseteq C \cup st(x)$ and $[z] \subseteq D \cup st(x)$ by assumption. This implies that $[z] \subseteq (C \cap st(x)) \cap (D \cup st(x))$. i.e., $[z] \subseteq (C \cap D) \cup st(x)$. Hence, $\alpha_{C\cap D,x} \in W_V$. □

Lemma 4.3.6. *Let $\alpha_{C,x}, \alpha_{D,x} \in W_V$ and let $D' = V \backslash (D \cup st(y))$ such that $y^{\pm 1} \notin C$. If $\alpha_{C\cap D',x}$ is a well defined automorphism then it belongs to W_V.*

Proof. Note that, $\alpha_{C\cap D',x}$ is a well defined automorphism if and only if $C \cap D'$ is a union of connected components of $\Gamma \backslash st(x)$. Now suppose $\alpha_{C\cap D',x}$ is a well defined automorphism. So we need to show that $\alpha_{C\cap D',x} \in W_V$.

If $[z] \cap C = \phi$ then $[z] \cap (C \cap D') = \phi$ (as in previous lemma), so we assume $[z] \subseteq C \cup st(x)$. Therefore, there are two possibilities:

(i) $[z] \cap D = \phi$; or

(ii) $[z] \subseteq D \cup st(y)$.

If $[z] \cap D' = \phi$ then $[z] \cap (C \cap D') = \phi$ so we assume there exists $u \in [z] \cap D'$. We need to show $[z] \subseteq D' \cup st(x)$:

Case (i) If $[z] \cap D = \phi$ then suppose there exists $v \in [z]$ with $v \in st(y)$. As $v \sim z$ either (a) $st(z) = st(v)$ or (b) $\ell k(z) = \ell k(v)$. In case (a) we have $v \in st(y)$ implies $y \in st(v) = st(z)$ implies $[z] \subseteq st(y)$ so $u \notin D'$, a contradiction.

If (b), $\ell k(z) = \ell k(v)$ then if $y \in \ell k(v)$ with $y \neq v$, as above we obtain $y \in \ell k(z)$ and $z \in st(y)$ implies $[z] \in st(y)$. Hence in case (b) we must have $y = v$.

Note we assume that $[z] \subseteq C \cup st(x)$ and $y \notin C$ (as $y^{\pm 1} \notin C$) so we must have $y \in st(x)$. Hence in this case $v = y \in D' \cup st(x)$. On the other hand if $v \in [z]$ and $v \notin st(y)$ then $v \notin D \cup st(y)$ so $v \in D$; so that $[z] \subseteq D' \cup st(x)$ in this case.

Case (ii) We assume that $[z] \subseteq D \cup st(y)$. We show $[z] \cap D' = \phi$. Note that,

$$\begin{aligned} D' \cap [z] &= [V \backslash (D \cup st(y)] \cap [z] \\ &= (V \cap [z]) \backslash (D \cup st(y)) \quad (\text{since } (B \backslash A) \cap C = (B \cap C) \backslash A) \\ &= [z] \backslash (D \cup st(y)) \\ &= \phi \quad (\text{since } [z] \subseteq D \cup st(y) (\text{by assumption}). \end{aligned}$$

Hence, $\alpha_{C \cap D', x} \in W_V$. □

Lemma 4.3.7. *If* $\alpha_{C,x}, \alpha_{D,x} \in W_V$ with $x \in L$. Then $\alpha_{C \cup D,x} \in W_V$.

Proof. Note that, to prove this it is necessary only to check that if $z \in V$ then either $[z] \cap (C \cup D) = \phi$ or $[z] \subseteq (C \cup D) \cup st(x)$.

Suppose that $[z] \cap (C \cup D) \neq \phi$. We have $[z] \cap (C \cup D) = ([z] \cap C) \cup ([z] \cap D)$ (distributive laws). So we have $[z] \cap C \neq \phi$ or $[z] \cap D \neq \phi$. Now if $[z] \cap C \neq \phi$ this implies that $[z] \subseteq C \cup st(x)$ (by detention of W_V). Similarly, if $[z] \cap D \neq \phi$ then $[z] \subseteq (C \cup D) \cup st(x)$. But, this implies to $[z] \subseteq (C \cup D) \cup st(x)$. Hence, $\alpha_{C \cup D,x} \in W_V$. □

Recall that, W_V is the set of partial conjugations $\alpha_{C,x}$, where $x \in L = V \cup V^{-1}$ and (as well as being a union of connected components of $\Gamma \backslash st(x)$) the set C satisfies the condition that, for all $z \in V$ either

(i) $[z] \cap C = \phi$; *or*

(ii) $[z] \subseteq C \cup st(x)$.

Definition 4.3.8. [24] Let w be a graphically reduced cyclic word and let $a \in L$. Then for $b, c \in L \backslash \ell k_L(a)$, we define the **adjacency counter** of w relative to a, written as $\langle b, c \rangle_{w,a}$, to be the number of subsegments of w of the form $(buc^{-1})^{\pm 1}$, where u is any (possibly empty) word in $\ell k_L(a)$.

For a k-tuple of graphically reduced cyclic words $W = (w_1, \dots, w_k)$, define the adjacency counter of W relative to a as:

$$\langle b, c \rangle_{W,a} = \sum_{i=1}^{k} \langle b, c \rangle_{w_i,a}$$

For $B, C \subset L$, we define:

$$\langle B, C\rangle_{W,a} = \sum_{b\in(B\setminus \ell k_L(a))} \sum_{c\in(C\setminus \ell k_L(a))} \langle b, c\rangle_{W,a}$$

For $\alpha = \alpha_{C,a} \in W_V$, we define:

$$D_{[W]}(\alpha) = \mid \alpha \cdot [W] \mid - \mid [W] \mid$$

When W is clear, we leave it out, writing $\langle B, C\rangle_a$ and $D(\alpha)$.

With W and a as above, note that for any $B, C \subset L$, the number $\langle B, C\rangle_a \geq 0$. Further, we have $\langle B, C\rangle_a = \langle C, B\rangle_a$. If $D \subset L$ with $D \cap C = \emptyset$, then we have:

$$\langle B, C \cup D\rangle_a = \langle B, C\rangle_a + \langle B, D\rangle_a$$

Also note that $\langle a, a\rangle_a = 0$ (since each w_i is graphically reduced).

From the discussion of Section 4.2 recall that, for $\alpha_{C,a} \in W_V$ we have $A = C \cup C^{-1} \cup \{a\}$.

Lemma 4.3.9. *If W is a k-tuple of graphically reduced cyclic words, $\alpha_{C,a} \in W_V$, and W' is the obvious representative of $\alpha_{C,a} \cdot [W]$, then let $E = C \cup C^{-1}$*

$$D_{[W]}(\alpha_{C,a}) = \mid W' \mid - \mid W \mid = \langle E, L\setminus(E \cup \{a\})\rangle_{W,a} - \langle a, E\rangle_{W,a}.$$

Proof. This is immediate from counting the letters removed and added in the definition of W'.

□

Lemma 4.3.10. *[24] Let W be a k-tuple of graphically reduced cyclic words. If $\alpha_{C,a} \in W_V$, then let $A = C \cup C^{-1} \cup \{a\}$*

$$D_{[W]}(\alpha_{C,a}) = \langle A, L\setminus A\rangle_{W,a} - \langle a, L\rangle_{W,a}$$

Proof. From Lemma 4.3.9:

$$\begin{aligned} D(\alpha_{C,a}) &= \langle A\setminus\{a\}, L\setminus A\rangle_a - \langle a, A\setminus\{a\}\rangle_a \\ &= \langle A, L\setminus A\rangle_a - (\langle a, L\setminus A\rangle_a + \langle a, A\setminus\{a\}\rangle_a + \langle a, a\rangle_a) \\ &= \langle A, L\setminus A\rangle_a - \langle a, L\rangle_a \end{aligned}$$

□

Lemma 4.3.11. *[24] Let α, $\beta \in W_V$ and let $[W]$ be a k-tuple of conjugacy classes of G_Γ. Then we have:*

$$2 \mid \alpha^{-1} \cdot [W] \mid > \mid [W] \mid + \mid \beta\alpha^{-1} \cdot [W] \mid \tag{4.3.1}$$

Proof. Since $\beta\alpha^{-1}$ is a peak with respect to $[W]$, we can sum the two inequalities in the definition of a peak; by the fact that one of them is strict, we obtain this new inequality. □

Lemma 4.3.12. *[24] Suppose we have $\alpha_{C,a}, \alpha_{D,b} \in W_V$ with $a \notin D$ and a not adjacent to b in Γ (possibly $a = b^{-1}$). Then $\ell k_L(a) \cap D = \emptyset$.*

Proof. If $x \in \ell k_L(a) \cap D$, then $x \in D$ and by 2.4.6, either $b \geq x$ or $\alpha_{D,b}$ acts on the connected component of x in $\Gamma \backslash st(b)$ by conjugation. If the latter were true, since a is adjacent to x and not b, we would have that $a \in D$, a contradiction. So $b \geq x$, in which case a is adjacent to b, a contradiction. □

Lemma 4.3.13. *[24] Suppose α, $\beta \in W_V$ and $[W]$ is a k-tuple of conjugacy classes of G_Γ, and also that $\alpha = \alpha_{C,a}$, $\beta = \alpha_{D,b}$, and that either $e = \{a, b\}$ or that $(C \cap D) \cup (C \cap \{b, b^{-1}\}) \cup (D \cap \{a, a^{-1}\}) \cup (\{a\} \cap \{b\}) = \emptyset$ with $a^{-1} \notin D$. Then $\mid \beta \cdot [W] \mid < \mid \alpha^{-1} \cdot [W] \mid$.*

For the proof see [24] for all automorphisms in $Aut(G_\Gamma)$.

Given $\alpha = (A, a)$ Day defines $\bar{\alpha} = (A', a^{-1})$, where $A' = L \backslash (A \cup \ell k_L(a))$. In our terminology, when α is a basis conjugating automorphism, $\alpha = \alpha_{C,a}$, where $C = \{x \in V : x \in A, x \notin st_L(a)\}$, as above, so we define $\bar{\alpha} = \alpha_{C',a^{-1}}$, where $C' = V \backslash (C \cup st_L(a)) = \{x \in V : x \notin C \cup st_L(a)\}$.

Now suppose that $\beta = \alpha_{D,b}$ is another basis conjugating automorphism, and let $B = D \cup D^{-1} \cup \{b\}$ such that $D \subseteq \Gamma \backslash st(b) \subseteq V$ and $b \in L$, so that, written as a Whitehead automorphism, β is (B, b).

Note that, in our terminology $A \cap B = \emptyset$ if and only if

$$(C \cap D) \cup (C \cap \{b, b^{-1}\}) \cup (D \cap \{a, a^{-1}\}) \cup [\{a\} \cap \{b\}] = \emptyset.$$

Since $A = C \cup C^{-1} \cup \{a\}$ and $B = D \cup D^{-1} \cup \{b\}$, then $A \cap B = \emptyset$ if and only if

$$(C \cup C^{-1} \cup \{a\}) \cap (D \cup D^{-1} \cup \{b\}) = \emptyset$$

But,

$$
\begin{aligned}
(C \cup C^{-1} \cup \{a\}) \cap (D \cup D^{-1} \cup \{b\}) &= [(C \cup C^{-1} \cup \{a\}) \cap D][(C \cup C^{-1} \cup \{a\}) \cap \\
&\qquad D^{-1}] \cup [(C \cup C^{-1} \cup \{a\}) \cap \{b\})] \\
&= [(C \cap D) \cup (C^{-1} \cap D) \cup (\{a\} \cap D)] \\
&\quad \cup [(C \cap D^{-1}) \cup (C^{-1} \cap D^{-1}) \cup (\{a\} \cap D^{-1})] \\
&\quad \cup [(C \cap \{b\}) \cup (C^{-1} \cap \{b\}) \cup (\{a\} \cap \{b\}] \\
&= \emptyset \text{ if and only if}
\end{aligned}
$$

$$
(C \cap D) \cup (C \cap \{b, b^{-1}\}) \cup (D \cap \{a, a^{-1}\}) \cup (\{a\} \cap \{b\}) = \emptyset \iff
$$

$$
C \cap D = \emptyset,\ C \cap \{b, b^{-1}\} = \emptyset,\ D \cap \{a, a^{-1}\} = \emptyset \text{ and } \{a\} \cap \{b\} = \emptyset.
$$

Therefore,
$A \cap B = \emptyset \iff (C \cap D) \cup (C \cap \{b, b^{-1}\}) \cup (D \cap \{a, a^{-1}\}) \cup (\{a\} \cap \{b\}) = \emptyset$.

By the same argument we have that,
$A \cap B \neq \emptyset \iff (C \cap D) \cup (C \cap \{b, b^{-1}\}) \cup (D \cap \{a, a^{-1}\}) \cup (\{a\} \cap \{b\}) \neq \emptyset$
$\iff C \cap D \neq \emptyset,\ C \cap \{b, b^{-1}\} \neq \emptyset,\ D \cap \{a, a^{-1}\} \neq \emptyset$ and $\{a\} \cap \{b\} \neq \emptyset$.

Lemma 4.3.14. *Suppose α, $\beta \in W_V$ and $[W]$ is a k-tuple of conjugacy classes of G_Γ. If $\beta\alpha^{-1}$ forms a peak with respect to $[W]$, there exist $\delta_1, \ldots, \delta_k \in W_V$ such that $\beta\alpha^{-1} = \delta_k \ldots \delta_1$ and for each i, $1 \leq i < k$, we have:*

$$
|(\delta_i \ldots \delta_1) \cdot [W]| < |\alpha^{-1} \cdot [W]|
$$

A factorization of $\beta\alpha^{-1}$ is ***peak-lowering*** *if it satisfies the conclusions of the lemma, so Lemma 4.3.14 states that every peak has a peak-lowering factorization. Such a factorization might not be peak-reduced, but the height of its highest peak is lower than the height of the peak in $\beta\alpha^{-1}$.*

Proof. Assume that $\alpha = \alpha_{C,a}$ and $\beta = \alpha_{D,b} \in W_V$. As in the discussion following Lemma 4.3.3 let $\bar{\alpha} = \alpha_{C',a^{-1}}$, where $C' = V \backslash (C \cup st_L(a)$ and let $\bar{\beta} = \alpha_{D',b^{-1}}$, where $D' = V \backslash (D \cup st_L(b)$. (As usual refer to $a \in V$ as an element of G_Γ or a vertex of Γ, as convenient.) Also we refer to a^{-1} as a vertex of Γ (when really we mean a). By Equation ($R8$) in Section 4.2, these automorphisms describe the same element of $Out(G_\Gamma)$, and therefore

$$\alpha^{-1} \cdot [W] = \bar{\alpha}^{-1} \cdot [W] \text{ and } \beta\alpha^{-1} \cdot [W] = \bar{\beta}\alpha^{-1} \cdot [W].$$

Moreover, from Lemma 4.3.3, $\bar{\alpha}$ and $\bar{\beta}$ belong to W_V. We claim that if the lemma holds with α or β replaced with $\bar{\alpha}$ or $\bar{\beta}$ respectively, then it holds as originally stated. Suppose $\delta_k \ldots \delta_1$, with $\delta_i \in W_V$, is a peak-lowering factorization of $\bar{\beta}\alpha^{-1}$ (for example). By Equation $(R2)$ and $(R8)$ in Section 4.2, the element $\beta\bar{\beta}^{-1}$ is the partial conjugation $\alpha_{D \cup D',b}$ which is in W_V, because α, β and $\bar{\beta}$ are in W_V. If $|\beta\alpha^{-1} \cdot [W]| < |\alpha \cdot [W]|$ then

$$\beta\alpha^{-1} = \alpha_{D \cup D',b}\delta_k \ldots \delta_1$$

is a peak-lowering factorization of $\beta\alpha^{-1}$, since $\alpha_{D \cup D',b}$ does not change the length of any conjugacy class. Otherwise $|W| < |\alpha \cdot [W]|$. Again by Equation $(R8)$, $\bar{\beta}\beta$ is the partial conjugation (inner automorphism of conjugation by b) γ_b. So $(\bar{\beta}\alpha^{-1})^{-1}\beta\alpha^{-1}$ is $\alpha\gamma_b\alpha^{-1}$.

If $b \notin C$, then by Equations $(R9)$ in Section 4.2, we know $(\bar{\beta}\alpha^{-1})^{-1}\beta\alpha^{-1}$ is the conjugation γ_b.

If $b \in C$, then by Equation $(R8)$, we know $(\bar{\beta}\alpha^{-1})^{-1}\beta\alpha^{-1}$ is $\gamma_a\bar{\alpha}\gamma_b\bar{\alpha}^{-1}\gamma_a^{-1}$ which is then a product of conjugations by Equation $(R9)$. In any case, we have a product of conjugations $\gamma_j^{'} \ldots \gamma_1^{'}$ equal to to $(\bar{\beta}\alpha^{-1})^{-1}\beta\alpha^{-1}$; then

$$\beta\alpha^{-1} = \delta_k \ldots \delta_1\gamma_j^{'} \ldots \gamma_1^{'}$$

is a peak-lowering factorization of $\beta\alpha^{-1}$, since conjugation does not change the length of conjugacy classes. So we may swap out $\bar{\alpha}$ for α and $\bar{\beta}$ for β as needs be in the proof of this lemma. Also, by the symmetry in the definition of a peak, we may switch α and β if needed.

We fix a k-tuple of graphically reduced cyclic words W representing the conjugacy class $[W]$. Throughout this proof W' will denote the obvious representative of $\alpha^{-1} \cdot [W]$ based on W. We break this proof down into several cases.

Case(1): $a \in \ell k(b)$. Of course, this implies that $a \in st(b)$ and $b \in st(a)$ and since $C \cap st(a) = \phi = D \cap st(b)$, then $a \notin D \subseteq V$ and $b \notin C \subseteq V$. So a^{-1}, b^{-1} are not in C or D. Then by Equation $(R3b)$ of Section 4.2, we have:

$$\beta\alpha^{-1} = \alpha_{D,b}\alpha_{C,a^{-1}} = \alpha_{C,a^{-1}}\alpha_{D,b} = \alpha^{-1}\beta.$$

By Lemma 4.3.13, we know $|\beta \cdot [W]| < |\alpha^{-1} \cdot [W]|$, so the factorization is peak-lowering.

Case(2): $(C\cap D)\cup(C\cap\{b,b^{-1}\})\cup(D\cap\{a,a^{-1}\})\cup(\{a\}\cap\{b\}) = \emptyset$ and $a \notin \ell k(b)$. Note that the first condition means that $a \neq b$ and $a^{\pm 1} \notin D$, so either $a = b^{-1}$ or $a^{-1} \notin (D \cup D_1^{-1} \cup \{b\})$.

We have the following sub-cases:

Sub-case(2a): $a = b^{-1}$. By Equation $(R2)$ of Section 4.2, the following factorization is peak-lowering:

$$\beta\alpha^{-1} = \alpha_{D,b}\alpha_{C,b} = \alpha_{C\cup D,b}.$$

($\beta\alpha^{-1} = \delta_1$ and there is nothing to check to verify that this factorization is peak-lowering.)

Sub-case(2b): $a \neq b^{-1}$. In this case $a^{-1} \notin (D \cup D^{-1} \cup \{b\})$ and $a \notin \ell k(b)$. If $b^{\pm 1} \notin C$ then by $(R3a)$ of 4.2 we have,

$$\beta\alpha^{-1} = \alpha_{D,b}\alpha_{C,a^{-1}} = \alpha_{C,a^{-1}}\alpha_{D,b}$$

So by Lemma 4.3.13, we know that $|\beta \cdot [W]| < |\alpha^{-1} \cdot [W]|$, so these factorizations are peak-lowering.

Case(3): $(C\cap D)\cup(C\cap\{b,b^{-1}\})\cup(D\cap\{a,a^{-1}\})\cup(\{a\}\cap\{b\}) \neq \emptyset$ and $a \notin \ell k(b)$. We show we may assume that $a \notin (D \cup D^{-1} \cup \{b\})$ and $b \notin (C \cup C^{-1} \cup \{a\})$. First, by replacing β with $\bar{\beta}$, if necessary, we may assume $a \notin (D \cup D^{-1} \cup \{b\})$. If $b \notin (C \cup C^{-1} \cup \{a\})$ the claim holds, so assume that $b \in (C \cup C^{-1} \cup \{a\})$. If $b = a$ then $a \in (D \cup D^{-1} \cup \{b\})$, a contradiction. Hence we have $b \neq a$. If also $b \neq a^{-1}$ then swapping α with $\bar{\alpha}$ we have $b \notin (C \cup C^{-1} \cup \{a\})$, and the result holds. Thus we may assume that $b = a^{-1}$. However this gives $a^{-1} = b \in (C \cup C^{-1})$, a contradiction. This proves the claim.

Hence we assume that $a \notin D \cup D^{-1} \cup \{b\}$ and $b \notin C \cup C^{-1} \cup \{a\}$. We wish to show that $\alpha_{C\cap D',a}$ is a well defined element of W_V. Note that if $a = b^{-1}$ then $st_L(a) = st_L(b)$ so the result follows from Lemma 4.3.3 and Lemma 4.3.5, so we may assume $a \neq b^{-1}$.

If $\alpha_{C\cap D',a}$ is a well defined element of $Conj_V$; then it is in W_V by Lemma 4.3.6. Now $\alpha_{C\cap D',a}$ is well defined if, for all $x \in C \cap D'$, $x \notin st(a)$, the component of $\Gamma \backslash st(a)$ containing x is contained in $C \cap D'$.

Suppose that the connected component of $\Gamma\backslash st(a)$ containing x in Y and that there exists $y \in Y$ with $y \notin C \cap D'$. As $\alpha_{C,a}$ is in $Conj$, we have $Y \subseteq C$; so $y \in C$ and thus $y \notin D'$. Therefore $y \in V\backslash D'$ so $y \in D \cup st(b)$. By Lemma 4.3.12 we have $C \cap \ell k(b) = \emptyset$ (also $D \cap \ell k(a) = \emptyset$) so either $y \in D$ or $y = b$; but $b \notin C$ so $y \neq b$, and so $y \in D$.

Let Z be the connected component of $\Gamma\backslash st(b)$ containing y. Then, as $y \in D$ we have $Z \subseteq D$. As $a \notin D$ this means $a \notin Z$; so $st(a) \cap Z = \emptyset$, (because a is not adjacent to b, and not equal to b and if we had $a = b^{-1}$ then we would have $st(a) = st(b)$; which intersects Z trivially. In other words, if $v \in st(a)$ then either $a \in Z$ or $a \in \ell k(b)$ and either case gives a contradiction, so there is no $v \in Z \cap st(a)$.) As $b \notin \ell k(a)$ and $b \neq a^{\pm 1}$ we have $b \notin st(a) \cup C$. To walk from y to any vertex outside C we must use vertices of $st(a)$ which implies that $Z \subseteq Y \subseteq C$ so Z is a connected component of $\Gamma\backslash st(a)$ which implies that $Y = Z$ which in terms implies that $x \in Z \subseteq D$. However, by assumption $x \in D'$ so this is a contradiction. Thus $C \cap D'$ is a union of connected component of $\Gamma\backslash st(a)$ as required. Therefore, $\alpha_{C\cap D',a}$ is a well defined automorphism and from Lemma 4.3.6 it belongs to W_V. Note that $\alpha_{D\cap C',b}$ is well defined by the same argument.

Next we will show that either $\alpha_{C\cap D',x}$ or $\alpha_{D\cap C',y}$ shortens $\alpha^{-1} \cdot [W]$. By Equation (4.3.1), we know that $0 > D_{[\alpha^{-1}.W]}(\alpha) + D_{[\alpha^{-1}.W]}(\beta)$. Of course, from the definition of peak-lowering we have,

$\mid \alpha^{-1} \cdot [W] \mid \geq \mid [W] \mid$ and $\mid \alpha^{-1} \cdot [W] \mid \geq \mid \beta\alpha^{-1} \cdot [W] \mid$ (and one of these is strict). By adding these two inequalities to each other we will get that

$$2 \mid \alpha^{-1} \cdot [W] \mid > \mid [W] \mid + \mid \beta\alpha^{-1} \cdot [W] \mid . \tag{4.3.2}$$

Now from Definition 4.3.8 we have that,

$$D_{[W]}(\alpha) = \mid \alpha \cdot [W] \mid - \mid [W] \mid,$$

$$D_{[\alpha^{-1}.W]}(\alpha) = \mid \alpha \cdot \alpha^{-1} \cdot [W] \mid - \mid \alpha^{-1} \cdot [W] \mid = \mid [W] \mid - \mid \alpha^{-1} \cdot [W] \mid \tag{4.3.3}$$

and

$$D_{[\alpha^{-1}\cdot W]}(\beta) = \mid \beta \cdot \alpha^{-1} \cdot [W] \mid - \mid \alpha^{-1} \cdot [W] \mid . \tag{4.3.4}$$

By adding Equation (4.3.3) to Equation (4.3.4) we get that

$$D_{[\alpha^{-1}\cdot W]}(\alpha) + D_{[\alpha^{-1}\cdot W]}(\beta) = -(2 \mid \alpha^{-1} \cdot [W] \mid) + \mid [W] \mid + \mid \beta \cdot \alpha^{-1} \cdot [W] \mid < 0$$

(as $2 \mid \alpha^{-1} \cdot [W] \mid > \mid [W] \mid + \mid \beta \cdot \alpha^{-1} \cdot [W] \mid$ from Equation (4.3.2)).

Now by Lemma 4.3.10, where $A = C \cup C^{-1} \cup \{a\}$ and $A' = L \backslash (A \cup \ell k_L(a))$ we know that

$$\begin{aligned} D_{[\alpha^{-1}\cdot W]}(\alpha) &= \langle A, A' \rangle_{\alpha^{-1}\cdot W,a} - \langle a, L \rangle_{\alpha^{-1}\cdot W,a} \\ &= \langle A \cap B', A' \rangle_{\alpha^{-1}\cdot W,a} + \langle A \cap B, A' \rangle_{a} - \langle a, L \rangle_{\alpha^{-1}\cdot W,a} \end{aligned}$$

and similarly, where $B = D \cup D^{-1} \cup \{b\}$ and $B' = L \backslash (B \cup \ell k_L(b))$ we have:

$$\begin{aligned} D_{[\alpha^{-1}\cdot W]}(\beta) &= \langle B, B' \rangle_{\alpha^{-1}\cdot W,b} - \langle b, L \rangle_{\alpha^{-1}\cdot W,b} \\ &= \langle B \cap A', B' \rangle_{\alpha^{-1}\cdot W,b} + \langle B \cap A, B' \rangle_{\alpha^{-1}\cdot W,b} - \langle b, L \rangle_{\alpha^{-1}\cdot W,b} \end{aligned}$$

From above we have that $a, b \in L = V \cup V^{-1}$ with $a \neq b^{\pm 1}$, $C \subset V$, $D \subset V$, $A = C \cup C^{-1} \cup \{a\}$ and $A' = L \backslash (A \cup \ell k_L(a))$, $B = D \cup D^{-1} \cup \{b\}$ and $B' = L \backslash (B \cup \ell k_L(b))$, $D' = V \backslash (D \cup st_V(v(b)))$, $a \notin B$ with $a \notin \ell k_L(b)$, $b \notin A$, $C \cap \ell k_L(b) = \emptyset$ and from Lemma 4.3.12, $D \cap \ell k_L(a) = \emptyset$.

By definition $C \cap st_V(v(a)) = \emptyset$ and $D \cap st_V(v(b)) = \emptyset$.

Claim: $A \cap B' = (C \cap D') \cup (C \cap D')^{-1} \cup \{a\} = A_1$. First consider a. Note that $a \in A \cap B'$, as $a \in A$ and $a \notin \ell k_L(b)$ and $a \notin B$ implies that $a \in B'$ and by definition $a \in K$.

If $x = a^{-1}$ then $x \notin A$, as $C \cap st_V(v(a)) = \emptyset$ so $x \notin A \cap B'$. Also if $x = a^{-1}$ then $x \notin C \cap D'$ and $x \notin \{a\}$ so $x \notin K$.

Now consider $x = b^{\pm 1}$. We have $b \neq a^{\pm 1}$ and $b \notin A$, $b^{-1} \notin A$, so $b^{\pm 1} \notin A \cap B'$.

Also $b \notin C \cup C^{-1}$ implies that $b \notin C \cap D'$ or $(C \cap D')^{-1}$ and $b^{\pm 1} \notin \{a\}$ so $b^{\pm 1} \notin K$.

If $x \in A \cap B'$ with $x \neq a^{\pm 1}, b^{\pm 1}$ then $x \in A$, $x \neq a^{\pm 1}, b^{\pm 1}$ implies that $x \in C \cup C^{-1}$.

Also $x \in B'$ with $x \neq a^{\pm 1}, b^{\pm 1}$ implies that $x \notin B \cup \ell k_L(b)$ and $x \neq a^{\pm 1}, b^{\pm 1}$ if and only if $x \notin D \cup D^{-1} \cup \{b\} \cup \ell k_L(b)$, $x \neq a^{\pm 1}$, $x \neq b^{\pm 1}$. Then $x \in V$ and $x \in B'$ if and only if $x \notin D \cup st_V(v(b))$ and $x \neq a^{\pm 1}$; $x \in V^{-1}$ and $x \in B'$ if and only if $x \notin D^{-1} \cup st_V(v(b))^{-1}$ and $x \neq a^{\pm 1}$ so $x \in B'$ if and only if $x \notin (D \cup st_V(v(b)))^{\pm 1}$ and $x \neq a^{\pm 1}$, if and only if $x \in D^{\pm 1}$ and $x \neq a^{\pm 1}$. Hence $x \in A \cap B'$ and $x \neq a^{\pm 1}, b^{\pm 1}$

if and only if $x \in (C \cap D') \cup (C \cap D')^{-1}$ and $x \neq a^{\pm 1}$ if and only if $x \in A_1$.

$$A_1 = (C \cap D') \cup (C \cap D')^{-1} \cup \{a\} = A \cap B'$$

By the same argument we have that,

$$B_1 = (C' \cap D) \cup (C' \cap D)^{-1} \cup \{b\} = A' \cap B.$$

Let $A'_1 = L \backslash (A_1 \cup \ell k(a))$ and $B'_1 = L \backslash (B_1 \cup \ell k(b))$.

Now from Lemma 4.3.10, we know that

$$\begin{aligned} D_{[\alpha^{-1} \cdot W]}(\alpha_{C \cap D', a}) &= \langle A_1, A'_1 \rangle_{\alpha^{-1} \cdot W, a} - \langle a, L \rangle_{\alpha^{-1} \cdot W, a} \\ &= \langle A \cap B', L \backslash (A \cap B' \cup \ell k(a)) \rangle_{\alpha^{-1} \cdot W, a} - \langle a, L \rangle_{\alpha^{-1} \cdot W, a} \\ &= \langle A \cap B', L \backslash (A \cap B') \rangle_{\alpha^{-1} \cdot W, a} - \langle a, L \rangle_{\alpha^{-1} \cdot W, a} \end{aligned}$$

(as $\langle W, \ell k(a) \rangle = 0$ for all $W \subset L$.) Note that,

$L \backslash (A \cap B') = (A' \cup B) \cup (\ell k(a) \backslash B) \cup (A \cap \ell k(b)) = (A' \cup B) \cup (\ell k(a) \backslash B)$ as $A \cap \ell k(b) = \emptyset$ (by Lemma 4.3.12). Since if $U \subset L$ with $U \cap V = \emptyset$, then we have: $\langle B, U \cup V \rangle_a = \langle B, U \rangle_a + \langle B, V \rangle_a$, and $(A' \cup B) \cap (\ell k(a) \backslash B) = \emptyset$. So

$$\begin{aligned} D_{[\alpha^{-1} \cdot W]}(\alpha_{C \cap D', a}) &= \langle A \cap B', (A' \cup B) \cup (\ell k(a) \backslash B) \rangle_{\alpha^{-1} \cdot W, a} - \langle a, L \rangle_{\alpha^{-1} \cdot W, a} \\ &= \langle A \cap B', A' \cup B \rangle_{\alpha^{-1} \cdot W, a} + \langle A \cap B', \ell k(a) \backslash B \rangle_{\alpha^{-1} \cdot W, a} - \langle a, L \rangle_{\alpha^{-1} \cdot W, a} \\ &= \langle A \cap B', A' \cup B \rangle_{\alpha^{-1} \cdot W, a} - \langle a, L \rangle_{\alpha^{-1} \cdot W, a} \text{ (as } \langle A \cap B', \ell k(a) \backslash B \rangle = 0) \\ &= \langle A \cap B', A' \cup (A \cap B) \rangle_{\alpha^{-1} \cdot W, a} - \langle a, L \rangle_{\alpha^{-1} \cdot W, a} \\ &= \langle A \cap B', A' \rangle_{\alpha^{-1} \cdot W, a} + \langle A \cap B', A \cap B \rangle_{\alpha^{-1} \cdot W, a} - \langle a, L \rangle_{\alpha^{-1} \cdot W, a} \end{aligned}$$

(as $A' \cup B = A' \cup (A \cap B)$ with $A' \cap (A \cap B) = \emptyset$).

Similarly,

$$D_{[\alpha^{-1} \cdot W]}(\alpha_{C' \cap D, b}) = \langle B \cap A', B' \rangle_{\alpha^{-1} \cdot W, a} + \langle B \cap A', A \cap B \rangle_{\alpha^{-1} \cdot W, a} - \langle a, L \rangle_{\alpha^{-1} \cdot W, a}.$$

We claim that $\langle A \cap B, A' \rangle_{\alpha^{-1} \cdot W, a} \geq \langle A \cap B, A' \cap B \rangle_{\alpha^{-1} \cdot W, b}$. Recall that $\ell k_L(b) \cap C = \emptyset$. If $(cud^{-1})^{\pm 1}$ is a subsegment of $\alpha^{-1} \cdot W$ with $c \in A \cap B$, $d \in A' \cap B$, and u a word in $\langle \ell k(b) \rangle$, then either u is a word in $\langle \ell k(b) \cap \ell k(a) \rangle$, or $u = u' u_1 u''$ where u' a word in $\langle \ell k(b) \cap \ell k(a) \rangle$ and $u_1 \in \ell k(b) \backslash \ell k(a)$. If the former is true, cud^{-1} is counted by $\langle A \cap B, A' \rangle_{\alpha^{-1} \cdot W, a}$; if the latter holds, then instead $cu'u_1$ is counted by $\langle A \cap B, A' \rangle_{\alpha^{-1} \cdot W, a}$ (since $u_1 \notin \ell k(a)$). Either way, each subsegment of $\alpha^{-1} \cdot W$ counted by the counter on the right hand side of the inequality is also counted by the counter on the left hand side of the inequality, showing the inequality. Similarly, we know $\langle B \cap A, B' \rangle_{\alpha^{-1} \cdot W, b} \geq \langle B \cap A, B' \cap A \rangle_{\alpha^{-1} \cdot W, a}$.

According to the above we have the following:
$0 > D_{[\alpha^{-1} \cdot W]}(\alpha) + D_{[\alpha^{-1} \cdot W]}(\beta)$, $\langle A \cap B, A' \rangle_{\alpha^{-1} \cdot W, a} \geq \langle A \cap B, A' \cap B \rangle_{\alpha^{-1} \cdot W, b}$ and $\langle B \cap A, B' \rangle_{\alpha^{-1} \cdot W, b} \geq \langle B \cap A, B' \cap A \rangle_{\alpha^{-1} \cdot W, a}$.
So,

$$\begin{aligned}
0 &> D_{[\alpha^{-1} \cdot W]}(\alpha) + D_{[\alpha^{-1} \cdot W]}(\beta) \\
&\geq \langle A \cap B', A' \rangle_{\alpha^{-1} \cdot W, a} + \langle A \cap B', A \cap B \rangle_{\alpha^{-1} \cdot W, a} - \langle a, L \rangle_{\alpha^{-1} \cdot W, a} \\
&\quad + \langle B \cap A', B' \rangle_{\alpha^{-1} \cdot W, b} + \langle B \cap A', A \cap B \rangle_{\alpha^{-1} \cdot W, b} - \langle a, L \rangle_{\alpha^{-1} \cdot W, b} \\
&= D_{[\alpha^{-1} \cdot W]}(\alpha_{C \cap D', a}) + D_{[\alpha^{-1} \cdot W]}(\alpha_{D \cap C', b}).
\end{aligned}$$

So one of $\alpha_{C \cap D', a}$ and $\alpha_{D \cap C', b}$ shortens $[\alpha^{-1} \cdot W]$. □

Theorem 4.3.15. *The subgroup $Conj_V$ of $Aut(G_\Gamma)$ has a presentation with generators W_V (see Definition 4.3.1) and the finite set of relations $\Re$:*

($\Re$1) $(\alpha_{C,x})^{-1} = \alpha_{C,x^{-1}}$,

($\Re$2) $\alpha_{C,x} \alpha_{D,x} = \alpha_{C \cup D, x}$ *if* $C \cap D = \phi$,

($\Re$3) $\alpha_{C,x} \alpha_{D,y} = \alpha_{D,y} \alpha_{C,x}$ *if* $x \notin D$, $y \notin C$, $x \neq y, y^{-1}$ *and at least one of* $C \cap D = \phi$ *or* $y \in \ell k(x)$ *holds,*

($\Re$4) $\gamma_y \alpha_{C,x} \gamma_y^{-1} = \alpha_{C,x}$ *if* $y \notin C$, $x \neq y, y^{-1}$.

Proof. Our proof is based on arguments that were used in Lemma 4.3.14. Assume that $\alpha = \alpha_{C,a}$ and $\beta = \alpha_{D,b} \in W_V$. Let $\pi \in Aut(\Gamma)$, then by Lemma 4.3.4, $\alpha_{\pi(C), \pi(a)} \in W_V$. We also denote by Ω_ℓ the set of long-range Whitehead automorphisms. (As usual we refer to $a \in V$ as an element of G_Γ or a vertex of Γ, as convenient.) Also we refer to a^{-1} as a vertex of Γ (when really we mean $a = v(a^{-1})$). Let $\Re$ denote the set of relations given in the statement of Theorem

4.3.15. We shall construct a finite connected 2-complex K with fundamental group $Conj_V = \langle W_V \mid \Re \rangle$.

Let $V = V(\Gamma) = \{x_1, \ldots, x_n\} (n \geq 1)$. Let W denote the n-tuple $(x_1, \ldots, x_n)$.

The set of vertices $K^{(0)}$ of K is the set of n-tuples $\pi \cdot W$, where π ranges over the set $Aut(\Gamma)$. For any $\pi, \psi \in Aut(\Gamma)$, the vertices $\pi \cdot W$ and $\psi\pi \cdot W$ are joined by a directed edge $(\pi \cdot W, \psi\pi \cdot W; \psi)$ labelled ψ. Note that, at this stage, K is just the Cayley graph of $Aut(\Gamma)$. Next, for any $\pi \in Aut(\Gamma)$, and $\alpha_{C,a} \in W_V$, we add a loop $(\pi \cdot W, \pi \cdot W; \alpha_{C,a})$ labeled $\alpha_{C,a}$ at $\pi \cdot W$. This defines the 1-skeleton $K^{(1)}$ of K.

We shall define the 2-cells of K. These 2-cells will derive from the relations $(R1) - (R10)$ of Section 4.2. First, let K_1 be the 2-complex obtained by attaching 2-cells corresponding to relation $(R7)$ to $K^{(1)}$. Note that, if M is the 2-complex obtained from K_1 by deleting the loops $(\pi \cdot W, \pi \cdot W; \alpha_{C,a})$, $\pi \in \Omega_1, \alpha_{C,a} \in W_V$, then M is just the Cayley complex of Ω_1, and therefore is simply connected. We now explore the relations $(R1) - (R5)$ and $(R8) - (R10)$ of Section 4.2 to determine which of these will give rise to relations on the elements of W_V. When we apply these relations to elements $\alpha_{C,a}, \alpha_{D,b} \in W_V$ we have to write $\alpha_{C,a}$ as $\tau_{C_0,a}\alpha_{C_1,a}$ and $\alpha_{D,b}$ as $\tau_{D_0,b}\alpha_{D_1,b}$ and here $C_0 = D_0 = \emptyset$ and $C_1 = C$, $D_1 = D$. Thus $\tau_{C_0,a}\alpha_{C_1,a}$ and $\tau_{D_0,b}\alpha_{D_1,b}$ become $\alpha_{C,a}$ and $\alpha_{D,b}$ respectively. Relation $(R1)$ will give rise to the following:

$$\alpha_{C,a}^{-1} = \alpha_{C,a^{-1}} \tag{4.3.5}$$

for $\alpha_{C,a} \in W_V$ (by definition $\alpha_{C,a^{-1}} \in W_V$).

Relation $(R2)$ will give rise to

$$\alpha_{C,a}\alpha_{D,a} = \alpha_{C \cup D,a} \tag{4.3.6}$$

for $\alpha_{C,a}, \alpha_{D,a} \in W_V$ as, by Lemma 4.3.7, $\alpha_{C \cup D,a} \in W_V$, with $C \cap D = \emptyset$.

Relation $(R3)$ will give rise to

$$\alpha_{C,a}\alpha_{D,b} = \alpha_{D,b}\alpha_{C,a} \tag{4.3.7}$$

for $\alpha_{C,a}, \alpha_{D,b} \in W_V$, such that $a \notin D$, $a^{-1} \notin D$, $b \notin C$, $b^{-1} \notin C$, and at least one of (a) $C \cap D = \phi$ or (b) $b \in \ell k_L(a)$ holds.

From $(R4)$, no relations arise. Indeed, in our case $C_0 = \emptyset$ so we cannot have $b^{-1} \in C_0$.

From ($R5$), no relations arise (by the same argument as above).

From ($R8$), we obtain a relation which is a direct consequence of (4.3.5) and (4.3.6). Indeed, if $E_1 = V\backslash[C \cup st_V(v(a))]$ then, from (4.3.6) $\gamma_a = \alpha_{C\cup E_1,a} = \alpha_{C,a}\alpha_{E_1,a}$. So, from (4.3.5) $\alpha_{C,a} = \gamma_a \alpha_{E_1,a^{-1}}$.

Relation ($R9$) will give rise to the following:

$$\alpha_{C,a}\alpha_{V\backslash st_V(b),b}\alpha_{C,a}^{-1} = \alpha_{V\backslash st_V(b),b} \text{ (note that } \alpha_{\mathrm{V}\backslash \mathrm{st_V(b),b}} = \gamma_{\mathrm{b}}) \tag{4.3.8}$$

for $\alpha_{C,a} \in W_V$, and $b \in L$ such that $b \notin C$, and $b^{-1} \notin C$ as $\alpha_{V\backslash st_V(b),b} \in W_V$ by definition.

From ($R10$), no relations arise (by the same argument as above).

We rewrite the relations (4.3.5)-(4.3.8) in the form

$$\sigma_k^{\epsilon_k} \dots \sigma_1^{\epsilon_1} = 1$$

where $\sigma_1, \dots, \sigma_k \in W_V$ and $\epsilon_1, \dots, \epsilon_k \in \{-1, 1\}$. Let K_2 be the 2-complex obtained from K_1 by attaching 2-cells corresponding to the relations (4.3.5)-(4.3.8).

Note that the boundary of each of these 2-cells has the from

$$(\pi \cdot W, \pi \cdot W; \sigma_1)^{\epsilon_1}(\pi.W, \pi \cdot W; \sigma_2)^{\epsilon_2} \dots (\pi \cdot W, \pi \cdot W; \sigma_k)^{\epsilon_k},$$

for $\pi \in Aut(\Gamma)$.

Finally, relation ($R6$) will give rise to the following:

$$\pi(\alpha_{C,a})\pi^{-1}\alpha_{\pi(C),\pi(a)}^{-1} = 1, \tag{4.3.9}$$

for $\alpha_{C,a} \in W_V$ with $\pi \in Aut(\Gamma)$. As noted above $\alpha_{\pi(C),\pi(a)} \in W_V$. Then K is obtained from K_2 by attaching 2-cells corresponding to the relations (4.3.9). Observe that the boundary of each of these 2-cells has the form

$$\begin{aligned}(\psi \cdot W, \psi \cdot W; \alpha_{\pi(C),\pi(a)})^{-1}(\psi \cdot W, \pi^{-1}\psi \cdot W; \pi)\\ (\pi^{-1}\psi \cdot W, \pi^{-1}\psi \cdot W; \alpha_{C,a})(\pi^{-1}\psi \cdot W, \psi \cdot W; \pi),\end{aligned}$$

for $\psi \in Aut(\Gamma)$.

It remains to show that $\pi_1(K, W) = Conj_V = \langle W_V \mid \Re \rangle$.

Let T be a maximal tree in the 1-skeleton $K^{(1)}$ of K. Note that T is in fact a maximal tree in the 1-skeleton $C^{(1)}$ of C (i.e., the Cayley graph of $Aut(\Gamma)$. We compute a presentation of $\pi_1(K, W)$ using T. For every vertex V in K, there exists

a unique reduced path p_V from W to V in T. To each edge $(V_1, V_2; \pi)$ of K, we associate the element $\pi_1(K, W)$ represented by the loop $p_{V_1}(V_1, V_2; \pi)p_{V_2}^{-1}$. We again denote this by $(V_1, V_2; \pi)$. Evidently these elements generate $\pi_1(K, W)$.
Now, since M is simply connected, we have

$$(\pi \cdot W, \psi\pi \cdot W; \psi) = 1 \ \ (in\ \pi_1(K, W)), \tag{4.3.10}$$

for all $\pi, \psi \in Aut(\Gamma)$.

Let $\mathcal{P}$ be the set of combinatorial paths in the 1-skeleton $K^{(1)}$ of K. We define a map $\widehat{\varphi} : \mathcal{P} \to Aut(G_\Gamma)$ as follows. For an edge $e = (V_1, V_2; \pi)$, we set $\widehat{\varphi}(e) = \pi$, and for a path $p = e_k^{\epsilon_k} \dots e_1^{\epsilon_1}$, we set $\widehat{\varphi(p)} = \widehat{\varphi}(e_k)^{\epsilon_k} \dots \widehat{\varphi}(e_1)^{\epsilon_1}$. Clearly, if p_1 and p_2 are loops at W such that $p_1 \sim p_2$, then $\widehat{\varphi}(p_1) = \widehat{\varphi}(p_2)$. Hence, $\widehat{\varphi}$ induces a map $\varphi : \pi_1(K, W) \to Aut(G_\Gamma)$. Then from (4.3.9) and (4.3.10) it is easily seen that φ is a homomorphism. So φ maps $\pi_1(K, W)$ to $Conj_V$. It follows immediately from the construction of K that $\varphi : \pi_1(K, W) \to Conj_V$ is surjective. Thus, it suffices to show that φ is injective. Let p be a loop at W such that $\varphi(p) = 1$. We have to show that $p \sim 1$. Write $p = e_k^{\epsilon_k} \dots e_1^{\epsilon_1}$, where $k \geq 1$ and $\epsilon_i \in \{-1, 1\}$ for all $i \in \{1, \dots, k\}$. Using the 2-cells arising from (4.3.5) and the fact that $Aut(\Gamma)^{-1} = Aut(\Gamma)$, we can restrict our attention to the case where $p = e_k \dots e_1$. Set $\pi_i = \varphi(e_i)$ for all $i \in \{1, \dots, k\}$. Note that $\pi_i \in W_V \cup Aut(\Gamma) \subset \Omega_\ell$ for all $i \in \{1, \dots, k\}$.

Let Z be a tuple containing each conjugacy class of length 2 of G_Γ, each appearing once. We prove the following:

Claim There exist $e'_\ell \dots e'_1$ such that $p \sim e'_\ell \dots e'_1$ and if we set $\pi'_i = \varphi(e'_i)$ for all $i \in \{1, \dots, \ell\}$, then $\pi'_i \in Aut(\Gamma)$ or $\pi'_i \in W_V \cap Inn(G_\Gamma)$ for each $i \in \{1, \dots, \ell\}$.

First, we examine the case where $\pi_k \dots \pi_1$ is peak-reduced with respect to Z. We claim that the sequence

$$\mid Z \mid, \mid \pi_1 \cdot Z \mid, \mid \pi_2\pi_1 \cdot Z \mid, \dots, \mid \pi_{k-1} \dots \pi_1 \cdot Z \mid, \mid \pi_k \dots \pi_1 \cdot Z \mid = \mid Z \mid$$

is a constant sequence. Suppose the contrary. By Lemma 2.6.4, $\mid Z \mid$ is the least element of the set $\{\mid \pi \cdot Z \mid \mid \pi \in \langle \Omega_\ell \rangle\}$. Hence we can find $i \in \{1, \dots, k-1\}$ such that we have

$$\mid \pi_{i-1} \dots \pi_1 \cdot Z \mid \leq \mid \pi_i \dots \pi_1 \cdot Z \mid,$$

$$| \pi_{i+1} \dots \pi_1 \cdot Z | \leq | \pi_i \dots \pi_1 \cdot Z |,$$

and at least one of these inequalities is strict, which contradicts the fact that the product $\pi_k \dots \pi_1$ is peak-reduced. Therefore we have

$$| \pi_i \dots \pi_1 \cdot Z | = | Z |$$

for all indices $i \in \{1, \dots, k\}$. We argue by induction on $i \in \{1, \dots, k\}$ to prove that $\pi_i \dots \pi_1 \cdot Z$ is a tuple containing each conjugacy class of length 2 of G_Γ, each appearing once. The result holds for $i = 0$ by assumption. Suppose that $i \geq 1$, and that the result holds for $i-1$. Observe that $Aut(\Gamma)$ does not change the length of a conjugacy class. Thus, we can assume that π_i is in W_V. Since $| \pi_i \pi_{i-1} \dots \pi_1 \cdot Z | = | \pi_{i-1} \dots \pi_1 \cdot Z |, \pi_i$ is trivial, or an inner automorphism by Lemma 2.6.4 Thus, the result holds for i. In this case, p has already the desired form.

We now turn to prove the claim. We define

$$h_p = max\{| \pi_i \dots \pi_1 \cdot Z | \mid i \in \{0, \dots, k\}\}$$

and

$$N_p = | \{i \mid i \in \{0, \dots, k\} \; and \; | \pi_i \dots \pi_1 \cdot Z | = h_p\} | \, .$$

We use induction on pairs (h_p, N_p) with left lexicographic order. The base of induction is $| Z |$: the smallest possible value for h_p by Lemma 2.6.4. If $h_p = | Z |$, then the product $\pi_k \dots \pi_1$ is peak-reduced and we are done. Thus, we can assume that $h_p > | Z |$ and that the result has been proved for all loops p' with $h_{p'} < h_p$. Let $i \in \{1, \dots, k\}$ be such that π_i is a peak of height h_p. An examination of the proof of Lemma 4.3.14 shows that $e_{i+1} e_i \sim f_j \dots f_1$ such that, if we set $\psi_k = \varphi(f_k)$ for all $k \in \{1, \dots, j\}$, then

$$| \psi_k \dots \psi_1 \pi_{i-1} \dots \pi_1 \cdot Z | < | \pi_i \pi_{i-1} \dots \pi_1 \cdot Z | \tag{4.3.11}$$

for all $k \in \{1, \dots, j-1\}$. Therefore, we get

$$p \sim e_k \dots e_{i+2} f_j \dots f_1 e_{i-1} \dots e_1 = p',$$

and a new product $\pi_k \dots \pi_{i+2} \psi_j \dots \psi_1 \pi_{i-1} \dots \pi_1$. We argue by induction on N_p. If $N_p = 1$, then (4.3.11) implies that $h_{p'} < h_p$ and we can apply the induction hypothesis on h_p. If $N_p \geq 2$ then (4.3.11) implies that $h_p = h_{p'}$ and $N_{p'} < N_p$ and we can apply the induction hypothesis on N_p. This proves the claim.

Hence, using the 2-cells arising from the relations (4.3.9), we obtain

$$p \sim h_s \dots h_1 g_r \dots g_1,$$

where, if we set

$$\gamma_i = \varphi(g_i) \text{ for all } i \in \{1, \dots, r\} \text{ and } \delta_j = \varphi(h_j) \text{ for all } j \in \{1, \dots, s\},$$

then $\delta_i \in Aut(\Gamma)$ for all $i \in \{1, \dots, s\}$ and $\gamma_j \in W_V \cap Inn(G_\Gamma)$ for all $j \in \{1, \dots, r\}$. Using relation (4.3.7), we obtain $p \sim g_r \dots g_1$. Set $\mathcal{Z} = \bigcap_{v \in V} st(v)$. It follows from Servatius' Centralizer Theorem (see [69]) that the center $Z(G_\Gamma)$ of G_Γ is the special subgroup of G_Γ generated by $\mathcal{Z}$. Let Γ' be the full subgraph of Γ spanned by $V \setminus \mathcal{Z}$. We have

$$G_{\Gamma'} \simeq Inn(G_\Gamma),$$

where the isomorphism is given by $v \mapsto w_v$ (see, for example, Lemma 5.3 of [69]). Write

$$\gamma_i = \alpha_{V \setminus st_L(c_i), c_i}$$

where $c_i \in V \backslash \mathcal{Z} \cup (V \backslash \mathcal{Z})^{-1} (i \in \{1, \dots, r\})$. Since $\gamma_r \dots \gamma_1 = 1$ (in $Inn(G_\Gamma)$), we have $c_r \dots c_1 = 1$ (in $G_{\Gamma'}$). Therefore $c_r \dots c_1$ is a product of conjugates of defining relators of G_Γ. Using the 2-cells corresponding to the relations (4.3.5) and (4.3.7)(b), we deduce that $p \sim 1$. We conclude that φ is injective, and thus

$$Conj_V = \pi_1(K, W).$$

Now, using the 2-cells arising from the relations (4.3.9) (with $\pi = \psi$), we obtain

$$(\pi \cdot W, \pi \cdot W; \alpha_{\pi(C), \pi(a)}) = (\pi \cdot W, W; \pi^{-1})(W, W; \alpha_{C,a})(W, \pi \cdot W; \pi). \qquad (4.3.12)$$

Note that, using (4.3.10) with π^{-1} instead of π and $\psi = \pi$ then $(\pi \cdot W, W; \pi^{-1}) =$

$(\pi^{-1} \cdot W, W; \pi) = (\pi^{-1} \cdot W, W; \psi) = 1$, and also with $\pi = 1$ and $\psi = \pi$ then $(W, \pi \cdot W; \pi) = 1$. Thus (4.3.12) becomes

$$(\pi \cdot W, \pi \cdot W; \alpha_{\pi(C),\pi(a)}) = (W, W; \alpha_{C,a}), \tag{4.3.13}$$

for all $\pi \in Aut(\Gamma)$, and $\alpha_{C,a} \in W_V$. It then follows that $Conj_V$ is generated by the $(W, W; \alpha_{C,a})$, for $\alpha_{C,a} \in W_V$. We identify $(W, W; \alpha_{C,a})$ with $\alpha_{C,a}$ for all $\alpha_{C,a} \in W_V$. Any relation in $Conj_V = \pi_1(K, W)$ will be a product of conjugates of boundary labels of 2-cells of K. Then, using relation (4.3.13) and identifying $(W, W; \alpha_{C,a})$ with $\alpha_{C,a}$, we see that the relations (4.3.5)-(4.3.8) above are equivalent to those of R. We have shown that $Conj_V$ has the presentation $\langle W_V \mid \Re \rangle$. $\square$

Example 4.3.0.3

We will find a presentation for a subgroup $Conj_V$ of the automorphism group $Aut(G_\Gamma)$, that is correspond to the graph Γ of Figure 4.5.

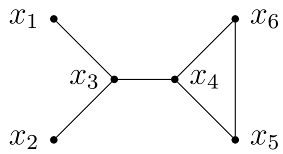

Figure 4.5: A Graph Γ

We have that $V = \{x_1, x_2, x_3, x_4, x_5, x_6\}$ the vertex list,
$E = \{\{x_1, x_3\}, \{x_2, x_3\}, \{x_3, x_4\}, \{x_4, x_5\}, \{x_4, x_6\}, \{x_5, x_6\}\}$ the edge list,
$L = V^{-1} \cup V = \{x_1^{-1}, x_2^{-1}, x_3^{-1}, x_4^{-1}, x_5^{-1}, x_6^{-1}, x_1, x_2, x_3, x_4, x_5, x_6\}$.

1. We find the star and the link of each vertex $x \in V$ as follows:

   (i) $st(x_1) = \{x_1, x_3\}$, $\quad \ell k(x_1) = \{x_3\}$.

   (ii) $st(x_2) = \{x_2, x_3\}$, $\quad \ell k(x_2) = \{x_3\}$.

   (iii) $st(x_3) = \{x_1, x_2, x_3, x_4\}$, $\quad \ell k(x_3) = \{x_1, x_2, x_4\}$.

   (iv) $st(x_4) = \{x_3, x_4, x_5, x_6\}$, $\quad \ell k(x_4) = \{x_3, x_5, x_6\}$.

   (v) $st(x_5) = \{x_4, x_5, x_6\}$, $\quad \ell k(x_5) = \{x_4, x_6\}$.

   (vi) $st(x_6) = \{x_4, x_5, x_6\}$, $\quad \ell k(x_6) = \{x_4, x_5\}$.

2. We find the equivelence classes for each vertex $x \in V$ as follows:

(i) $[x_1] = \{x_1, x_2\}$

(ii) $[x_2] = \{x_1, x_2\}$

(iii) $[x_3] = \{x_3\}$

(iv) $[x_4] = \{x_4\}$

(v) $[x_5] = \{x_5, x_6\}$

(vi) $[x_6] = \{x_5, x_6\}$

3. We find the connected components of each subgraph $\Gamma\backslash\{x_i\}$, where $x_i \in V$ and $i = 1, \ldots, 6$ as follows:

   (i) $\Gamma\backslash\{x_1\} = \{\{x_2, x_2^{-1}\}, \{x_4, x_5, x_6, x_4^{-1}, x_5^{-1}, x_6^{-1}\}\}$

   (ii) $\Gamma\backslash\{x_2\} = \{\{x_1, x_1^{-1}\}, \{x_4, x_5, x_6, x_4^{-1}, x_5^{-1}, x_6^{-1}\}\}$

   (iii) $\Gamma\backslash\{x_3\} = \{\{x_5, x_6, x_5^{-1}, x_6^{-1}\}\}$

   (iv) $\Gamma\backslash\{x_4\} = \{\{x_1, x_1^{-1}\}\{x_2, x_2^{-1}\}\}$

   (v) $\Gamma\backslash\{x_5\} = \{\{x_1, x_2, x_3, x_1^{-1}, x_2^{-1}, x_3^{-1}\}\}$

   (vi) $\Gamma\backslash\{6\} = \{\{x_1, x_2, x_3, x_1^{-1}, x_2^{-1}, x_3^{-1}\}\}$

4. We find the minimal connected components C of each subgraph $\Gamma\backslash\{x_i\}$, where $x_i \in V$ and $i = 1, \ldots, 6$, that is satisfies the condition that, for all $z \in V$ either

   (a) $[z] \cap C = \phi$; or

   (b) $[z] \subseteq C \cup st(x)$

   as follows:

   (i) The minimal connected components of $\Gamma\backslash st(x_1)$ are $\{\{x_2, x_2^{-1}\},$ $\{x_4, x_5, x_6, x_4^{-1}, x_5^{-1}, x_6^{-1}\}\}$.

   (ii) The minimal connected components of $\Gamma\backslash\{x_2\}$ are $\{\{x_1, x_1^{-1}\}, \{x_4,$ $x_5, x_6, x_4^{-1}, x_5^{-1}, x_6^{-1}\}\}$.

   (iii) The minimal connected components of $\Gamma\backslash\{x_3\}$ are $\{\{x_5, x_6, x_5^{-1}, x_6^{-1}\}\}$.

   (iv) The minimal connected components of $\Gamma\backslash\{x_4\}$ are $\{\{x_1, x_2, x_1^{-1}, x_2^{-1}\}\}$.

   (v) The minimal connected components of $\Gamma\backslash\{x_5\}$ are $\{\{x_1, x_2, x_3, x_1^{-1}, x_2^{-1},$ $x_3^{-1}\}\}$.

(vi) The minimal connected components of $\Gamma\backslash\{x_6\}$ are $\{\{x_1, x_2, x_3, x_1^{-1}, x_2^{-1}, x_3^{-1}\}\}$.

5. We find the union of the minimal connected components of $\Gamma\backslash\{x_i\}$, where $x_i \in V$ and $i = 1, \ldots, 6$ as follows:

$$\begin{aligned}\bigcup_{i=1}^{6} \Gamma\backslash\{x_i\} = \{&C_1 = \{x_2, x_2^{-1}\},\ C_2 = \{x_4, x_5, x_6, x_4^{-1}, x_5^{-1}, x_6^{-1}\},\\ &C_3 = \{x_1, x_1^{-1}\},\ C_4 = \{x_5, x_6, x_5^{-1}, x_6^{-1}\},\\ &C_5 = \{x_1, x_2, x_1^{-1}, x_2^{-1}\}, C_6 = \{x_1, x_2, x_3, x_1^{-1}, x_2^{-1}, x_3^{-1}\}\}.\end{aligned}$$

6. We find the partial conjugations automorphisms $\alpha_{C,x}$, where C is satisfies the condition in statement (4) above and $x \in L$. In fact these partial conjugations automorphisms form $Gens_1$ the first part of the generators of $Conj_V$. So

$Gens1 = \{f_1 = \alpha_{C_1,x_6^{-1}} = \{\{x_2, x_2^{-1}, x_6^{-1}\}, x_6^{-1}\},$

$f_2 = \alpha_{C_1,x_5^{-1}} = \{\{x_2, x_2^{-1}, x_5^{-1}\}, x_5^{-1}\},$

$f_3 = \alpha_{C_1,x_4^{-1}} = \{\{x_2, x_2^{-1}, x_4^{-1}\}, x_4^{-1}\},$

$f_4 = \alpha_{C_1,x_3^{-1}} = \{\{x_2, x_2^{-1}, x_3^{-1}\}, x_3^{-1}\},$

$f_5 = \alpha_{C_1,x_1^{-1}} = \{\{x_2, x_2^{-1}, x_1^{-1}\}, x_1^{-1}\},$

$f_6 = \alpha_{C_1,x_1} = \{\{x_2, x_2^{-1}, x_1\}, x_1\},$

$f_7 = \alpha_{C_1,x_3} = \{\{x_2, x_2^{-1}, x_3\}, x_3\},$

$f_8 = \alpha_{C_1,x_4} = \{\{x_2, x_2^{-1}, x_4\}, x_4\},$

$f_9 = \alpha_{C_1,x_5} = \{\{x_2, x_2^{-1}, x_5\}, x_5\},$

$f_{10} = \alpha_{C_1,x_6} = \{\{x_2, x_2^{-1}, x_6\}, x_6\},$

$f_{11} = \alpha_{C_2,x_3^{-1}} = \{\{x_4, x_5, x_6, x_4^{-1}, x_5^{-1}, x_6^{-1}, x_3^{-1}\}, x_3^{-1}\},$

$f_{12} = \alpha_{C_2,x_2^{-1}} = \{\{x_4, x_5, x_6, x_4^{-1}, x_5^{-1}, x_6^{-1}, x_2^{-1}\}, x_2^{-1}\},$

$f_{13} = \alpha_{C_2,x_1^{-1}} = \{\{x_4, x_5, x_6, x_4^{-1}, x_5^{-1}, x_6^{-1}, x_1^{-1}\}, x_1^{-1}\},$

$f_{14} = \alpha_{C_1,x_1} = \{\{x_4, x_5, x_6, x_4^{-1}, x_5^{-1}, x_6^{-1}, x_1\}, x_1\},$

$f_{15} = \alpha_{C_1,x_2} = \{\{x_4, x_5, x_6, x_4^{-1}, x_5^{-1}, x_6^{-1}, x_2\}, x_2\},$

$f_{16} = \alpha_{C_1,x_3} = \{\{x_4, x_5, x_6, x_4^{-1}, x_5^{-1}, x_6^{-1}, x_3\}, x_3\},$

$f_{17} = \alpha_{C_3,x_6^{-1}} = \{\{x_1, x_1^{-1}, x_6^{-1}\}, x_6^{-1}\},$

$$f_{18} = \alpha_{C_3,x_5^{-1}} = \{\{x_1, x_1^{-1}, x_5^{-1}\}, x_5^{-1}\},$$

$$f_{19} = \alpha_{C_3,x_4^{-1}} = \{\{x_1, x_1^{-1}, x_4^{-1}\}, x_4^{-1}\},$$

$$f_{20} = \alpha_{C_3,x_3^{-1}} = \{\{x_1, x_1^{-1}, x_3^{-1}\}, x_3^{-1}\},$$

$$f_{21} = \alpha_{C_3,x_2^{-1}} = \{\{x_1, x_1^{-1}, x_2^{-1}\}, x_2^{-1}\},$$

$$f_{22} = \alpha_{C_3,x_2} = \{\{x_1, x_1^{-1}, x_2\}, x_2\},$$

$$f_{23} = \alpha_{C_3,x_3} = \{\{x_1, x_1^{-1}, x_3\}, x_3\},$$

$$f_{24} = \alpha_{C_3,x_4} = \{\{x_1, x_1^{-1}, x_4\}, x_4\},$$

$$f_{25} = \alpha_{C_3,x_5} = \{\{x_1, x_1^{-1}, x_5\}, x_5\},$$

$$f_{26} = \alpha_{C_3}, x_6 = \{\{x_1, x_1^{-1}, x_6\}, x_6\},$$

$$f_{27} = \alpha_{C_4}, x_4^{-1} = \{\{x_5, x_6, x_5^{-1}, x_6^{-1}, x_4^{-1}\}, x_4^{-1}\},$$

$$f_{28} = \alpha_{C_4}, x_3 = \{\{x_5, x_6, x_5^{-1}, x_6^{-1}, x_3^{-1}\}, x_3^{-1}\},$$

$$f_{29} = \alpha_{C_4}, x_2^{-1} = \{\{x_5, x_6, x_5^{-1}, x_6^{-1}, x_2^{-1}\}, x_2^{-1}\},$$

$$f_{30} = \alpha_{C_4}, x_1^{-1} = \{\{x_5, x_6, x_5^{-1}, x_6^{-1}, x_1^{-1}\}, x_1^{-1}\},$$

$$f_{31} = \alpha_{C_4}, x_1 = \{\{x_5, x_6, x_5^{-1}, x_6^{-1}, x_1\}, x_1\},$$

$$f_{32} = \alpha_{C_4}, x_2 = \{\{x_5, x_6, x_5^{-1}, x_6^{-1}, x_2\}, x_2\},$$

$$f_{33} = \alpha_{C_4}, x_3 = \{\{x_5, x_6, x_5^{-1}, x_6^{-1}, x_3\}, x_3\},$$

$$f_{34} = \alpha_{C_4}, x_4 = \{\{x_5, x_6, x_5^{-1}, x_6^{-1}, x_4\}, x_4\},$$

$$f_{35} = \alpha_{C_5}, x_6^{-1} = \{\{x_1, x_2, x_1^{-1}, x_2^{-1}, x_6^{-1}\}, x_6^{-1}\},$$

$$f_{36} = \alpha_{C_5}, x_5^{-1} = \{\{x_1, x_2, x_1^{-1}, x_2^{-1}, x_5^{-1}\}, x_5^{-1}\},$$

$$f_{37} = \alpha_{C_5}, x_4^{-1} = \{\{x_1, x_2, x_1^{-1}, x_2^{-1}, x_4^{-1}\}, x_4^{-1}\},$$

$$f_{38} = \alpha_{C_5}, x_3^{-1} = \{\{x_1, x_2, x_1^{-1}, x_2^{-1}, x_3^{-1}\}, x_3^{-1}\},$$

$$f_{39} = \alpha_{C_5}, x_3 = \{\{x_1, x_2, x_1^{-1}, x_2^{-1}, x_3\}, x_3\},$$

$$f_{40} = \alpha_{C_5}, x_4 = \{\{x_1, x_2, x_1^{-1}, x_2^{-1}, x_4\}, x_4\},$$

$$f_{41} = \alpha_{C_5}, x_5 = \{\{x_1, x_2, x_1^{-1}, x_2^{-1}, x_5\}, x_5\},$$

$$f_{42} = \alpha_{C_5}, x_6 = \{\{x_1, x_2, x_1^{-1}, x_2^{-1}, x_6\}, x_6\},$$

$$f_{43} = \alpha_{C_6}, x_6^{-1} = \{\{x_1, x_2, x_3, x_1^{-1}, x_2^{-1}, x_3^{-1}, x_6^{-1}\}, x_6^{-1}\},$$

$$f_{44} = \alpha_{C_6}, x_5^{-1} = \{\{x_1, x_2, x_3, x_1^{-1}, x_2^{-1}, x_3^{-1}, x_5^{-1}\}, x_5^{-1}\},$$

$$f_{45} = \alpha_{C_6}, x_4^{-1} = \{\{x_1, x_2, x_3, x_1^{-1}, x_2^{-1}, x_3^{-1}, x_4^{-1}\}, x_4^{-1}\},$$

$f_{46} = \alpha_{C_6}, x_4 = \{\{x_1, x_2, x_3, x_1^{-1}, x_2^{-1}, x_3^{-1}, x_4\}, x_4\},$

$f_{47} = \alpha_{C_6}, x_5 = \{\{x_1, x_2, x_3, x_1^{-1}, x_2^{-1}, x_3^{-1}, x_5\}, x_5\},$

$f_{48} = \alpha_{C_6}, x_6 = \{\{x_1, x_2, x_3, x_1^{-1}, x_2^{-1}, x_3^{-1}, x_6\}, x_6\}\}.$

7. We find the inner automorphisms $\alpha_{C,x}$, where C is satisfies the condition in statement (4) above and $x \in L$. In fact these inner automorphisms which are also partial conjugations automorphisms form $Gens_2$ the second part of the generators of $Conj_V$.

   $Gens_2 = \{w_1 = \{\{x_6^{-1}, x_5^{-1}, x_4^{-1}, x_2^{-1}, x_1^{-1}, x_2, x_4, x_5, x_6\}, x_1^{-1}\},$

   $w_2 = \{\{x_6^{-1}, x_5^{-1}, x_4^{-1}, x_2^{-1}, x_1, x_2, x_4, x_5, x_6\}, x_1\},$

   $w_3 = \{\{x_6^{-1}, x_5^{-1}, x_4^{-1}, x_2^{-1}, x_1^{-1}, x_1, x_4, x_5, x_6\}, x_2^{-1}\},$

   $w_4 = \{\{x_6^{-1}, x_5^{-1}, x_4^{-1}, x_2^{-1}, x_1^{-1}, x_2, x_4, x_5, x_6\}, x_1^{-1}\},$

   $w_5 = \{\{x_6^{-1}, x_5^{-1}, x_4^{-1}, x_2^{-1}, x_1, x_2, x_4, x_5, x_6\}, x_1\},$

   $w_6 = \{\{x_6^{-1}, x_5^{-1}, x_4^{-1}, x_1^{-1}, x_1, x_2, x_4, x_5, x_6\}, x_2\},$

   $w_7 = \{\{x_6^{-1}, x_5^{-1}, x_4^{-1}, x_2^{-1}, x_1^{-1}, x_1, x_4, x_5, x_6\}, x_2^{-1}\},$

   $w_8 = \{\{x_6^{-1}, x_5^{-1}, x_4^{-1}, x_1^{-1}, x_1, x_2, x_4, x_5, x_6\}, x_2\},$

   $w_9 = \{\{x_6^{-1}, x_5^{-1}, x_4^{-1}, x_2^{-1}, x_1^{-1}, x_1, x_4, x_5, x_6\}, x_2^{-1}\},$

   $w_{10} = \{\{x_6^{-1}, x_5^{-1}, x_4^{-1}, x_2^{-1}, x_1^{-1}, x_1, x_4, x_5, x_6\}, x_1^{-1}\},$

   $w_{11} = \{\{x_6^{-1}, x_5^{-1}, x_4^{-1}, x_2^{-1}, x_1, x_2, x_4, x_5, x_6\}, x_1\},$

   $w_{12} = \{\{x_6^{-1}, x_5^{-1}, x_4^{-1}, x_1^{-1}, x_1, x_2, x_4, x_5, x_6\}, x_2\}\}$

8. We find $Gens$ the set of the generators of the subgroup $Conj_V$ as follows:

   $$\begin{aligned} Gens &= Gens_1 \cup Gens_2 \\ &= \{f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8, f_9, f_{10}, f_{11}, f_{12}, f_{13}, f_{14}, f_{15}, f_{16}, f_{17}, f_{18}, f_{19}, \\ &\quad f_{20}, f_{21}, f_{22}, f_{23}, f_{24}, f_{25}, f_{26}, f_{27}, f_{28}, f_{29}, f_{30}, f_{31}, f_{32}, f_{33}, f_{34}, f_{35}, f_{36}, \\ &\quad f_{37}, f_{38}, f_{39}, f_{40}, f_{41}, f_{42}, f_{43}, f_{44}, f_{45}, f_{46}, f_{47}, f_{48}, w_1, w_2, w_3, w_4, w_5, \\ &\quad w_6, w_7, w_8, w_9, w_{10}, w_{11}, w_{12}\}. \end{aligned}$$

9. We find $Rels$ the set of the relations according to Theorem 4.3.15 as follows:

   $Rels1 = \{f_1 * f_{10}, f_2 * f_9, f_3 * f_8, f_4 * f_7, f_5 * f_6, f_6 * f_5, f_7 * f_4, f_8 * f_3, f_9 * f_2, f_{10} * f_1, f_{11} * f_{16}, f_{12} * f_{15}, f_{13} * f_{14}, f_{14} * f_{13}, f_{15} * f_{12}, f_{16} * f_{11}, f_{17} * f_{26}, f_{18} * f_{25}, f_{19} *$

$f_{24}, f_{20}*f_{23}, f_{21}*f_{22}, f_{22}*f_{21}, f_{23}*f_{20}, f_{24}*f_{19}, f_{25}*f_{18}, f_{26}*f_{17}, f_{27}*f_{34}, f_{28}*f_{33}, f_{29}*f_{32}, f_{30}*f_{31}, f_{31}*f_{30}, f_{32}*f_{29}, f_{33}*f_{28}, f_{34}*f_{27}, f_{35}*f_{42}, f_{36}*f_{41}, f_{37}*f_{40}, f_{38}*f_{39}, f_{39}*f_{38}, f_{40}*f_{37}, f_{41}*f_{36}, f_{42}*f_{35}, f_{43}*f_{48}, f_{44}*f_{47}, f_{45}*f_{46}, f_{46}*f_{45}, f_{47}*f_{44}, f_{48}*f_{43}\}$.

$Rels2 = \{f_1*f_{17}*f_{42}, f_2*f_{18}*f_{41}, f_3*f_{19}*f_{40}, f_3*f_{27}*f_8, f_4*f_{11}*f_{33}, f_4*f_{28}*f_{33}, f_7*f_{16}*f_{28}, f_7*f_{33}*f_{28}, f_8*f_{24}*f_{37}, f_8*f_{34}*f_3, f_9*f_{25}*f_{36}, f_{10}*f_{26}*f_{35}, f_{11}*f_{20}*f_{33}, f_{11}*f_{38}*f_{33}, f_{16}*f_{23}*f_{28}, f_{16}*f_{39}*f_{28}, f_{19}*f_{27}*f_{24}, f_{20}*f_{28}*f_{33}, f_{23}*f_{33}*f_{28}, f_{24}*f_{34}*f_{19}, f_{27}*f_{37}*f_{40}, f_{27}*f_{45}*f_{40}, f_{28}*f_{38}*f_{33}, f_{33}*f_{39}*f_{28}, f_{34}*f_{40}*f_{37}, f_{34}*f_{46}*f_{37}\}$.

$Rels3 = \{f_1*f_2*f_{10}*f_9, f_1*f_3*f_{10}*f_8, f_1*f_8*f_{10}*f_3, f_1*f_9*f_{10}*f_2, f_1*f_{18}*f_{26}*f_9, f_1*f_{19}*f_{26}*f_8, f_1*f_{20}*f_{26}*f_7, f_1*f_{23}*f_{26}*f_4, f_1*f_{24}*f_{26}*f_3, f_1*f_{25}*f_{26}*f_2, f_1*f_{36}*f_{42}*f_9, f_1*f_{37}*f_{42}*f_8, f_1*f_{40}*f_{42}*f_3, f_1*f_{41}*f_{42}*f_2, f_1*f_{44}*f_{48}*f_9, f_1*f_{45}*f_{48}*f_8, f_1*f_{46}*f_{48}*f_3, f_1*f_{47}*f_{48}*f_2, f_2*f_3*f_9*f_8, f_2*f_8*f_9*f_3, f_2*f_{10}*f_9*f_1, f_2*f_{17}*f_{25}*f_{10}, f_2*f_{19}*f_{25}*f_8, f_2*f_{20}*f_{25}*f_7, f_2*f_{23}*f_{25}*f_4, f_2*f_{24}*f_{25}*f_3, f_2*f_{26}*f_{25}*f_1, f_2*f_{35}*f_{41}*f_{10}, f_2*f_{37}*f_{41}*f_8, f_2*f_{40}*f_{41}*f_3, f_2*f_{42}*f_{41}*f_1, f_2*f_{43}*f_{47}*f_{10}, f_2*f_{45}*f_{47}*f_8, f_2*f_{46}*f_{47}*f_3, f_2*f_{48}*f_{47}*f_1, f_3*f_4*f_8*f_7, f_3*f_7*f_8*f_4, f_3*f_9*f_8*f_2, f_3*f_{10}*f_8*f_1, f_3*f_{17}*f_{24}*f_{10}, f_3*f_{18}*f_{24}*f_9, f_3*f_{20}*f_{24}*f_7, f_3*f_{23}*f_{24}*f_4, f_3*f_{25}*f_{24}*f_2, f_3*f_{26}*f_{24}*f_1, f_3*f_{28}*f_{34}*f_7, f_3*f_{30}*f_{34}*f_6, f_3*f_{31}*f_{34}*f_5, f_3*f_{33}*f_{34}*f_4, f_3*f_{35}*f_{40}*f_{10}, f_3*f_{36}*f_{40}*f_9, f_3*f_{38}*f_{40}*f_7, f_3*f_{39}*f_{40}*f_4, f_3*f_{41}*f_{40}*f_2, f_3*f_{42}*f_{40}*f_1, f_3*f_{43}*f_{46}*f_{10}, f_3*f_{44}*f_{46}*f_9, f_3*f_{47}*f_{46}*f_2, f_3*f_{48}*f_{46}*f_1, f_4*f_5*f_7*f_6, f_4*f_6*f_7*f_5, f_4*f_8*f_7*f_3, f_4*f_{13}*f_{16}*f_6, f_4*f_{14}*f_{16}*f_5, f_4*f_{17}*f_{23}*f_{10}, f_4*f_{18}*f_{23}*f_9, f_4*f_{19}*f_{23}*f_8, f_4*f_{24}*f_{23}*f_3, f_4*f_{25}*f_{23}*f_2, f_4*f_{26}*f_{23}*f_1, f_4*f_{27}*f_{33}*f_8, f_4*f_{30}*f_{33}*f_6, f_4*f_{31}*f_{33}*f_5, f_4*f_{34}*f_{33}*f_3, f_4*f_{37}*f_{39}*f_8, f_4*f_{40}*f_{39}*f_3, f_5*f_7*f_6*f_4, f_5*f_{11}*f_{14}*f_7, f_5*f_{16}*f_{14}*f_4, f_5*f_{27}*f_{31}*f_8, f_5*f_{28}*f_{31}*f_7, f_5*f_{33}*f_{31}*f_4, f_5*f_{34}*f_{31}*f_3, f_6*f_7*f_5*f_4, f_6*f_{11}*f_{13}*f_7, f_6*f_{16}*f_{13}*f_4, f_6*f_{27}*f_{30}*f_8, f_6*f_{28}*f_{30}*f_7, f_6*f_{33}*f_{30}*f_4, f_6*f_{34}*f_{30}*f_3, f_7*f_8*f_4*f_3, f_7*f_{13}*f_{11}*f_6, f_7*f_{14}*f_{11}*f_5, f_7*f_{17}*f_{20}*f_{10}, f_7*f_{18}*f_{20}*f_9, f_7*f_{19}*f_{20}*f_8, f_7*f_{24}*f_{20}*f_3, f_7*f_{25}*f_{20}*f_2, f_7*f_{26}*f_{20}*f_1, f_7*f_{27}*f_{28}*f_8, f_7*f_{30}*f_{28}*f_6, f_7*f_{31}*f_{28}*f_5, f_7*f_{34}*f_{28}*f_3, f_7*f_{37}*f_{38}*f_8, f_7*f_{40}*f_{38}*f_3, f_8*f_9*f_3*f_2, f_8*f_{10}*f_3*f_1, f_8*f_{17}*f_{19}*f_{10}, f_8*f_{18}*f_{19}*f_9, f_8*f_{20}*f_{19}*f_7, f_8*f_{23}*f_{19}*f_4, f_8*f_{25}*f_{19}*f_2, f_8*f_{26}*f_{19}*f_1, f_8*f_{28}*f_{27}*f_7, f_8*f_{30}*f_{27}*f_6, f_8*f_{31}*f_{27}*f_5, f_8*f_{33}*f_{27}*f_4, f_8*f_{35}*f_{37}*f_{10}, f_8*f_{36}*f_{37}*f_9, f_8*f_{38}*f_{37}*f_7, f_8*f_{39}*f_{37}*f_4, f_8*f_{41}*f_{37}*f_2, f_8*f_{42}*f_{37}*f_1, f_8*f_{43}*f_{45}*f_{10}, f_8*f_{44}*f_{45}*f_9, f_8*f_{47}*f_{45}*f_2, f_8*f_{48}*f_{45}*f_1, f_9*f_{10}*f_2*f_1, f_9*f_{17}*f_{18}*f_{10}, f_9*f_{19}*f_{18}*$

$f_8, f_9*f_{20}*f_{18}*f_7, f_9*f_{23}*f_{18}*f_4, f_9*f_{24}*f_{18}*f_3, f_9*f_{26}*f_{18}*f_1, f_9*f_{35}*f_{36}*$
$f_{10}, f_9*f_{37}*f_{36}*f_8, f_9*f_{40}*f_{36}*f_3, f_9*f_{42}*f_{36}*f_1, f_9*f_{43}*f_{44}*f_{10}, f_9*f_{45}*f_{44}*$
$f_8, f_9*f_{46}*f_{44}*f_3, f_9*f_{48}*f_{44}*f_1, f_{10}*f_{18}*f_{17}*f_9, f_{10}*f_{19}*f_{17}*f_8, f_{10}*f_{20}*f_{17}*$
$f_7, f_{10}*f_{23}*f_{17}*f_4, f_{10}*f_{24}*f_{17}*f_3, f_{10}*f_{25}*f_{17}*f_2, f_{10}*f_{36}*f_{35}*f_9, f_{10}*f_{37}*f_{35}*$
$f_8, f_{10}*f_{40}*f_{35}*f_3, f_{10}*f_{41}*f_{35}*f_2, f_{10}*f_{44}*f_{43}*f_9, f_{10}*f_{45}*f_{43}*f_8, f_{10}*f_{46}*$
$f_{43}*f_3, f_{10}*f_{47}*f_{43}*f_2, f_{11}*f_{12}*f_{16}*f_{15}, f_{11}*f_{13}*f_{16}*f_{14}, f_{11}*f_{14}*f_{16}*f_{13}, f_{11}*$
$f_{15}*f_{16}*f_{12}, f_{11}*f_{21}*f_{23}*f_{15}, f_{11}*f_{22}*f_{23}*f_{12}, f_{11}*f_{29}*f_{33}*f_{15}, f_{11}*f_{30}*f_{33}*$
$f_{14}, f_{11}*f_{31}*f_{33}*f_{13}, f_{11}*f_{32}*f_{33}*f_{12}, f_{12}*f_{16}*f_{15}*f_{11}, f_{12}*f_{20}*f_{22}*f_{16}, f_{12}*$
$f_{23}*f_{22}*f_{11}, f_{12}*f_{28}*f_{32}*f_{16}, f_{12}*f_{33}*f_{32}*f_{11}, f_{13}*f_{16}*f_{14}*f_{11}, f_{13}*f_{28}*f_{31}*f_{16},$
$f_{13}*f_{33}*f_{31}*f_{11}, f_{14}*f_{16}*f_{13}*f_{11}, f_{14}*f_{28}*f_{30}*f_{16}, f_{14}*f_{33}*f_{30}*f_{11}, f_{15}*f_{16}*$
$f_{12}*f_{11}, f_{15}*f_{20}*f_{21}*f_{16}, f_{15}*f_{23}*f_{21}*f_{11}, f_{15}*f_{28}*f_{29}*f_{16}, f_{15}*f_{33}*f_{29}*f_{11}, f_{16}*$
$f_{21}*f_{20}*f_{15}, f_{16}*f_{22}*f_{20}*f_{12}, f_{16}*f_{29}*f_{28}*f_{15}, f_{16}*f_{30}*f_{28}*f_{14}, f_{16}*f_{31}*f_{28}*$
$f_{13}, f_{16}*f_{32}*f_{28}*f_{12}, f_{17}*f_{18}*f_{26}*f_{25}, f_{17}*f_{19}*f_{26}*f_{24}, f_{17}*f_{24}*f_{26}*f_{19}, f_{17}*f_{25}*$
$f_{26}*f_{18}, f_{17}*f_{36}*f_{42}*f_{25}, f_{17}*f_{37}*f_{42}*f_{24}, f_{17}*f_{40}*f_{42}*f_{19}, f_{17}*f_{41}*f_{42}*f_{18}, f_{17}*$
$f_{44}*f_{48}*f_{25}, f_{17}*f_{45}*f_{48}*f_{24}, f_{17}*f_{46}*f_{48}*f_{19}, f_{17}*f_{47}*f_{48}*f_{18}, f_{18}*f_{19}*f_{25}*$
$f_{24}, f_{18}*f_{24}*f_{25}*f_{19}, f_{18}*f_{26}*f_{25}*f_{17}, f_{18}*f_{35}*f_{41}*f_{26}, f_{18}*f_{37}*f_{41}*f_{24}, f_{18}*f_{40}*$
$f_{41}*f_{19}, f_{18}*f_{42}*f_{41}*f_{17}, f_{18}*f_{43}*f_{47}*f_{26}, f_{18}*f_{45}*f_{47}*f_{24}, f_{18}*f_{46}*f_{47}*f_{19}, f_{18}*$
$f_{48}*f_{47}*f_{17}, f_{19}*f_{20}*f_{24}*f_{23}, f_{19}*f_{23}*f_{24}*f_{20}, f_{19}*f_{25}*f_{24}*f_{18}, f_{19}*f_{26}*f_{24}*$
$f_{17}, f_{19}*f_{28}*f_{34}*f_{23}, f_{19}*f_{29}*f_{34}*f_{22}, f_{19}*f_{32}*f_{34}*f_{21}, f_{19}*f_{33}*f_{34}*f_{20}, f_{19}*f_{35}*$
$f_{40}*f_{26}, f_{19}*f_{36}*f_{40}*f_{25}, f_{19}*f_{38}*f_{40}*f_{23}, f_{19}*f_{39}*f_{40}*f_{20}, f_{19}*f_{41}*f_{40}*f_{18}, f_{19}*$
$f_{42}*f_{40}*f_{17}, f_{19}*f_{43}*f_{46}*f_{26}, f_{19}*f_{44}*f_{46}*f_{25}, f_{19}*f_{47}*f_{46}*f_{18}, f_{19}*f_{48}*f_{46}*$
$f_{17}, f_{20}*f_{21}*f_{23}*f_{22}, f_{20}*f_{22}*f_{23}*f_{21}, f_{20}*f_{24}*f_{23}*f_{19}, f_{20}*f_{27}*f_{33}*f_{24}, f_{20}*f_{29}*$
$f_{33}*f_{22}, f_{20}*f_{32}*f_{33}*f_{21}, f_{20}*f_{34}*f_{33}*f_{19}, f_{20}*f_{37}*f_{39}*f_{24}, f_{20}*f_{40}*f_{39}*f_{19}, f_{21}*$
$f_{23}*f_{22}*f_{20}, f_{21}*f_{27}*f_{32}*f_{24}, f_{21}*f_{28}*f_{32}*f_{23}, f_{21}*f_{33}*f_{32}*f_{20}, f_{21}*f_{34}*f_{32}*$
$f_{19}, f_{22}*f_{23}*f_{21}*f_{20}, f_{22}*f_{27}*f_{29}*f_{24}, f_{22}*f_{28}*f_{29}*f_{23}, f_{22}*f_{33}*f_{29}*f_{20}, f_{22}*f_{34}*$
$f_{29}*f_{19}, f_{23}*f_{24}*f_{20}*f_{19}, f_{23}*f_{27}*f_{28}*f_{24}, f_{23}*f_{29}*f_{28}*f_{22}, f_{23}*f_{32}*f_{28}*f_{21}, f_{23}*$
$f_{34}*f_{28}*f_{19}, f_{23}*f_{37}*f_{38}*f_{24}, f_{23}*f_{40}*f_{38}*f_{19}, f_{24}*f_{25}*f_{19}*f_{18}, f_{24}*f_{26}*f_{19}*$
$f_{17}, f_{24}*f_{28}*f_{27}*f_{23}, f_{24}*f_{29}*f_{27}*f_{22}, f_{24}*f_{32}*f_{27}*f_{21}, f_{24}*f_{33}*f_{27}*f_{20}, f_{24}*f_{35}*$
$f_{37}*f_{26}, f_{24}*f_{36}*f_{37}*f_{25}, f_{24}*f_{38}*f_{37}*f_{23}, f_{24}*f_{39}*f_{37}*f_{20}, f_{24}*f_{41}*f_{37}*f_{18}, f_{24}*$
$f_{42}*f_{37}*f_{17}, f_{24}*f_{43}*f_{45}*f_{26}, f_{24}*f_{44}*f_{45}*f_{25}, f_{24}*f_{47}*f_{45}*f_{18}, f_{24}*f_{48}*f_{45}*$
$f_{17}, f_{25}*f_{26}*f_{18}*f_{17}, f_{25}*f_{35}*f_{36}*f_{26}, f_{25}*f_{37}*f_{36}*f_{24}, f_{25}*f_{40}*f_{36}*f_{19}, f_{25}*f_{42}*$
$f_{36}*f_{17}, f_{25}*f_{43}*f_{44}*f_{26}, f_{25}*f_{45}*f_{44}*f_{24}, f_{25}*f_{46}*f_{44}*f_{19}, f_{25}*f_{48}*f_{44}*f_{17}, f_{26}*$
$f_{36}*f_{35}*f_{25}, f_{26}*f_{37}*f_{35}*f_{24}, f_{26}*f_{40}*f_{35}*f_{19}, f_{26}*f_{41}*f_{35}*f_{18}, f_{26}*f_{44}*f_{43}*$
$f_{25}, f_{26}*f_{45}*f_{43}*f_{24}, f_{26}*f_{46}*f_{43}*f_{19}, f_{26}*f_{47}*f_{43}*f_{18}, f_{27}*f_{28}*f_{34}*f_{33}, f_{27}*$
$f_{33}*f_{34}*f_{28}, f_{27}*f_{38}*f_{40}*f_{33}, f_{27}*f_{39}*f_{40}*f_{28}, f_{28}*f_{29}*f_{33}*f_{32}, f_{28}*f_{30}*f_{33}*f_{31},$

$f_{28} * f_{31} * f_{33} * f_{30}, f_{28} * f_{32} * f_{33} * f_{29}, f_{28} * f_{34} * f_{33} * f_{27}, f_{28} * f_{37} * f_{39} * f_{34}, f_{28} * f_{40} * f_{39} * f_{27}, f_{29} * f_{33} * f_{32} * f_{28}, f_{30} * f_{33} * f_{31} * f_{28}, f_{31} * f_{33} * f_{30} * f_{28}, f_{32} * f_{33} * f_{29} * f_{28}, f_{33} * f_{34} * f_{28} * f_{27}, f_{33} * f_{37} * f_{38} * f_{34}, f_{33} * f_{40} * f_{38} * f_{27}, f_{34} * f_{38} * f_{37} * f_{33}, f_{34} * f_{39} * f_{37} * f_{28}, f_{35} * f_{36} * f_{42} * f_{41}, f_{35} * f_{37} * f_{42} * f_{40}, f_{35} * f_{40} * f_{42} * f_{37}, f_{35} * f_{41} * f_{42} * f_{36}, f_{35} * f_{44} * f_{48} * f_{41}, f_{35} * f_{45} * f_{48} * f_{40}, f_{35} * f_{46} * f_{48} * f_{37}, f_{35} * f_{47} * f_{48} * f_{36}, f_{36} * f_{37} * f_{41} * f_{40}, f_{36} * f_{40} * f_{41} * f_{37}, f_{36} * f_{42} * f_{41} * f_{35}, f_{36} * f_{43} * f_{47} * f_{42}, f_{36} * f_{45} * f_{47} * f_{40}, f_{36} * f_{46} * f_{47} * f_{37}, f_{36} * f_{48} * f_{47} * f_{35}, f_{37} * f_{38} * f_{40} * f_{39}, f_{37} * f_{39} * f_{40} * f_{38}, f_{37} * f_{41} * f_{40} * f_{36}, f_{37} * f_{42} * f_{40} * f_{35}, f_{37} * f_{43} * f_{46} * f_{42}, f_{37} * f_{44} * f_{46} * f_{41}, f_{37} * f_{47} * f_{46} * f_{36}, f_{37} * f_{48} * f_{46} * f_{35}, f_{38} * f_{40} * f_{39} * f_{37}, f_{39} * f_{40} * f_{38} * f_{37}, f_{40} * f_{41} * f_{37} * f_{36}, f_{40} * f_{42} * f_{37} * f_{35}, f_{40} * f_{43} * f_{45} * f_{42}, f_{40} * f_{44} * f_{45} * f_{41}, f_{40} * f_{47} * f_{45} * f_{36}, f_{40} * f_{48} * f_{45} * f_{35}, f_{41} * f_{42} * f_{36} * f_{35}, f_{41} * f_{43} * f_{44} * f_{42}, f_{41} * f_{45} * f_{44} * f_{40}, f_{41} * f_{46} * f_{44} * f_{37}, f_{41} * f_{48} * f_{44} * f_{35}, f_{42} * f_{44} * f_{43} * f_{41}, f_{42} * f_{45} * f_{43} * f_{40}, f_{42} * f_{46} * f_{43} * f_{37}, f_{42} * f_{47} * f_{43} * f_{36}, f_{43} * f_{44} * f_{48} * f_{47}, f_{43} * f_{45} * f_{48} * f_{46}, f_{43} * f_{46} * f_{48} * f_{45}, f_{43} * f_{47} * f_{48} * f_{44}, f_{44} * f_{45} * f_{47} * f_{46}, f_{44} * f_{46} * f_{47} * f_{45}, f_{44} * f_{48} * f_{47} * f_{43}, f_{45} * f_{47} * f_{46} * f_{44}, f_{45} * f_{48} * f_{46} * f_{43}, f_{46} * f_{47} * f_{45} * f_{44}, f_{46} * f_{48} * f_{45} * f_{43}, f_{47} * f_{48} * f_{44} * f_{43}\}.$

$Rels4 = \{w_1 * f_1 w_{11} * f_{10}, w_2 * f_1 * w_{10} * f_{10}, w_4 * f_1 * w_{11} * f_{10}, w_5 * f_1 * w_{10} * f_{10}, w_{10} * f_1 * w_{11} * f_{10}, w_{11} * f_1 * w_{10} * f_{10}, w_1 * f_2 * w_{11} * f_9, w_2 * f_2 * w_{10} * f_9, w_4 * f_2 * w_{11} * f_9, w_5 * f_2 * w_{10} * f_9, w_{10} * f_2 * w_{11} * f_9, w_{11} * f_2 * w_{10} * f_9, w_1 * f_3 * w_{11} * f_8, w_2 * f_3 * w_{10} * f_8, w_4 * f_3 * w_{11} * f_8, w_5 * f_3 * w_{10} * f_8, w_{10} * f_3 * w_{11} * f_8, w_{11} * f_3 * w_{10} * f_8, w_1 * f_4 * w_{11} * f_7, w_2 * f_4 * w_{10} * f_7, w_4 * f_4 * w_{11} * f_7, w_5 * f_4 * w_{10} * f_7, w_{10} * f_4 * w_{11} * f_7, w_{11} * f_4 * w_{10} * f_7, w_1 * f_7 * w_{11} * f_4, w_2 * f_7 * w_{10} * f_4, w_4 * f_7 * w_{11} * f_4, w_5 * f_7 * w_{10} * f_4, w_{10} * f_7 * w_{11} * f_4, w_{11} * f_7 * w_{10} * f_4, w_1 * f_8 * w_{11} * f_3, w_2 * f_8 * w_{10} * f_3, w_4 * f_8 * w_{11} * f_3, w_5 * f_8 * w_{10} * f_3, w_{10} * f_8 * w_{11} * f_3, w_{11} * f_8 * w_{10} * f_3, w_1 * f_9 * w_{11} * f_2, w_2 * f_9 * w_{10} * f_2, w_4 * f_9 * w_{11} * f_2, w_5 * f_9 * w_{10} * f_2, w_{10} * f_9 * w_{11} * f_2, w_{11} * f_9 * w_{10} * f_2, w_1 * f_{10} * w_{11} * f_1, w_2 * f_{10} * w_{10} * f_1, w_4 * f_{10} * w_{11} * f_1, w_5 * f_{10} * w_{10} * f_1, w_{10} * f_{10} * w_{11} * f_1, w_{11} * f_{10} * w_{10} * f_1, w_1 * f_{11} * w_{11} * f_{16}, w_2 * f_{11} * w_{10} * f_{16}, w_3 * f_{11} * w_{12} * f_{16}, w_4 * f_{11} * w_{11} * f_{16}, w_5 * f_{11} * w_{10} * f_{16}, w_6 * f_{11} * w_9 * f_{16}, w_7 * f_{11} * w_{12} * f_{16}, w_8 * f_{11} * w_9 * f_{16}, w_9 * f_{11} * w_{12} * f_{16}, w_{10} * f_{11} * w_{11} * f_{16}, w_{11} * f_{11} * w_{10} * f_{16}, w_{12} * f_{11} * w_9 * f_{16}, w_1 * f_{12} * w_{11} * f_{15}, w_2 * f_{12} * w_{10} * f_{15}, w_4 * f_{12} * w_{11} * f_{15}, w_5 * f_{12} * w_{10} * f_{15}, w_{10} * f_{12} * w_{11} * f_{15}, w_{11} * f_{12} * w_{10} * f_{15}, w_3 * f_{13} * w_{12} * f_{14}, w_6 * f_{13} * w_9 * f_{14}, w_7 * f_{13} * w_{12} * f_{14}, w_8 * f_{13} * w_9 * f_{14}, w_9 * f_{13} * w_{12} * f_{14}, w_{12} * f_{13} * w_9 * f_{14}, w_3 * f_{14} * w_{12} * f_{13}, w_6 * f_{14} * w_9 * f_{13}, w_7 * f_{14} * w_{12} * f_{13}, w_8 * f_{14} * w_9 * f_{13}, w_9 * f_{14} * w_{12} * f_{13}, w_{12} * f_{14} * w_9 * f_{13}, w_1 * f_{15} * w_{11} * f_{12}, w_2 * f_{15} * w_{10} * f_{12}, w_4 * f_{15} * w_{11} * f_{12}, w_5 * f_{15} * w_{10} * f_{12}, w_{10} * f_{15} * w_{11} * f_{12}, w_{11} * f_{15} * w_{10} * f_{12}, w_1 * f_{16} * w_{11} * f_{11}, w_2 * f_{16} * w_{10} * f_{11}, w_3 * f_{16} * w_{12} * f_{11}, w_4 * f_{16} * w_{11} * f_{11}, w_5 *$

$f_{16}*w_{10}*f_{11}, w_6*f_{16}*w_9*f_{11}, w_7*f_{16}*w_{12}*f_{11}, w_8*f_{16}*w_9*f_{11}, w_9*f_{16}*w_{12}*f_{11}, w_{10}*f_{16}*w_{11}*f_{11}, w_{11}*f_{16}*w_{10}*f_{11}, w_{12}*f_{16}*w_9*f_{11}, w_3*f_{17}*w_{12}*f_{26}, w_6*f_{17}*w_9*f_{26}, w_7*f_{17}*w_{12}*f_{26}, w_8*f_{17}*w_9*f_{26}, w_9*f_{17}*w_{12}*f_{26}, w_{12}*f_{17}*w_9*f_{26}, w_3*f_{18}*w_{12}*f_{25}, w_6*f_{18}*w_9*f_{25}, w_7*f_{18}*w_{12}*f_{25}, w_8*f_{18}*w_9*f_{25}, w_9*f_{18}*w_{12}*f_{25}, w_{12}*f_{18}*w_9*f_{25}, w_3*f_{19}*w_{12}*f_{24}, w_6*f_{19}*w_9*f_{24}, w_7*f_{19}*w_{12}*f_{24}, w_8*f_{19}*w_9*f_{24}, w_9*f_{19}*w_{12}*f_{24}, w_{12}*f_{19}*w_9*f_{24}, w_3*f_{20}*w_{12}*f_{23}, w_6*f_{20}*w_9*f_{23}, w_7*f_{20}*w_{12}*f_{23}, w_8*f_{20}*w_9*f_{23}, w_9*f_{20}*w_{12}*f_{23}, w_{12}*f_{20}*w_9*f_{23}, w_3*f_{23}*w_{12}*f_{20}, w_6*f_{23}*w_9*f_{20}, w_7*f_{23}*w_{12}*f_{20}, w_8*f_{23}*w_9*f_{20}, w_9*f_{23}*w_{12}*f_{20}, w_{12}*f_{23}*w_9*f_{20}, w_3*f_{24}*w_{12}*f_{19}, w_6*f_{24}*w_9*f_{19}, w_7*f_{24}*w_{12}*f_{19}, w_8*f_{24}*w_9*f_{19}, w_9*f_{24}*w_{12}*f_{19}, w_{12}*f_{24}*w_9*f_{19}, w_3*f_{25}*w_{12}*f_{18}, w_6*f_{25}*w_9*f_{18}, w_7*f_{25}*w_{12}*f_{18}, w_8*f_{25}*w_9*f_{18}, w_9*f_{25}*w_{12}*f_{18}, w_{12}*f_{25}*w_9*f_{18}, w_3*f_{26}*w_{12}*f_{17}, w_6*f_{26}*w_9*f_{17}, w_7*f_{26}*w_{12}*f_{17}, w_8*f_{26}*w_9*f_{17}, w_9*f_{26}*w_{12}*f_{17}, w_{12}*f_{26}*w_9*f_{17}, w_1*f_{27}*w_{11}*f_{34}, w_2*f_{27}*w_{10}*f_{34}, w_3*f_{27}*w_{12}*f_{34}, w_4*f_{27}*w_{11}*f_{34}, w_5*f_{27}*w_{10}*f_{34}, w_6*f_{27}*w_9*f_{34}, w_7*f_{27}*w_{12}*f_{34}, w_8*f_{27}*w_9*f_{34}, w_9*f_{27}*w_{12}*f_{34}, w_{10}*f_{27}*w_{11}*f_{34}, w_{11}*f_{27}*w_{10}*f_{34}, w_{12}*f_{27}*w_9*f_{34}, w_1*f_{28}*w_{11}*f_{33}, w_2*f_{28}*w_{10}*f_{33}, w_3*f_{28}*w_{12}*f_{33}, w_4*f_{28}*w_{11}*f_{33}, w_5*f_{28}*w_{10}*f_{33}, w_6*f_{28}*w_9*f_{33}, w_7*f_{28}*w_{12}*f_{33}, w_8*f_{28}*w_9*f_{33}, w_9*f_{28}*w_{12}*f_{33}, w_{10}*f_{28}*w_{11}*f_{33}, w_{11}*f_{28}*w_{10}*f_{33}, w_{12}*f_{28}*w_9*f_{33}, w_1*f_{29}*w_{11}*f_{32}, w_2*f_{29}*w_{10}*f_{32}, w_4*f_{29}*w_{11}*f_{32}, w_5*f_{29}*w_{10}*f_{32}, w_{10}*f_{29}*w_{11}*f_{32}, w_{11}*f_{29}*w_{10}*f_{32}, w_3*f_{30}*w_{12}*f_{31}, w_6*f_{30}*w_9*f_{31}, w_7*f_{30}*w_{12}*f_{31}, w_8*f_{30}*w_9*f_{31}, w_9*f_{30}*w_{12}*f_{31}, w_{12}*f_{30}*w_9*f_{31}, w_3*f_{31}*w_{12}*f_{30}, w_6*f_{31}*w_9*f_{30}, w_7*f_{31}*w_{12}*f_{30}, w_8*f_{31}*w_9*f_{30}, w_9*f_{31}*w_{12}*f_{30}, w_{12}*f_{31}*w_9*f_{30}, w_1*f_{32}*w_{11}*f_{29}, w_2*f_{32}*w_{10}*f_{29}, w_4*f_{32}*w_{11}*f_{29}, w_5*f_{32}*w_{10}*f_{29}, w_{10}*f_{32}*w_{11}*f_{29}, w_{11}*f_{32}*w_{10}*f_{29}, w_1*f_{33}*w_{11}*f_{28}, w_2*f_{33}*w_{10}*f_{28}, w_3*f_{33}*w_{12}*f_{28}, w_4*f_{33}*w_{11}*f_{28}, w_5*f_{33}*w_{10}*f_{28}, w_6*f_{33}*w_9*f_{28}, w_7*f_{33}*w_{12}*f_{28}, w_8*f_{33}*w_9*f_{28}, w_9*f_{33}*w_{12}*f_{28}, w_{10}*f_{33}*w_{11}*f_{28}, w_{11}*f_{33}*w_{10}*f_{28}, w_{12}*f_{33}*w_9*f_{28}, w_1*f_{34}*w_{11}*f_{27}, w_2*f_{34}*w_{10}*f_{27}, w_3*f_{34}*w_{12}*f_{27}, w_4*f_{34}*w_{11}*f_{27}, w_5*f_{34}*w_{10}*f_{27}, w_6*f_{34}*w_9*f_{27}, w_7*f_{34}*w_{12}*f_{27}, w_8*f_{34}*w_9*f_{27}, w_9*f_{34}*w_{12}*f_{27}, w_{10}*f_{34}*w_{11}*f_{27}, w_{11}*f_{34}*w_{10}*f_{27}, w_{12}*f_{34}*w_9*f_{27}\}$

Therefore, the set of the relations is

$$Rels = Rels1 \cup Rels2 \cup Rels3 \cup Rels4.$$

10. From above we have a finite presentation for the subgroup $Conj_V$ of the automorphism groups of the partially commutative group $Aut(G_\Gamma)$ as follows:

$$Conj_V = \langle Gens|Rels\rangle$$

4.4 GAP Presentation for $Conj_V$

This section describes the functions available from the *AutParCommGrp* package which we have written for computing a finite presentation for the subgroup $Conj_V$ of $Aut(G_\Gamma)$ with commuting graph Γ generated by partial conjugations W_V.

To write an algorithm to produce this presentation we first construct W_V the set of generators of the subgroup $Conj_V$ that is defined earlier in Section 4.3, and then find the set $\Re$ of relations that are defined in Theorem 4.3.15. The input of the main function `FinitePresentationOfSubgroupConjv` that provides finite presentation for $Conj_V$ is a simple graph $\Gamma = (V, E)$. A graph with vertex set V of size n always has vertices $\{1, \ldots, n\}$ and E is a list of pairs of elements of V. For example if Γ is a simple graph with vertex set $V = \{x_1, x_2, x_3\}$ and edge set $E = \{[x_1, x_2], [x_1, x_3], [x_2, x_3]\}$ (where $[x, y]$ denotes an edge joining x to y) then Γ will be represented as $([1, 2, 3], [[1, 2], [1, 3], [2, 3]])$. The output of `FinitePresentationOfSubgroupConjv` consists of two sets *gens* and *rels*, where *gens* is the list of the generators of the automorphism $\alpha_{C,x}$ defined above and *rels* is the list of the relators.

In addition, to the functions `IsSimpleGraph`, `StarLinkOfVertex`, `DeleteverticesFromGraph` and `ConnectedComponentsOfGraph` which we have described in Sections 2.7.1, 3.3.1, 2.7.3 and 2.7.4 respectively the function `FinitePresentationOfSubgroupConjv` runs the following functions:

4.4.1 `EquivalenceClassOfVertex` Function

The input of the function `EquivalenceClassOfVertex`(St) is the list of stars St that is defined in Section 3.3.1. It computes the equivalence classes for each vertex v. The algorithm carries out the following instructions:

EQUIVALENCECLASSOFVERTEX(St)

$sV \leftarrow$ SIZE(St)
for i **in** $\{1, \ldots, sV\}$
    do for j **in** $\{1, \ldots, sV\}$

do $diff1 \leftarrow$ DIFFERENCE$(St[i], [i, j])$
$diff2 \leftarrow$ DIFFERENCE$(St[j], [i, j])$
if $diff1 = diff2$
then ADD j to new list $EqCl1$
ADD $EqCl1$ to new list $EqCl$
return $EqCl$

4.4.2 `ClassPreservingConnectedComponents` Function

The input of the function `ClassPreservingConnectedComponents`$(EqCl, Comps)$ is $EqCl$ the list of equivalence classes of vertices of Γ and the list of connected components $Comps$ of a subgraph B of Γ (usually $B = \Gamma \backslash St(x)$, for some vertex x). It constructs a new list of connected components $Comps$ from the connected components of the subgraph B by finding the connected components which satisfy the conditions of partial conjugation for W_V. The algorithm carries out the following instructions:

CLASSPRESERVINGCONNECTEDCOMPONENTS$(EqCl, Comps)$
$sizeEqCl \leftarrow$ SIZE$(EqCl)$
for i **in** $\{1, \ldots, sizeEqCl\}$
do $sizeComps \leftarrow$ SIZE$(Comps)$
$sizeEqClcurrent \leftarrow$ SIZE$(EqCl[i])$
$cdash \leftarrow$ EMPTYLIST
$remainingcdash \leftarrow$ EMPTYLIST
for j **in** $\{1, \ldots, sizeEqClcurrent\}$
do for k **in** $\{1, \ldots, sizeComps\}$
if $EqCl[i][j] \in Comps[k]$
then $cdash \leftarrow$ UNION$(cdash, Comps[k])$
for k **in** $\{1, \ldots, sizeComps\}$
do if $Comps[k] \not\subset cdash$
then ADD $Comps[k]$ to the list $remainingcdash$
ADD $cdash$ to the list $remainingcdash$
$Comps = remainingcdash$
return $Comps$

4.4.3 `GeneratorsOfSubgroupConjv` Function

The input of the function `GeneratorsOfSubgroupConjv`(NE, NV, St, V) is the list NE of all lists of edges of $\Gamma\backslash St(v)$, the list NV of all lists of vertices of $\Gamma\backslash St(v)$, the list of stars St that is defined in Section 3.3.1 and the list of vertices V. It computes the list $gens1$ which form the type (1) generators of $Conj_V$. The algorithm carries out the following instructions:

```
GeneratorsOfSubgroupConjv(NE, NV, St, V)
sNE ← Size(NE)
invV ← ComputeTheInveres(V)
L ← Concatenation(V, invV)
EqCl ← EquivalenceClassOfVertex(St)
for h in {1, . . . , sNE}                ▷ h ∈ V
      do G2 ← NE(h)
         G1 ← NV(h)
         R3 ← ConnectedComponentsOfGraph(G1, G2)
         Comps ← R3(1)              ▷ Comps is the list of all components
         sComps ← R3(2)
         P ← ClassPreservingConnectedComponents(EqCl, Comps)
         Add the non-empty element of P to new list Y4
         sY4 ← Size(Y4)
         for i in {1, . . . , sY4}
               do diff2 ← Difference(L, Y4[i])
                  Add diff2 to new list xs1
         for i in {1, . . . , sY4}
               do sz ← Size(xs1[i])
                  for j in {1, . . . , sz}
                        do KK ← Concatenation(Y4[i], [xs1[i][j]])
                           HH ← [KK, xs1[i][j]]
                           Add HH to new list Y5
         sY5 ← Size(Y5)
         Add Y5 to new list Y6
         Add xs1 to new list xs2
         Add Bs to new list Y3
         sY6 ← Size(Y6)
```

if $sY6 \neq 0$
then $Y7 \leftarrow$ CONCATENATION$(Y6)$
$sY7 \leftarrow$ SIZE$(Y7)$
$xs3 \leftarrow$ CONCATENATION$(xs2)$
$sxs3 \leftarrow$ SIZE$(xs3)$
for i **in** $\{1, \ldots, sxs3\}$
do ADD the non-empty element of $xs3$ to new list xs
$sxs \leftarrow$ SIZE(xs)
$Uxs \leftarrow$ UNION(xs)
$Uxs \leftarrow$ SIZE(Uxs)
for i **in** $\{1, \ldots, sY7\}$
do ADD the non-empty element of $Y7$ to new list $CxY1$
$sCxY1 \leftarrow$ SIZE$(CxY1)$
for j **in** $\{1, \ldots, sCxY1\}$
do COMPUTE CxY a list of the definitions of the partial conjugations W_V of $Conj_V$
$sCxY \leftarrow$ SIZE(CxY)
$Y8 \leftarrow$ CONCATENATION(Bs)
for i **in** $\{1, \ldots, sY8\}$
do ADD the non-empty element of $Y8$ to new list Y
$sY \leftarrow$ SIZE(Y)
for k **in** $\{1, \ldots, sCxY\}$
do CONSTRUCT a list f such that $f(n) = CxY(n)$, $n \in N$
$sf \leftarrow$ SIZE(f)
for j **in** $\{1, \ldots, sf\}$
do ADD f_i the name of the i^{th} element of f to new list $gens1$
$sgens1 \leftarrow$ SIZE$(gens1)$
return either $[CxY, sCxY, Y, sY, f, sf, gens1, sgens1]$ or
an empty list if there is no component C satisfies the Definition 4.3.1

Remark 4.4.1. Note that,

(1) We use the functions `APCGRelationRConj1`, `APCGRelationRConj2`, `APCGRelationRConj3` and `APCGRelationRConj4` which are described in Sections 3.3.4, 3.3.5, 3.3.6 and 3.3.7 respectively to find the set $\Re$ of relations that are defined in Theorem 4.3.15, by using the output of `GeneratorsOfSubgroupConjv` above

rather than the output of `GeneratorsOfSubgroupConj` which is described in Section 3.3.3.

(2) We use the function `APCGConjLastReturn`$(gens4, R2a, sR2a)$ which is described in Section 3.3.8 to return the final return $[gens, rels, GGG]$ of the functions `FinitePresentationOfSubgroupConjv` below.

4.4.4 `FinitePresentationOfSubgroupConjv` Function

The function `FinitePresentationOfSubgroupConjv`(V, E) provides finite presentation for the subgroup $Conj_V$. The input of this function is a simple graph $\Gamma = (V, E)$. It returns $[gens, rels, GGG]$, where,

(i) $gens$ is a list of free generators of the subgroup $Conj_V$ of the automorphism group $Aut(G_\Gamma)$ of G_Γ .

(ii) $rels$ is a list of relations in the generators of the free group F. Note that relations are entered as relators, i.e., as words in the generators of the free group.

(iii) $GGG := F/rels$ is a finitely presented of the subgroup $Conj_V$ of the automorphism group $Aut(G_\Gamma)$ of G_Γ.

The algorithm carries out the following instructions:

FinitePresentationOfSubgroupConjv(V, E)

if Γ is simple graph
  then Call The Function StarLinkOfVertex
    Call The Function DeleteVerticesFromGraph
    Call The Function GeneratorsOfSubgroupConjv
    Call The Function APCGRelationRConj1
    Call The Function APCGRelationRConj2
    Call The Function APCGRelationRConj3
    Call The Function APCGRelationRConj4
    Call The Function APCGConjLastReturn
  else **return** “The graph must be a simple graph”
return $[gens, rels, GGG]$

For example:

```
gap> C:=FinitePresentationOfSubgroupConjv([1,2,3],[[1,2],[2,3]]);
[ [ f1, f2, f3, f4, f5, f6, f7, f8 ], [f1*f4, f2*f3, f3*f2, f4*f1,
f5*f8, f6*f7, f7*f6, f8*f5, f1*f2*f4*f3, f1*f3*f4*f2, f2*f4*f3*f1,
f3*f4*f2*f1, f5*f6*f8*f7, f5*f7*f8*f6, f6*f8*f7*f5, f7*f8*f6*f5,
f2*f1*f3*f4, f3*f1*f2*f4, f2*f4*f3*f1, f3*f4*f2*f1, f5*f6*f8*f7,
f8*f6*f5*f7,f5*f7*f8*f6, f8*f7*f5*f6], <fp group on the generators
[ f1, f2, f3, f4, f5, f6, f7, f8 ]> ]
```

Remark 4.4.2. We use the function `TietzeTransformations`(G) which is described in Section 2.7.19 to simplify the presentation of $Conj_V$. For example, using the output of `FinitePresentationOfSubgroupConjv` above:

```
gap> G:=C[3];
<fp group on the generators [ f1, f2, f3, f4, f5, f6, f7, f8 ]>
gap> TietzeTransformations(G);
[ <fp group of size infinity on the generators [ f1, f2, f5, f6 ]>,
[ f1*f2*f1^-1*f2^-1, f5*f6*f5^-1*f6^-1 ] ]
```

Part II

Differential Graded Algebraic structures

Chapter 5

Introduction and Preliminaries for DG Algebraic structures

5.1 Introduction

Let G be a group with identity e and R be a ring with unit 1 different from 0. Then R is said to be G-graded ring if there exists an additive subgroup R_g of R such that $R = \oplus \sum\limits_{g \in G} R_g$ and $R_g R_h \subseteq R_{gh}$ for all $g, h \in G$. Let K be a field of characteristic two, $R = K[x_1, x_2, \cdots, x_n]$ be a graded ring, graded in the negative way, and let M be differential graded R-module, where the degree of the differential is P.

Our aim is to study the case that ($P \leq -2$, $n > 1$), and we give classification for the types where M is a solvable module and the cases where M is not solvable, using the dimension of the module and the degree of the differential on the module. Also we will give an algorithm for these cases, implement in GAP.

5.2 Preliminaries

In this section, we give a brief overview of some definitions and results of exact homology sequences from [5], [42], [50] and [54]. For background on rings and modules we use [21], [33] and [42].

5.2.1 Exact Homology Sequences

Definition 5.2.1. Consider a sequence (finite or infinite) of abelian group and homomorphisms

$$\cdots A_1 \xrightarrow{\phi_1} A_2 \xrightarrow{\phi_2} A_3 \longrightarrow \cdots$$

This sequence is said to be **exact** at A_2 if and only if $Im(\phi_1) = Ker(\phi_2)$. If it is every where exact, it is said to be an **exact sequence**.

Theorem 5.2.2. *(1) $A_1 \xrightarrow{\phi_1} A_2 \xrightarrow{\phi_2} 0$, is exact sequence if and only if ϕ_1 is epimorphism.*

(2) $0 \xrightarrow{\phi_1} A_1 \xrightarrow{\phi_2} A_2$ is exact sequence if and only if ϕ_2 is monomorphism.

Proof. 1) $A_1 \xrightarrow{\phi_1} A_2 \xrightarrow{\phi_2} 0$ is exact sequence at A_2 if and only if $Im(\phi_1) = Ker(\phi_2) = A_2$ iff ϕ_1 is epimorphism.

2) $0 \xrightarrow{\phi_1} A_1 \xrightarrow{\phi_2} A_2$ is exact sequence at A_1 if and only if $Ker(\phi_2) = Im(\phi_1)$ iff ϕ_1 if and only if ϕ_2 is monomorphism.

□

Definition 5.2.3. An exact sequence of the form

$$0 \longrightarrow A_1 \xrightarrow{\phi_1} A_2 \xrightarrow{\phi_2} A_3 \xrightarrow{\phi_3} 0$$

is called **a short exact sequence**. A diagram of modules

$$\begin{tikzcd} A_1 \arrow[r, "\phi_1"] \arrow[d, "\psi_1"'] & A_2 \arrow[d, "\psi_2"] \\ A_3 \arrow[r, "\phi_2"'] & A_4 \end{tikzcd}$$

and homomorphisms is said be **commutative** iff $\psi_2\phi_1 = \phi_2\psi_1$.

Theorem 5.2.4. *Consider the following commutative diagram*

$$\begin{tikzcd} 0 \arrow[r] & A_2 \arrow[r, "\varphi"] \arrow[d, "\eta_2"] & A_1 \arrow[r, "\psi"] \arrow[d, "\eta_1"] & A_0 \arrow[r] \arrow[d, "\eta_0"] & 0 \\ 0 \arrow[r] & B_2 \arrow[r, "\varphi'"] & B_1 \arrow[r, "\psi'"] & B_0 \arrow[r] & 0 \end{tikzcd}$$

with exact rows. If any two of the three homomorphisms η_0, η_1 and η_2 are isomorphism, then the third is an isomorphism too.

Lemma 5.2.5. *Suppose $\phi: A \longrightarrow B$ is epimorphism with kernel K, then the sequence $0 \longrightarrow K \xrightarrow{i} A \xrightarrow{\phi} B \xrightarrow{\psi} 0$ is exact where i is the inclusion map.*

Proof. Since ϕ is onto, then $Im(\phi) = B = Ker(\psi)$. Hence the sequence is exact at B

Also, $Im(i) = A = Ker(\phi)$, and hence the sequence is exact at A. Therefore, $0 \longrightarrow K \xrightarrow{i} A \xrightarrow{\phi} B \xrightarrow{\psi} 0$ is exact. □

Theorem 5.2.6. *Suppose that the sequence $A_1 \xrightarrow{\phi_1} A_2 \xrightarrow{\phi_2} A_3 \xrightarrow{\phi_3} A_4$ is exact, then the following are equivalent:*

1) ϕ_1 is epimorphism.

2) ϕ_2 is the zero homomorphism.

2) ϕ_3 is monomorphism.

Proof. **(1) gives (2):** Suppose ϕ_1 is epimorphism, so $\mathrm{Im}(\phi_1) = A_2$. Since the sequence is exact we have $Im(\phi_1) = Ker(\phi_2)$, and so $Ker(\phi_2) = A_2$, which gives that $\phi_2 = 0$.

(2) gives (3): Suppose ϕ_2 is the zero map. Then $\mathrm{Im}(\phi_2) = 0$, using that the sequence is exact we have $Im(\phi_2) = Ker(\phi_3) = 0$. Therefore, ϕ_3 is monomorphism.

(3) gives (1): Suppose that ϕ_3 is monomorphism. Then the sequence is exact at A_3, so $Ker(\phi_3) = Im(\phi_2)$. But ϕ_3 is 1-1, we have $Im(\phi_2) = 0$ and so ϕ_2 is a zero map. Since the sequence is exact at A_2 we have $Ker(\phi_2) = Im(\phi_1) = A_1$. Hence ϕ_1 is epimorphism.

□

Definition 5.2.7. Consider the sequences

$$\cdots \longrightarrow A_1 \xrightarrow{\phi_1} A_2 \xrightarrow{\phi_2} \cdots$$

$$\cdots \longrightarrow B_1 \xrightarrow{\psi_1} B_2 \xrightarrow{\psi_2} \cdots$$

A homomorphism from the first sequence into the second sequence is a family of homomorphisms $\alpha_i: A_i \longrightarrow B_i$ such that the following diagram commutes.

$$\begin{tikzcd}
\cdots \arrow[r] & A_{-1} \arrow[r, "\varphi_{-1}"] \arrow[d, "\alpha_{-1}"] & A_0 \arrow[r, "\varphi_0"] \arrow[d, "\alpha_0"] & A_1 \arrow[r] \arrow[d, "\alpha_1"] & \cdots \arrow[r] \arrow[d] & A_i \arrow[r, "\varphi_1"] \arrow[d, "\alpha_i"] & A_{i+1} \arrow[r] \arrow[d, "\alpha_i+1"] & \cdots \\
\cdots \arrow[r] & B_{-1} \arrow[r, "\psi_{-1}"] & B_0 \arrow[r, "\psi_0"] & B_1 \arrow[r] & \cdots \arrow[r] & B_i \arrow[r, "\psi_1"] & B_{i+1} \arrow[r] & \cdots
\end{tikzcd}$$

(i.e. $\alpha_{i+1} \circ \varphi_i = \psi_i \circ \alpha_i$ for all i). It is an isomorphism of sequences if each α_i is an isomorphism.

Definition 5.2.8. Let $C = \{C_p, \partial_p\}$ and $C' = \{C'_p, \partial'_p\}$ be a chain complexes. A chain map $\phi : C \to C'$ is a collection of homomorphisms $\phi_p : C_p \to \acute{C}_p$ such that $\acute{\partial}_p \circ \phi_p = \phi_{p-1} \circ \partial_p$, for all p (i.e., the following diagrams commutes)

$$\begin{tikzcd}
\cdots \arrow[r] & C_{p+1} \arrow[r, "\partial_{p+1}"] \arrow[d, "\phi_{p+1}"] & C_p \arrow[r, "\partial_p"] \arrow[d, "\phi_p"] & C_{p-1} \arrow[r] \arrow[d, "\phi_{p-1}"] & \cdots \\
\cdots \arrow[r] & \acute{C}_{p+1} \arrow[r, "\partial'_{p+1}"] & \acute{C}_p \arrow[r, "\partial'_p"] & \acute{C}_{p-1} \arrow[r] & \cdots
\end{tikzcd}$$

Lemma 5.2.9. *A chain map $\phi : C \to C'$ induces a homomorphism*
$(\phi_*)_p : H_p(C) \to H_P(C')$, for all p given by:
$(\phi_*)_p(x + im(\partial_{p+1})) = \phi_p(x) + Im(\partial'_{p+1})$

Proof. Suppose $\phi : C \to C'$ is a chain map. To show that $(\phi_*)_p$ is well-defined. Let $x + Im(\partial_{p+1}) = y + Im(\partial_{p+1})$. Then $x - y \in Im(\partial_{p+1})$. Since ∂_{p+1} is onto, there is $z \in C_{p+1}$ such that $\partial_{p+1}(z) = x - y$.

But $\phi_p \circ \partial_{p+1} = \partial'_{p+1} \circ \phi_{p+1}$, implies to $\partial'_{p+1}(\phi_{p+1}(z)) = \phi_p(\partial_{p+1}(z)) = \phi_p(x-y) = \phi_p(x) - \phi_p(y)$.

Therefore, $\phi_p(x) + im\partial'_{p+1} = \phi_p(y) + im(\partial'_{p+1})$. Also,

$$\begin{aligned}
(\phi_*)_p(x + Im(\partial_{p+1}) + y + im(\partial_{p+1})) &= (\phi_*)_p(x + y) + im(\partial_{p+1})) \\
&= \phi_p(x + y) + im(\partial'_{p+1}) \\
&= \phi_p(x) + \phi_p(y) im(\partial'_{p+1}) \\
&= \phi_p(x) + im(\partial'_{p+1}) + \phi_p(y) + im(\partial'_{p+1}). \\
&= (\phi_*)_p(x + Im(\partial_{p+1})) + (\phi_*)_p(y + im(\partial_{p+1})).
\end{aligned}$$

And
$$\begin{aligned}
(\phi_*)_p(r \cdot (x + im(\partial_{p+1})) &= (\phi_*)_p(rx + im(\partial_{p+1}) \\
&= \phi_p(rx) + im(\partial'_{p+1}) \\
&= r \cdot \phi_p(x) + im(\partial'_{p+1}) \\
&= r(\phi_*)_p(x + im\partial_{p+1}).
\end{aligned}$$

Hence $(\phi_*)_p$ is a homomorphism. $\square$

Lemma 5.2.10. *a) The identity map $i : C \to C$ is a chain map and $(i_*)_p : H_p(C) \to H_p(C)$ is the identity homomorphism.*

b) If $\phi : C \to C'$ and $\psi : C' \to \acute{C}'$ are chain maps, then $\psi \circ \phi : C \to \acute{C}'$ is a chain map and $(\psi \circ \phi)_* = \psi_* \circ \phi_*$.

Proof. a) Clear by Lemma.

b) Consider the following digram

$$\begin{tikzcd}
\cdots \arrow[r] & C_{p+1} \arrow[r, "\partial_{p+1}"] \arrow[d, "\phi_{p+1}"'] & C_p \arrow[r, "\partial_p"] \arrow[d, "\phi_p"] & C_{p-1} \arrow[r, "\partial_{p-1}"] \arrow[d, "\phi_{p-1}"'] & \cdots \\
\cdots \arrow[r] & C'_{p+1} \arrow[r, "\partial'_{p+1}"] \arrow[d, "\psi_{p+1}"'] & C'_p \arrow[r, "\partial'_p"] \arrow[d, "\psi_p"] & C'_{p-1} \arrow[r, "\partial'_{p-1}"] \arrow[d, "\psi_{p-1}"'] & \cdots \\
\cdots \arrow[r] & C''_{p+1} \arrow[r, "\partial''_{p+1}"] & C''_p \arrow[r, "\partial''_p"] & C''_{p-1} \arrow[r, "\partial''_{p-1}"] & \cdots
\end{tikzcd}$$

Since the diagram commutes we have $\phi_{p-1} \circ \partial_p = \partial'_p \circ \phi_p$, and so $\psi_{p-1}(\phi_{p-1} \circ \partial_p = \psi_{p-1}(\partial'_p \circ \phi_p)$. Similarly, we have $\psi_{p-1} \circ \partial'_p = \partial''_p \circ \psi_p$, and so $\psi_{p-1} \circ \partial'_p \circ \phi_p = \partial''_p \circ \psi_p \circ \phi_p$. Therefore, $\psi_{p-1} \circ \phi_{p-1} \partial_p = \partial''_p \circ \psi_p \circ \phi_p$.

By definition $(\phi_*)_p : H_p(C) \to H_p(C')$ is given by

$(\phi_*)_p(x + im\partial_{p+1}) = \phi(x) + im\partial'_{p+1}$ and

$(\psi_*)_p : H_p(C') \to H_p(C'')$ is given by

$(\psi_*)_p(\phi(x) + im\partial'_{p+1}) = \psi(\phi(x)) + im\partial''_{p+1}$. Now,

$((\psi \circ \phi)*)_p : H_p(C) \to H_p(C'')$ is given by:

$((\psi \circ \phi)*)_p(x + im\partial_{p+1}) = (\psi \circ \phi)(x) + im\partial'_{p+1}$.

So, $((\psi \circ \phi)*)_p(x + im\partial_{p+1}) = (\psi \circ \phi)(x) + im\partial''_{p+1}$.

$$= \psi(\phi(x)) + im\partial''_{p+1}.$$

$$= (\psi_*)_p(\phi(x)) + im\partial''_{p+1}.$$

Hence $((\psi \circ \phi)*)_p = (\psi_*)_p \circ (\phi_*)_p$.

$\square$

Chapter 6

Graded Rings and Graded Modules

In this chapter the concept of graded rings and some of its properties are presented. We also, give the definitions of graded algebras, and differential graded modules over the graded polynomial ring $R = K[x_1, x_2, \ldots, x_n]$.

6.1 Graded Rings

Definition 6.1.1. [59] Let G be a group with identity e. Then a ring R is said to be G-**graded ring** if there exist an additive subgroups R_g of R such that $R = \oplus \sum\limits_{g \in G} R_g$ and $R_g R_h \subseteq R_{gh}$ $for\ all\ g, h \in G$ (some references use $R_g R_h \subseteq R_{g+h}$ rather than $R_g R_h \subseteq R_{gh}$, for example see [33]).

We denote the G-graded ring R by (R, G), and we denote the **support** of the graded ring (R, G) by

$$supp(R, G) = \{g \in G : R_g \neq 0\}.$$

The elements of R_g are called homogeneous of degree g. If $x \in R$, then x can be written uniquely as $\sum\limits_{g \in G} x_g$ where x_g is the component of x in R_g. Also we write, $h(R) = \bigcup\limits_{g \in G} R_g$.

Definition 6.1.2. [21] Let A be a subset of R, for $\lambda \in G$ we write A_λ for $A \cap R_\lambda$. A subset A is called **graded subset** of R if $A = \sum\limits_{\lambda \in G} A_\lambda$.

Let I be an ideal of R, we say I is a **graded ideal** of (R,G) if $I = \oplus \sum\limits_{g\in G}(R_g \cap I)$.

Remark 6.1.3. Clearly, $\oplus \sum\limits_{g\in G}(R_g \cap I) \subseteq I$ and hence I is a graded of (R,G) if $I \subseteq \sum\limits_{g\in G}(R_g \cap I)$. Also, $J = \sum\limits_{g\in G}(R_g \cap I)$ is the largest graded ideal of R which is contained in I.

Now, we give some examples of G-graded ring.

Example 6.1.0.1
Let G be any group, then R is a G-graded ring with: $R_e = R$ and $R_g = 0$ for all $g \in G - \{e\}$. This grading is called the **trivial grading** of R by G.

Example 6.1.0.2
The polynomial ring $S = R[x_1, x_2, \ldots, x_n]$ in n variables over the commutative ring R is an example of a graded ring. Here $S_0 = R$ and the homogeneous component of degree k is the subgroup of all R-linear combinations of monomials of degree k i.e., $S_d = \{\sum_{m\in N} r_m X^m \mid r_m \in R \text{ and } m_1 + \ldots + m_n = d\}$. This is called the standard grading on the polynomial ring $R[x_1, \ldots, x_n]$. The ideal I generated by $x_1, \ldots, x_n$ is a graded ideal: every polynomial with zero constant term may be written uniquely as a sum of homogeneous polynomials of degree $k > 1$, and each of these has zero constant term hence lies in I. More generally, an ideal is a graded ideal if and only if it can be generated by homogeneous polynomials (see Lemma 6.1.4 for the proof).

Example 6.1.0.3
[64] Let K be a field, and $R = K[x]$ be the polynomial ring over K in one variable x. Let $G = \mathbb{Z}_3$, then R is a G-graded ring with:

$$R_j = (kx^{3r+j} : k \in K,\ r = 0, 1, 2, ...),\ for\ j \in \mathbb{Z}_3.$$

Example 6.1.0.4
Let $R = \mathbb{Z}[i] = \{a + ib : a, b \in \mathbb{Z}\}$ (the Gaussian integers), and $G = \mathbb{Z}_2$, then R is a G-graded ring with: $R_0 = \mathbb{Z}$, and $R_1 = i\mathbb{Z}$.

The following example shows that an ideal of a G-graded ring need not be a graded ideal in general:

Example 6.1.0.5
Let $R = \mathbb{Z}[i]$, and Let $G = \mathbb{Z}_2$. Then R is a G-graded ring with: $R_0 = \mathbb{Z}$, and

$R_1 = i\,\mathbb{Z}$. Let $I =< 1 + i >$, where $x = (1 + i), x_0 = 1$ and $x_1 = i$. Clearly $x_0 \notin I$ because if $x_0 \in I$ then there is $a + ib \in \mathbb{Z}[i]$ such that $1 = (a + ib)(1 + i)$ which implies $a - b = 1$ and $a + b = 0$. Hence $2a = 1$, contradiction. Thus I is not a graded ideal of $(R\ , G)$.

Lemma 6.1.4. *An ideal is a graded (homogeneous) ideal if and only if it can be generated by homogeneous polynomials.*

Proof. Let R be a graded ring such that $R = \oplus \sum_{g \in G} R_g$, where the R_g are additive abelian groups such that $R_g R_h \subseteq R_{g+h}$ for $g, h \geq 1$. If $I \subset K[x]$ is graded (homogeneous), the homogeneous parts of the generators of I obviously generate I. Conversely, let I be an ideal generated by homogeneous polynomials f_i, $i = 1, \ldots, r$. Suppose that $w \in I$ i.e., $w = \sum_{i=1}^{r} a_i f_i$, $a_i \in K[x]$. Note that each homogeneous part $(a_i)_{[j]} f_i$ of a_i is in I, because I is an ideal. Since this holds for any $g \in I$, we have that

$$\oplus_{i \geq 1}(I \cap R_g) \subseteq I \subseteq \oplus_{i \geq 1}(I \cap R_g)$$

This means both are equal and I is graded ideal. □

Proposition 6.1.5. *[33] Stated that: Let R be a graded ring, let I be a graded ideal in R and let $I_k = I \cap R_k$ for all $k \geq 0$. Then R/I is naturally a graded ring whose homogeneous component of degree k is isomorphic to R_k/I_k.*

There would be necessary to prove the proposition above.

Proof.

1. We show that $R_i I_j \subseteq I_{i+j}$. Let $x \in R_i I_j$ then $x = r_i a_j$ where $r_i \in R_i$ and $a_j \in I_j$. So $x \in R_i I_j$ implies that $r_i a_j \in R_i I_j$ implies that $r_i a_j \in R_i R_j \cap I$ (since $R_j \cap I = I_j$) implies that $r_i a_j \in R_{i+j} \cap I$ (since $R_i R_j \subseteq R_{i+j}$) which implies that $r_i a_j \in I_{i+j}$ (since $R_{i+j} \cap I = I_{i+j}$). Thus $R_i I_j \subseteq I_{i+j}$.

2. We show that the multiplication $(R_i/I_i)(R_j/I_j) \subseteq R_{i+j}/I_{i+j}$. is well defined. We need to show that:

$$(r_i + I_j)(r_j + I_j) = r_i r_j + I_{i+j}$$

where $r_i + I_i \in R_i/I_i$ and $r_j + I_j \in R_j/I_j$.

Let $r_i + I_i = r_i' + I_i$ and $r_j + I_j = r_j' + I_j$. We need to show that:

$$(r_i + I_i)(r_j + I_j) = (r_i' + I_i)(r_j' + I_j)$$

i.e., we need to show $r_ir_j + I_{i+j} = r'_ir'_j + I_{i+j}$. So if we show that $(r_ir_j - r'_ir'_j) \in I_{i+j}$ we are done. Note that $r_i + I_i = r'_i + I_i$ implies that $r_i - r'_i \in I_i$ implies that $(r_i - r'_i)r_j \in I_i$ (by multiply both sides by r_j). So $r_ir_j - r'_ir_j \in I_j$ (because I_i is an ideal). Similarly, $r_j + I_j = r'_i + I_j$ implies that $r_j - r'_i \in I_j$ implies that $r'_i(r_i - r'_i) \in I_j$ (by multiply both sides by r'_i). Hence $r'_ir_j - r'_ir'_j \in I_j$. Therefore, $(r_ir_j - r'_ir_j) + (r'_ir_j - r'_ir'_j) \in I_i + I_j \subset I$, which implies that $(r_ir_j - r'_ir'_j) \in I_i + I_j \subset I$. But, $r_ir_j \in R_iR_j \subset R_{i+j}$. So $r_ir_j \in R_{i+j}$ and $r'_ir'_j \in R_{i+j}$. Hence $r_ir_j - r'_ir'_j \in I \cap R_{i+j} = I_{i+j}$.

3. Now we prove that $R/I \cong \oplus_{k=0}^{\infty} R_k/I_k$ where $I_k = R_k \cap I$.

   For each $r \in R$, $r = \sum_{i=0}^{n} r_i$ such that $r_i \in R_i$, we define $\varphi : R \longrightarrow \oplus_{k=0}^{\infty} R_k/I_k$ by :

$$\varphi(r) = \sum r_i + I_i$$

   (a) φ is ring homomorphism for:
   - If $r = \sum r_i$ and $t = \sum t_i \in R$ then,
     $\varphi(r + t) = \varphi(\sum r_i + \sum t_i) = \varphi(\sum r_i + t_i) = \sum(r_i + t_i) + I_i$
     $= (\sum r_i + I_i) + (\sum t_i + I_i) = \varphi(r) + \varphi(t)$.
   - If $r = \sum r_i$ and $t = \sum t_i \in R$ then,
     $\varphi(r \cdot t) = \varphi((\sum r_i) \cdot (\sum t_i)) = \varphi(\sum \sum r_it_i) = \sum \sum r_it_i + I_i$
     $= (\sum r_i + I_i)(\sum t_i + I_i) = \varphi(r) \cdot \varphi(t)$.
     So φ is ring homomorphism.

   (b) φ is onto for:
   Let $y \in \oplus_{k=0}^{\infty} R_k/I_k$ implies that $y = \sum_{i=0}^{n} r_i + I_i$ implies that there exists $x \in R$; $x = \sum r_i$ such that $\varphi(x) = \varphi(\sum r_i) = \sum r_i + I_i$. Thus φ is onto.

   (c) $ker(\varphi) = I$ for :
   $x \in ker(\varphi)$ if and only if $\varphi(\sum_{i=0}^{n} x_i) = 0$ if and only if $\varphi(\sum_{i=0}^{n} x_i) = \sum x_i + I_i = \sum_{i=0}^{n} I_i$ if and only if $\sum x_i \in \sum I_i \cong \oplus_{k=0}^{\infty} I_i = I$. Hence $R/I \cong \oplus_{k=0}^{\infty} s_k/I_k$ (by the first isomorphism theorem).

4. Now we check the ring axioms:

   (a) $R/I = \oplus_{k=0}^{\infty} R_k/I_k$ is abelian group.

   (b) If $r_i + I_i, r_j + I_j$ and $r_n + I_n \in R/I$ then,
   $[(r_i + I_i) \cdot (r_j + I_j)] \cdot (r_n + I_n) = (r_ir_j + I_{i+j})(r_n + I_n) = r_ir_jr_n + I_{i+j+n} = (r_i + I_i) \cdot (r_jr_n + I_{j+n}) = (r_i + I_i) \cdot [(r_j + I_j) \cdot (r_n + I_n)]$.

Also, $(r_i+I_i)\cdot[(r_j+I_j)+(r_n+I_n)] = [(r_i+I_i)\cdot(r_j+I_j)]+[(r_i+I_i)\cdot(r_n+I_n)]$. Hence associative holds.

$\square$

Proposition 6.1.6. *Let R be a G-graded ring and $x, y \in R, g \in G$. Then*

(1) $(x + y)_g = x_g + y_g$.

(2) $(xy)_g = \sum\limits_{\lambda\in G} x_\lambda y_{\lambda^{-1}g}$.

Proof. Let $x, y \in R$, then $x = \sum\limits_{h\in G} x_h$, $y = \sum\limits_{s\in G} y_s$.

(1) If $x_h + y_s \in R_g - \{0\}$, then $x_h + y_s \in R_g \cap (R_h + R_s) \neq 0$. Thus $g = h = s$ and hence $(x + y)_g = x_g + y_g$.

(2) Assume $xy = \sum\limits_{h,s\in G} x_h y_s$. If $x_h y_s \in R_g$ then $x_h y_s = 0$ or $hs = g$. Thus $s = h^{-1}g$ and hence, $(xy)_g = \sum\limits_{h\in G} x_h y_{h^{-1}g}$.

$\square$

Proposition 6.1.7. *Let R be a G-graded ring. Then*

(1) *R_e is a subring of R and $1 \in R_e$.*

(2) *R_g and R are R_e-modules.*

Proof. (1) R_e is closed under multiplication, because $R_eR_e \subseteq R_e$ so R_e is a subring of R. Let $1 = \sum\limits_{s\in G} r_s$ be the homogeneous decomposition of $1 \in R$. pick $\iota \in G$, and $\lambda_\iota \in R_\iota$, then $\lambda_\iota = 1.\lambda_\iota = \sum\limits_{s\in G} r_s\lambda_\iota$ with $r_s\lambda_\iota \in R_{s\,\iota}$. Consequently $r_s\lambda = 0$ for all $s \neq e$ in G. It follows that $r_s\lambda = 0$ for all $s \neq e$ in G and for all $\lambda \in R$. Therefore, $1 = r_e \in R_e$.

(2) Since $R_eR_g \subseteq R_g$, and $R_gR_e \subseteq R_e$, we have R and R_g are left R_e-modules.

$\square$

6.2 Graded Modules

In this section, we will give a brief overview of some definitions and results of graded algebras, and differential graded modules over the graded polynomial ring $R = K[x_1, x_2, \ldots, x_n]$ following [54], [64], [3], [23] and [66].

Definition 6.2.1. A graded K-algebra A is a sequence of K-vector spaces $\{A_j\}_{j\in\mathbb{Z}}$, together with vector space homomorphisms:

$$\pi : A_i \otimes_K A_j \longrightarrow A_{i+j}\ for\ i, j \in \mathbb{Z}\ and$$

$$\mu : K \longrightarrow A_0, such\ that\ the\ following\ diagrams$$

$$\begin{tikzcd} A_i \otimes A_j \otimes A_m \arrow[r, "\pi\otimes 1"] \arrow[d, "1\otimes\pi"'] & A_{i+j} \otimes A_m \arrow[d, "\pi"] \\ A_i \otimes A_{j+m} \arrow[r, "\pi"] & A_{i+j+m} \end{tikzcd}$$

$$\begin{tikzcd} K \otimes A_j = A_j \otimes K \arrow[r, "1\otimes\mu"] \arrow[d, "\mu\otimes 1"'] \arrow[dr, equal] & A_j \otimes A_0 \arrow[d, "\pi"] \\ A_0 \otimes A_j \arrow[r, "\pi"'] & A_j \end{tikzcd}$$

commute for all $i, j, m \in \mathbb{Z}$

Definition 6.2.2. Let A be a graded K-algebra and $\psi : A_j \otimes_K A_j \to A_j \otimes_K A_i$ be the K-vector space isomorphism which takes $a \otimes b$ into $b \otimes a$. Then A is **commutative** iff the following diagram:

$$\begin{tikzcd} A_i \otimes_K A_j \arrow[rr, "\psi"] \arrow[dr, "\pi"'] & & Aj \otimes_K A_i \arrow[dl, "\pi"] \\ & A_{i+j} & \end{tikzcd}$$

commutes for all $i, j, \in \mathbb{Z}$.

A graded K-algebra A is called **graded integral domain** iff whenever $ab = 0$ for some $a \in A_i$ and $b \in A_j$ then $a = 0$ or $b = 0$.

Note that K is a graded K-algebra: the grading is given by

$$K_i = \begin{cases} K & \text{if } i = 0 \\ 0 & \text{if } i \neq 0 \end{cases}$$

Example 6.2.0.6
Let $R = K[x_1, x_2, ..., x_n]$, be the ring of polynomials in n indeterminates over a field K. Let

$R_j = 0$ for all $j > 0$,

$R_0 = K$, and

$R_j =$ the set of all homogeneous polynomials of *degree* $-j$ if $j < 0$. Then R is a graded K-algebra and a graded integral domain, with the **negative grading**.

Note that in R, if $dim(f) = j$, *i.e.*, $f \in R_j$ then degree of $f = -j$. **From now on R will be graded in the negative way above, where K is a field of characteristic two, unless otherwise indicated**.

Definition 6.2.3. Let R be a graded K-algebra. A **(left) graded** R-module M is a graded K-module, together with a sequence $\phi : R_i \otimes M_j \to M_{i+j}$ of K-homomorphisms, for $i, j \in \mathbb{Z}$ such that the following diagrams:

$$\begin{tikzcd}
R_i \otimes R_j \otimes M_m \arrow[r, "\pi\otimes 1"] \arrow[d, "1\otimes\phi"'] & R_{i+j} \otimes M_m \arrow[d, "\phi"] \\
R_i \otimes M_{j+m} \arrow[r, "\phi"] & M_{i+j+m}
\end{tikzcd}$$

$$\begin{tikzcd}
K \otimes M_j \arrow[r, equal] \arrow[d, "\mu\otimes 1"'] & M_j \arrow[d, equal] \\
R_0 \otimes M_j \arrow[r, "\phi"'] & M_j
\end{tikzcd}$$

commute for $i, j, m \in \mathbb{Z}$ where $\mu : K \longrightarrow R_0$ here $R_0 = K$, $(k \otimes m) \mapsto km \mapsto km$ and $(k \otimes m) \mapsto (\mu(k) \otimes m) \mapsto \phi(\mu(k) \otimes m) = km$ is the map given by the definition. We denote this by $M = \oplus \sum_{i=-\infty}^{\infty} M_i$. Similarly, we can define the **right graded** R-modules. If R is commutative, we may regard left R-modules as right R-modules, and vice versa. If $m \in M_j$ define $dim(m) = j$.

Definition 6.2.4. Let $M = \oplus \sum\limits_{i=-\infty}^{\infty} M_i$ and $N = \oplus \sum\limits_{i=-\infty}^{\infty} N_i$ be a graded R-modules. **A map of degree** P from M to N is a family $F = \{f_n : M_n \to N_{n+P}, n \in \mathbb{Z}\}$ of R-module homomorphisms such that $F(rm) = rF(m)$, for $r \in R$ and $m \in M$.

Note that we will consider all elements in R to be homogeneous, so if we write $a \in R$, we mean $a \in R_i$ for some $i \in \mathbb{Z}$.

Definition 6.2.5. **A differential graded** (DG) R-module M of degree P is a graded R-module with an R-module homomorphism $\partial : M \to M$ of degree P such that $\partial^2 = 0$.

Definition 6.2.6. A graded R-module M is is said to be **generated** by a set $S = \cup_{i=-\infty}^{\infty} S_i$, where $S_i \subseteq M_i$ for all i, if every element $g \in M_i$ can be written as follows:

$$g = \sum r_j s_j, \text{ where } r_j \in R \text{ and } s_j \in S \text{ such that } dim(r_j) + dim(s_j) = i. \quad (6.2.1)$$

The set S is called a **generating set** for M. Moreover, M is said to be **finitely generated** if it has a finite generating set S. M is **free** if there exists a generating set S such that every $g \in M_i$ can be **uniquely** expressed as in (6.2.1) above.

Here we give an example of DG R-module, and also an illustration of a construction of Carlsson's in [15].

Example 6.2.0.7
[64] Let $K = \mathbb{Z}/2$, and $G = \mathbb{Z}/2 \cong \{1, a\}$, where $a^2 = 1$.

Let $R = K[x]$ be the graded polynomial ring in one variable of dimension -1 over K. Define the chain complex C_* by

$$0 \to 0 \to C_1 \xrightarrow{\delta_1} C_0 \xrightarrow{\delta_0} 0 \to 0$$

where; $C_1 \cong \mathbb{Z}/2 \oplus \mathbb{Z}/2, \quad C_0 \cong \mathbb{Z}/2 \oplus \mathbb{Z}/2,$

$$\delta_1(1,0) = (0,1) + (1,0) \quad (6.2.2)$$

$$\delta_1(0,1) = (1,0) + (0,1) \text{ and}$$

$$\delta_i \equiv 0 \text{ for } i \neq 1.$$

Clearly $\delta_{j-1} \circ \delta_j = 0$ for all j, and the matrix of δ_1 with respect to the basis $\{(1,0),(0,1)\}$ is

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \text{ since } K = \mathbb{Z}/2$$

For $i = 0, 1$, define an action of G on C_i by

$a(1,0) = (1,0)$ and

$a(0,1) = (1,0)$. Then

$a\delta_1 = \delta_1 a$ for $i = 0, 1$ and hence for all i. i.e., C_* is a chain complex of $K[G]$ -modules.

Denote by $(1,0)_0$ and $(0,1)_0$ for the generators of C_0, and similarly $(1,0)_1$ and $(0,1)_1$ for the generators of C_1. Since $C_i \cong \mathbb{Z}/2 \oplus \mathbb{Z}/2 \cong \mathbb{Z}/2[G]$ for $i = 0, 1$; C_i is a free $\mathbb{Z}/2[G]$-module for all i, and a basis for C_0 is $\{e_{0,1} = (1,0)_0, e_{0,2} = (0,1)_0\}$. Similarly a basis for C_1 is $\{e_{1,1} = (1,0)_1, e_{0,2} = (0,1)_1\}$.

From this chain complex C_*, Carlsson constructs in [15] a free differential graded module M over the graded polynomial ring $K[X]$ as follows:

Let $M_i = 0$ for $i \geq 2$

$$\begin{aligned}
&M_1 = 0 \cdot C_0 \oplus 1 \cdot C_1\\
&M_0 = 1 \cdot C_0 \oplus x \cdot C_1\\
&M_{-1} = x \cdot C_0 \oplus x^2 \cdot C_1\\
&M_{-2} = x^2 \cdot C_0 \oplus x^3 \cdot C_1\\
&\vdots\\
&M_{-j} = x^j \cdot C_0 \oplus x^{j+1} \cdot C_1\\
&M_{-j-1} = x^{j+1} \cdot C_0 \oplus x^{j+2} \cdot C_1\\
&\vdots
\end{aligned}$$

One can see that, for $j \geq -1$, the map

$R_{-i} \otimes M_{-j} \longrightarrow M_{-(i+j)}$

$ax^i \otimes (x^j \cdot c_0\ ,\ x^{j+1} \cdot c_1) \longmapsto (\alpha c_0 x^{i+j}\ ,\ \alpha c_1 x^{i+j+1})$

defines an R-module structure on M.

For $j \geq -1$, define $\partial_{-j} : M_{-j} \longrightarrow M_{-(j+1)}$ by

$$\partial_{-j}(x^j \cdot c_0, x^{j+1} \cdot c_1) = [x^{j+1}\delta_1(c_1) + x^{j+1}(a-1)c_0\ ,\ x^{j+2}(a-1)c_1] \tag{6.2.3}$$

where δ_1 as in equation (6.2.3) and a as in the assumption. Now

$$\begin{aligned}
&\partial_{-j-1} \circ \partial_{-j}(x^j \cdot c_0\ ,\ x^{j+1} \cdot c_1)\\
&\quad = \partial_{-j-1}[x^{j+1}\delta_1(c_1) + x^{j+1}(a-1)c_0\ ,\ x^{j+2}(a-1)c_1]\\
&\quad = [x^{j+2} \cdot \delta_1((a-1)c_1) + x^{j+2}(a-1)\delta_1(c_1)\ ,\ x^{j+3}(a-1)(a-1)c_1]\\
&\quad = [x^{j+2} \cdot [\delta_1 a(c_1) - \delta_1(c_1)] + x^{j+2} \cdot [a\delta_1(c_1) - \delta_1(c_1)]\ ,\ x^{j+3}(a^2-1)c_1]\\
&\quad = [x^{j+2} \cdot [a\delta_1(c_1) - \delta_1(c_1) + a\delta_1(c_1) - \delta_1(c_1)]\ ,\ x^{j+3}(a^2-1)c_1], \text{ (since } a\delta_1 = \delta_1 a)\\
&\quad = 0 \quad (\text{since } a^2 = 1 \text{ and } K = \mathbb{Z}/2).
\end{aligned}$$

Let $e_1 = e_{1,1}$, $e_2 = e_{1,2}$, $e_3 = e_{0,1}$ and $e_4 = e_{0,2}$. If $m \in M_{-j}$, then m can be written **uniquely** as

$$m = x^j \cdot c_0 + x^{j+1}c_1 \text{ for some } c_0 \in C_0 \text{ and } c_1 \in C_1.$$

But $c_0 = \alpha_1 e_{0,1} + \alpha_2 e_{0,2}$ and $c_1 = \beta_1 e_{1,1} + \beta_2 e_{1,2}$ for some α_i *'s* and β_i *'s* in $K[G]$. Therefore,

$$\begin{aligned} m &= x^j(\alpha_1 e_{0,1} + \alpha_2 e_{0,2}) + x^{j+1}(\beta_1 e_{1,1} + \beta_2 e_{1,2}) \\ &= (x^j\alpha_1)e_{0,1} + (x^j\alpha_2)e_{0,2} + (x^{j+1}\beta_1)e_{1,1} + (x^{j+1}\beta_2)e_{1,2} \\ &= (x^{j+1}\beta_1)e_1 + (x^{j+1}\beta_2)e_2 + (x^j\alpha_1)e_3 + (x^j\alpha_2)e_4, \end{aligned}$$

and hence $\gamma = \{e_i\}_{i=1}^4$ is an R-basis for M, and M, is a **free** DG R-module.

Example 6.2.0.8

[63] Let R be a differential graded algebra and M and N be DG R-modules. Suppose $f : M \to N$ is a morphism of DG R-modules. Then $ker(f)$, $coker(f)$, $im(f)$ and $coim(f)$ are also DG R-modules.

Let M be free finite generated differential graded $R-$module of degree -1 with basis S and differential ∂. Then S can be written as a finite union $\cup_{i=1}^m S_{ki}$. So there exist two integers $t > r$ such that $M_i = 0$ for $i > tj$ and $s_j = \phi$ for $j > t$ and $j \leq r$. Thus we get the following diagram:

$$\begin{array}{llllllllll} M: 0 & \longrightarrow \cdots \longrightarrow & 0 & \longrightarrow M_t & \longrightarrow M_{t-1} & \longrightarrow \cdots \longrightarrow & M_{r+1} & \longrightarrow \cdots \\ \quad\ \cup & \cdots & \cup & \quad\ \cup & \quad\ \cup & \cdots & \cup & & \cdots \\ S:\ \phi & \cdots & \phi & \quad\ S_t & \quad\ S_{t-1} & \cdots & S_{r+1} & \phi & \cdots \end{array}$$

Note that some of $\{S_j\}_{j=r+1}^t$ could be ϕ.

To make the last diagram clear, Let as consider the following example.

Example 6.2.0.9

Let $R = K[x, y]$. Then $0 = R_1 = R_2 = \cdots, R_0 = K$ and R_{-1} is the set of all homogeneous polynomials of degree 1, R_{-2} is the set of all homogeneous polynomials of degree 2, and so on. Hence $x^3y \in R_{-4}$ and of degree 4 but dimension -4. Now, let M be a left R-module with basis $\{e_1, e_2\}$. suppose $e_1, e_2 \in M_T$ for some T, so $S_T = \{e_1, e_2\}$ and $S_i = \phi$ if $i \neq T$.

Note that $dim(am) = dim(a)+dim(m)$, where $a \in R$, $m \in M$. If $g \in M_T$, then g can be written uniquely as $g = ae_1+be_2$. Thus $T = dim(ae_1) = dim(a)+dim(e_1) = dim(a)+T$, so $dim(a) = 0, i.e., a \in K$. Similarly, $b \in K$. Therefore $M_T = Ke_1 \oplus Ke_2$.

If $g \in M_j$ and $j > T$, then g can be written uniquely as : $g = ae_1 + be_2$. Thus $j = dim(ae_1) = dim(a) + dim(e_1) = dim(a) + T$. So $dim(a) = j - T > 0$. Hence $a \in R_{j-T} = 0$. Similarly $b = 0$ Therefore, $M_j = 0$ for $j > T$.

If $g \in M_j$ and $j < T$, then g can be written uniquely as : $g = ae_1 + be_2$, and hence $j = dim(ae_1) = dim(a) + dim(e_1) = dim(a) + T$. Then $dim(a) = j - T < 0$. Hence $a \in R_{j-T}$. Similarly, $b \in R_{j-T}$ Therefore, $M_j = R_{j-T}e_1 \oplus R_{j-T}e_2$ for $j < T$. Hence, we get

$$\begin{array}{llllllll} M: 0 \longrightarrow & \cdots \longrightarrow & 0 \longrightarrow & M_T & \xrightarrow{\partial_T} & M_{T-1} \xrightarrow{\partial_{T-1}} & M_{r+1} \longrightarrow & \cdots \\ \quad\ \bigcup\vert & \cdots & \bigcup\vert & \bigcup\vert & & \bigcup\vert & \bigcup\vert & \\ S: \ \phi & \cdots & \phi & S_T = \{e_1, e_2\} & & \phi & \phi & \cdots \end{array}$$

Suppose that M is a free finitely generated differential graded R-module of degree -1 with basis S, and differential ∂. Let $L =$ the total number of elements in the R-basis S. Then ∂ will be completely determined by an $L \times L$ matrix as in the diagram Δ of Figure 6.1:

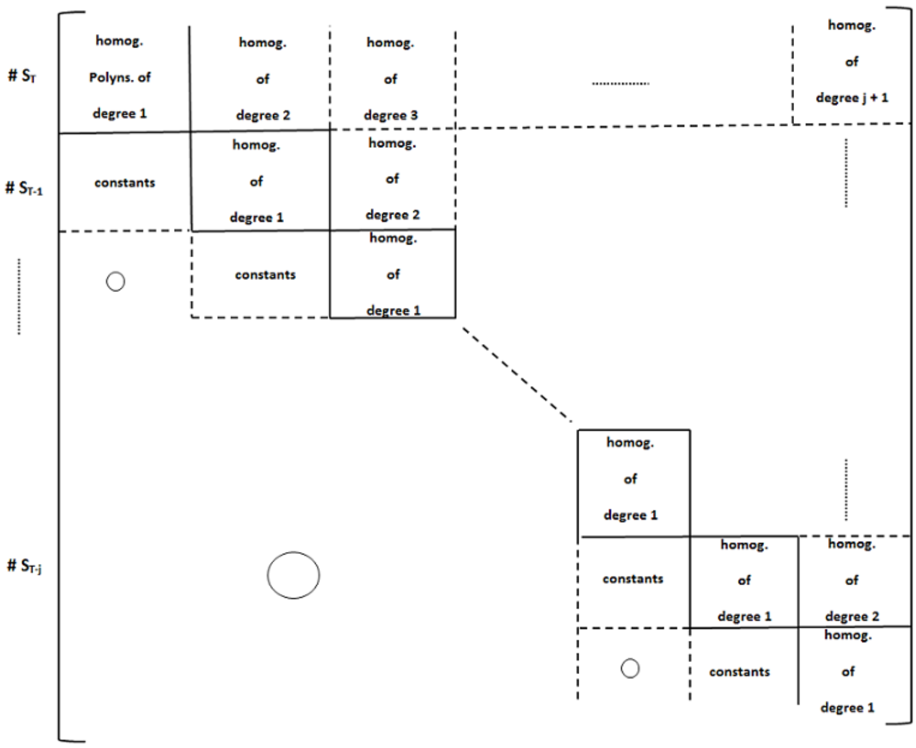

Figure 6.1: Diagram Δ

with $\partial^2 = 0$.

Note that, some of the constants could be equal zeros. Also, some of the homogeneous polynomials may be equal to zero.

Similarly, if degree of ∂ equal $-j$ such that $j \geq 0$, then we can see the matrix of ∂ with respect to the basis S as in the diagram $\Lambda.1$ of Figure 6.2:

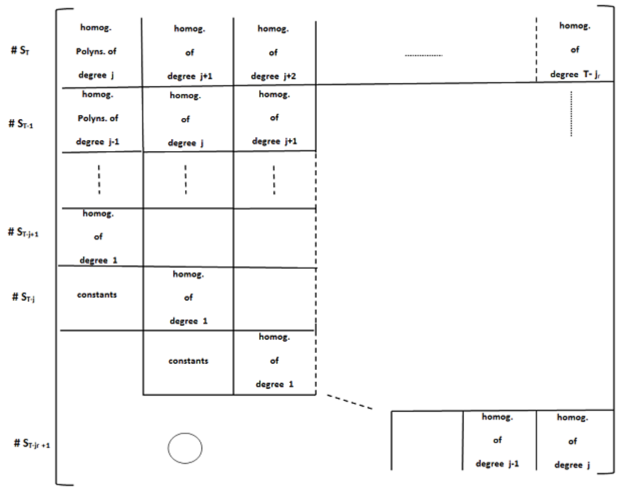

Figure 6.2: Diagram $\Lambda.1$

with $\partial^2 = 0$.

Finally, if degree of ∂ equal is j such that $j > 0$, then we can see the matrix of ∂ with respect to the basis S as in the diagram $\Lambda.2$ of Figure 6.3::

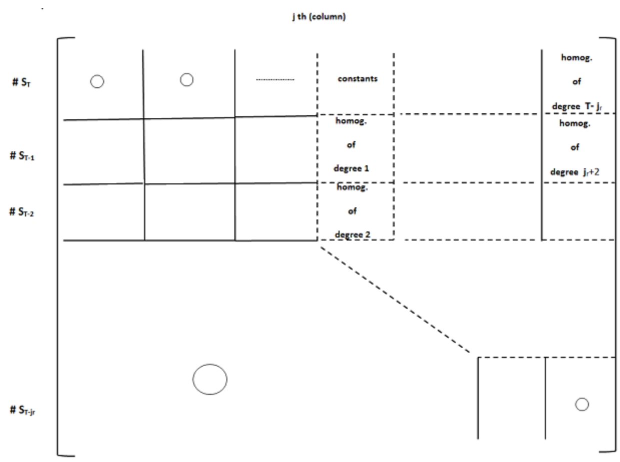

Figure 6.3: Diagram $\Lambda.2$

Chapter 7

Solvable Differential Graded Modules

Let K be a field of characteristic two, $R = K[x_1, x_2, \ldots, x_n]$ is a graded ring of polynomials graded in the negative way, and M be a free finitely generated differential graded R-module of degree P such that $(P \leq -2)$. We will give an example that M is not necessarily solvable when $(P \leq -2)$.

In this Chapter we will construct a classification for some types of differential graded R-modules, based on the degree P of the differential module and dimension of the module. This classification gives a partial algorithm to test whether such modules are solvable. For modules outside the classification we cannot decide, using our methods, whether or not they are solvable.

7.1 Composition Series

We will describe in this section the composition series by giving a definition for this series as well as give some of the concepts and definitions and theories that will help us in our study of differential graded modules.

Definition 7.1.1. By [64] Let M be a finitely generated free DG R-module of degree P. **A composition series** for M is a sequence of free DG R-modules

$$0 = C_0 \subset C_1 \subset \ldots \subset C_q = M$$

such that (C_j/C_{j-1}) is free DG R-modules, whose differential is identically zero i.e., $\partial(C_j/C_{j-1}) = 0$. The length H of the series is called the **composition length**.

Any module having a basis of size t is isomorphic to any other module having a basis of size t. If $\pi : M \longrightarrow F$ is a surjective homomorphism from an R-module to a free module F then $M \cong Ker(\pi) \oplus F$. Therefore, if M has a composition series, as in Definition 7.1.1 then $C_j \cong C_{j-1} \oplus (C_j/C_{j-1}),\ \forall\ j$ (see [42]).

Suppose M is finitely generated free DG R-module of degree P. M has basis $S = S_T \cup \ldots \cup S_{T-k}, T, k \geq 0$. If $g \in M_T$ then $g = \sum r_j s_j$ where $dim(r_j) + dim(s_j) = T$. As $s_j \in S$ we have $dim(s_j) \leq T$ and as $dim(r_j) \leq 0$ we have $dim(s_j) = T - dim(r_j) \geq T$. This holds $\forall\ j$, so M is generated by S_T.

Similarly, if $g \in M_{T+s}$, where $s \neq 0$ we have $g = \sum r_j s_j$ with $dim(r_j) + dim(s_j) = T + s$. So $T \geq dim(s_j) = T + s - dim(r_j) \geq T + s$ (as $-dim(r_j) \geq 0$). If $s > 0$, it follows that $M_{T+s} = 0$, while if $s < 0$ then, setting $t = -s$, M_{T-t} is generated by $S_{T-t} \cup \ldots \cup S_T$.

Note that, as M_{T-t} is generated by $S_{T-t} \cup \ldots \cup S_T$, it is also a finitely generated free DG R-module for $t = 0, \ldots, k$. (M_{T-t} is free on $S_{T-t} \cup \ldots \cup S_T$, since M is free on S.)

Suppose M has a composition series $0 = C_0 \subset C_1 \subset \ldots \subset C_q = M$. Then C_j is finitely generated free; so has a finite basis S_j, say $j = 0, \ldots, H$. Then $S_j = \cup_{t=0}^{\infty} S_i$, where $(S_j)_{T-t} \in M_{T-t}$; so $(C_j)_{T-t}$ is free on $(S_j)_{T-t}$. Moreover we have a sequence of free DG R-modules,

$$\forall t\ 0 = (C_0)_{T-t} \subseteq (C_1)_{T-t} \subseteq \ldots \subseteq (C_{q-1})_{T-t} \subseteq (C_q)_{T-t} = M_{T-t}.$$

Also, as C_j/C_{j-1} is free, so is $(\frac{C_j}{C_{j-1}})_{T-t} = (C_j)_{T-t}/(C_{j-1})_{T-t}$, for all $t \geq 0$.

Finally as $\partial(C_j/C_{j-1}) = 0$ we have $\partial((C_j)_{T-t}/(C_{j-1})_{T-t}) = 0\ \forall\ t$. Therefore, M_{T-t} has a composition series. So $M_T, M_{T-1}, \ldots$ are free DG R-modules and also C_q/C_{q-1} is free DG R-modules by the definition.

For a special case if degree ∂ is -1, i.e, $P = -1$ then we have that,

$$\begin{array}{lr}
 & 0 \\
 & \downarrow \\
0 = (C_0)_T \subset (C_1)_T \subset \ldots \subset (C_{q-1})_T \subset (C_q)_T & = M_T \\
 & \downarrow \partial_T \\
0 = (C_0)_{T-1} \subset (C_1)_{T-1} \subset \ldots \subset (C_{q-1})_{T-1} \subset (C_q)_{T-1} & = M_{T-1} \\
 & \downarrow \partial_{T-1} \\
0 = (C_0)_{T-2} \subset (C_1)_{T-2} \subset \ldots \subset (C_{q-1})_{T-2} \subset (C_q)_{T-2} & = M_{T-2} \\
 & \downarrow \partial_{T-2}
\end{array}$$

$$\vdots$$

Therefore, $\partial(C_1) = 0$, i.e., $\partial(C_1) \subseteq C_0 = \{0\}$, $\partial(C_2) \subseteq C_1, \dots \partial(C_q) \subseteq C_{q-1}$. So one can note that, the matrix ∂ with respect to the basis S is a strictly upper triangular.

In the general case, if degree $\partial = -j$ such that $j > 0$, then

$$\begin{array}{ll}
 & 0 \\
 & \downarrow \\
0 = (C_0)_T \subset (C_1)_T \subset \dots \subset (C_{q-1})_T \subset (C_q)_T & = M_T \\
 & \downarrow \partial_T \\
0 = (C_0)_{T-j} \subset (C_1)_{T-j} \subset \dots \subset (C_{q-1})_{T-j} \subset (C_q)_{T-j} & = M_{T-j} \\
 & \downarrow \partial_{T-j} \\
0 = (C_0)_{T-2j} \subset (C_1)_{T-2j} \subset \dots \subset (C_{q-1})_{T-2j} \subset (C_q)_{T-2j} & = M_{T-2j} \\
 & \downarrow \partial_{T-2j} \\
 & \vdots
\end{array}$$

Then $(C_q)_j/(C_{q-1})_{j-1}$ is free as C_j/C_{j-1} is free and $\partial_T((C_q)_T/(C_{q-1})_T) = 0$, which means $\partial_T((C_q)_T) \subset (C_{q-1})_{T-j}$. So M_j has composition series as follows,

$$0 = (C_0)_j \subseteq (C_1)_j \subseteq \dots \subseteq (C_{q-1})_j \subseteq (C_q)_j = M_j.$$

Therefore, the matrix of ∂ with respect to the basis S is a strictly upper triangular with the diagonal elements which are zeros.

Example 7.1.0.10

Let K be a field and $R = K[x]$, be a polynomials ring with one variable over the field K. Let M be a graded R-module with basis $S = \{e_1, e_2, e_3, e_4\}$, such that $\{e_1, e_2\}$ have the same dimension T, while $\{e_3, e_4\}$ have dimension $T - 1$. Then M is graded as follows:

$$\begin{array}{l}
\quad\quad\;\, 0 \\
\quad\quad\;\, \downarrow \\
e_1, e_2 \in M_T = k \cdot e_1 \oplus k \cdot e_2. \\
\quad\quad\;\, \downarrow \\
e_3, e_4 \in M_{T-1} = R_{-1} \cdot e_1 \oplus R_{-1}.e_2 \oplus k \cdot e_3 \oplus k \cdot e_4. \\
\quad\quad\;\, \downarrow \\
\quad\quad\;\, \vdots
\end{array}$$

$$\begin{array}{l}\downarrow \\ M_{T-i} = R_{-i} \cdot e_1 \oplus R_{-i}.e_2 \oplus R_{-i} \cdot e_3 \oplus R_{-i}.e_4. \\ \downarrow \\ \vdots\end{array}$$

We define the differential operator ∂ with respect to the basis S as follows: $\partial(e_1) = 0$,

$\partial(e_2) = 0$,

$\partial(e_3) = x^2e_1 + x^2e_2$, and

$\partial(e_4) = x^2e_1 + x^2e_2$.

Then the matrix of ∂ with respect to the basis S is given by:

$$\partial = \left[\begin{array}{cc|cc} 0 & 0 & x^2 & x^2 \\ 0 & 0 & x^2 & x^2 \\ \hline 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array}\right]$$

It's clear that $\partial^2 = 0$.

Now, $dim(\partial(e_3)) = dim(x^2e_1) = dim(e_1) + dim(x^2) = T - 2$. Similarly, $dim(e_3) = dim(e_4) = T - 1$, while $dim(\partial(e_4)) = T - 2$, so degree of the differential ∂ is equal to -1.

Let $(C_0) = 0$, $(C_1) = \langle e_1, e_2\rangle$ over $R = \{f_1e_1 + f_2e_2 : f_1, f_2 \in R\}$, and $(C_2) = M$. Thus, $(C_1/C_0) = \langle e_1, e_2\rangle$. But, $\partial(e_1) = \partial(e_2) = 0$, So $\partial(C_1/C_0) = 0$. Now, $(C_2/C_1) = \langle e_3, e_4\rangle$, but $\partial(e_3) = \partial(e_4) = x^2e_1 + x^2e_2 \in C_1$, also $\partial((C_2/C_1)) = \partial(C_1) = 0$. Therefore, we have that $0 = (C_0) \subset (C_1) \subset (C_2) = M$ which is a composition series of M.

Note: If M has a composition series, then the matrix of ∂ is similar to the upper triangular matrix has its diagonal zeros and we call it a strictly upper triangular matrix.

Example 7.1.0.11

Let M as in the previous example, and the differential operator ∂ with respect to the basis $S = \{e_1, e_2, e_3, e_4\}$ has the following form:

$$\partial = \begin{bmatrix} x & x & 0 & 0 \\ x & x & 0 & 0 \\ 1 & 1 & x & x \\ 1 & 1 & x & x \end{bmatrix}$$

Then $\partial^2 = 0$ and the differential operator ∂ has degree is -1. Let $\beta_1 = e_3 + e_4$, $\beta_2 = e_1 + e_2$, $\beta_3 = e_2$, and $\beta_4 = e_3$.

We claim that: $\beta_1, \beta_2, \beta_3$ *and* β_4 form a basis to M over R. We will show that:

Let $m \in M_j$. Then, $m = \alpha_1 e_1 + \alpha_2 e_2 + \alpha_3 e_3 + \alpha_4 e_4$. Hence, $m = \alpha_4 \beta_1 + \alpha_1 \beta_2 + (\alpha_1 + \alpha_2)\beta_3 + (\alpha_3 + \alpha_4)\beta_4$. Also, if $\alpha_1\beta_1 + \alpha_2\beta_2 + \alpha_3\beta_3 + \alpha_4\beta_4 = 0$, then $\alpha_1(e_3+e_4)+\alpha_2(e_1+e_2)+\alpha_3 e_2+\alpha_4 e_3 = 0$. Thus, $\alpha_2 e_1+(\alpha_2+\alpha_3)e_2+(\alpha_1+\alpha_4)e_3+\alpha_1 e_4 = 0$. But, $\{e_1, e_2, e_3, e_4\}$ is a basis for M, also $\alpha_2 = \alpha_1 = 0$ and $\alpha_2 + \alpha_3 = \alpha_1 + \alpha_4 = 0$ this implies that $\alpha_1 = \alpha_2 = \alpha_3 = \alpha_4 = 0$. So, $\{\beta_1, \beta_2, \beta_3, \beta_4, \}$ is a basis to M.

Now, $\partial(\beta_1) = \partial(e_3)+\partial(e_4) = (xe_3+xe_4)+(xe_3+xe_4) = 0$, $\partial(\beta_2) = \partial(e_1)+\partial(e_2) = (xe_1+xe_2+e_3+e_4)+(xe_1+xe_2+e_3+e_4) = 0$, $\partial(\beta_3) = \partial(e_2) = xe_1+xe_2+e_3+e_4 = \beta_1 + x\beta_2$ and $\partial(\beta_4) = \partial(e_3) = xe_3 + xe_4 = x\beta_1$. Hence, the matrix ∂ with respect to the basis $\{\beta_1, \beta_2, \beta_3, \beta_4\}$ is given by:

$$\partial^* = \begin{bmatrix} 0 & 0 & 1 & x \\ 0 & 0 & x & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Let $(C_0) = 0, (C_1) = \langle \beta_1, \beta_2 \rangle$, and $(C_2) = M$, then one can be easily shows that M has a composition series.

Theorem 7.1.2. *[15] Let M be a free finitely generated differential graded R-module with differential ∂ of degree $P = -1$, then M has a composition series.*

Remark 7.1.3. If M admits a composition series, then we say that M is solvable.

Remark 7.1.4. Let M be any DGR-modules of rank 1 and ∂ be a differential on M of any degree. Then the matrix of ∂ with respect to the basis $\{e_1\}$ is given by $\partial = [a],\ a \in R$. But $\partial^2 = 0$ which implies that $a = 0$. Then M has a composition series. From now we will only consider DG $R-modules$ of rank greater than 1.

In our work we will use the following lemma:

Lemma 7.1.5. *[64] Let M be a free finitely generated differential graded R-module with differential ∂ and basis $S = \{e_i\}_{i=1}^m$. consider the following elementary row and column operations, on the matrix of ∂ with respect to this basis:*

(1) Exchange $row(i)$ and $row(j)$, and at the same time exchange $column(i)$ and $column(j)$.

(2) Replace row (j) by row $(j)+g(row(i))$ and at the same time replace $column(i)$ by $column(i) + g(column(j))$, where $g \in R$ and $deg(g) = dim(e_j)$ - $dim(e_i)$. Then each of these operations corresponds to a change of basis in M.

Remark 7.1.6. Since the characteristic of the field which we deal with it is two, then (-) is (+), thus the step (2) of Lemma 7.1.5 becomes that:

(Replace $row(j)$ by $row(j) - g(row(i))$ and at the same time replace $column(i)$ by $column(i) - g(column(j))$, where $g \in R$ and $deg(g) = dim(e_j) - dim(e_i)$. Then each of these operations corresponds to a change of basis in M).

Remark 7.1.7. If the matrix of ∂ with respect to basis S is a strictly upper triangular matrix, then M is solvable.

7.2 Solvable differential Graded Modules

In the following example we show that if $R = K[x_1, x_2, \ldots, x_n]$, $n \geq 2$ and M is a free finitely generated differential graded R-module with differential ∂ of degree $P \leq -2$, then M is not necessarily solvable.

Example 7.2.0.12
Let $R = K[x_1, x_2, \ldots, x_n]$ be a graded ring of polynomials graded in the negative way and M be a free finitely generated differential graded R-module of dimension four with basis $\{e_1, e_2, e_3, e_4\}$. Suppose the differential ∂ on M has degree ($P \leq -2$), and its matrix with respect to $\{e_1, e_2, e_3, e_4\}$ is

$$\partial = \begin{bmatrix} x_1x_2^{m-1} & 0 & 0 & x_1^2x_2^{m-2} \\ 0 & x_1x_2^{m-1} & x_1^2x_2^{m-2} & 0 \\ 0 & x_2^m & x_1x_2^{m-1} & 0 \\ x_2^m & 0 & 0 & x_1x_2^{m-1} \end{bmatrix}$$

Clearly, $\partial^2 = 0$.

We suppose M has a composition series. Then there exists an invertible matrix $B = \{f_{ij}\}_{i,j=1}^{4}$, and strictly upper triangular matrix ∂' such that $\partial \cdot B = B \cdot \partial'$, i.e.,

$$\begin{bmatrix} x_1x_2^{m-1} & 0 & 0 & x_1^2x_2^{m-2} \\ 0 & x_1x_2^{m-1} & x_1^2x_2^{m-2} & 0 \\ 0 & x_2^m & x_1x_2^{m-1} & 0 \\ x_2^m & 0 & 0 & x_1x_2^{m-1} \end{bmatrix} \begin{bmatrix} f_{11} & f_{12} & f_{13} & f_{14} \\ f_{21} & f_{22} & f_{23} & f_{24} \\ f_{31} & f_{32} & f_{33} & f_{34} \\ f_{41} & f_{42} & f_{43} & f_{44} \end{bmatrix} =$$

$$\begin{bmatrix} f_{11} & f_{12} & f_{13} & f_{14} \\ f_{21} & f_{22} & f_{23} & f_{24} \\ f_{31} & f_{32} & f_{33} & f_{34} \\ f_{41} & f_{42} & f_{43} & f_{44} \end{bmatrix} \begin{bmatrix} 0 & g_1 & g_2 & g_3 \\ 0 & 0 & g_4 & g_5 \\ 0 & 0 & 0 & g_6 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Multiply $row(1)$ by $column(1)$ to get, $x_1x_2^{m-1}f_{11} + x_1^2x_2^{m-2}f_{41} = 0$ which implies that $x_1x_2^{m-1}f_{11} = x_1^2x_2^{m-2}f_{41}$ (since K is of characteristic 2) and this implies that $x_2f_{11} = x_1f_{41}$.

Now, $x_2 \mid x_1f_{41}$ implies that $x_2 \mid f_{41}$, say $f_{41} = x_2g_4$. Similarly $f_{11} = x_1g_1$.

In a similar way multiply $row(2)$ with $column(1)$ to get, $x_1x_2^{m-1}f_{21}+x_1^2x_2^{m-2}f_{31} = 0$, which implies that $x_1x_2^{m-1}f_{21} = x_1^2x_2^{m-2}f_{31}$ (since K is of characteristic 2) which implies that $x_2f_{21} = x_1f_{31}$ which implies that $x_2 \mid x_1f_{31}$ which implies that $x_2 \mid f_{31}$, say $f_{31} = x_2g_3$. Similarly, $f_{21} = x_1g_2$. Thus, $f_{j1}(0, 0, \dots, 0) = 0$ for $j = 1, 2, 3, 4$.

Now since B is an invertible, there exists

$$B^{-1} = \begin{bmatrix} h_{11} & h_{12} & h_{13} & h_{14} \\ h_{21} & h_{22} & h_{23} & h_{24} \\ h_{31} & h_{32} & h_{33} & h_{34} \\ h_{41} & h_{42} & h_{43} & h_{44} \end{bmatrix}$$

such that $BB^{-1} = I$.

Therefore, $h_{11}f_{11} + h_{12}f_{21} + h_{13}f_{31} + h_{14}f_{41} = 1$.

Now, by evaluating both sides at $(0, 0, \dots, 0)$ we will get that $0 = 1$, which is a contradiction. So M does not have a composition series. Hence M is not solvable.

Proposition 7.2.1. *Let K be a field and let $R = K[x_1, x_2, \dots, x_n]$ be a graded ring of polynomials graded graded in the negative way. Let M be a free finitely generated differential graded R-module with basis $S = \{e_1, e_2\}$, and with differential ∂ of degree*

$P \leq -2$. Suppose, $dim(e_1) = k_1$ and $dim(e_2) = k_2$, such that $k_1 > k_2$. If $k_1 - k_2 = t$ such that $t \geq -P$, then M is solvable.

Proof. M is graded as follows:

$$\begin{array}{l} 0 \\ \downarrow \\ e_1 \in M_{k_1} = K \cdot e_1 \oplus 0 \cdot e_2. \\ \downarrow \\ \vdots \\ \downarrow \\ e_2 \in M_{k_2} = R_{k_2 - k_1} \cdot e_1 \oplus k.e_2. \\ \downarrow \\ \vdots \\ \downarrow \\ e_j \in M_{k_j} = R_{k_j - k_1} \cdot e_1 \oplus R_{k_j - k_2} \cdot e_2 \oplus \ldots \oplus k.e_m. \\ \downarrow \\ \vdots \end{array}$$

Suppose that,

$$\begin{aligned} \partial(e_1) &= f_{11}e_1 + f_{21}e_2 \\ \partial(e_2) &= f_{12}e_1 + f_{22}e_2 \end{aligned}$$

Then the matrix of ∂ with respect to the basis $\{e_i\}_{i=1}^{2}$ is given by:

$$\partial = \begin{bmatrix} f_{11} & f_{12} \\ f_{21} & f_{22} \end{bmatrix}$$

Now,

$$dim(\partial(e_1)) = dim(f_{11}) + dim(e_1),$$

$$k_1 + P = dim(f_{11}) + k_1, \text{ implies that}$$

$$dim(f_{11}) = P < 0, \text{ and thus } deg(f_{11}) = -P.$$

So,

$$dim(\partial(e_1)) = dim(f_{21}) + dim(e_2),$$

$$k_1 + P = dim(f_{21}) + k_2, \text{ implies that}$$

$$dim(f_{21}) = P + k_1 - k_2 \geq P - P = 0, \text{ and thus}$$

$$f_{21} = C \neq 0 \text{ (constant) or } f_{21} = 0.$$

Also,

$$dim(\partial(e_2)) = dim(f_{12}) + dim(e_1),$$

$$k_2 + P = dim(f_{12}) + k_1, \text{ implies that}$$

$$dim(f_{12}) = P + k_2 - k_1 < 0, \text{ and thus } deg(f_{12}) = -(P + k_2 - k_1).$$

So,

$$dim(\partial(e_2)) = dim(f_{22}) + dim(e_2),$$

$$k_2 + P = dim(f_{22}) + k_2, \text{ implies that}$$

$$dim(f_{22}) = P + k_2 - k_2 = P, \text{ and thus } deg(f_{22}) = -P.$$

Hence, the matrix of ∂ is given by:

$$\partial = \begin{bmatrix} f_{11} & f_{12} \\ f_{21} & f_{22} \end{bmatrix}$$

where $f_{21} = 0$ or $f_{21} = C \neq 0$ (constant).

Case (1): If $f_{21} = 0$, then the matrix of ∂ is given by

$$\partial = \begin{bmatrix} f_{11} & f_{12} \\ 0 & f_{22} \end{bmatrix}$$

since $\partial^2 = 0$, implies that $f_{11}^2 = 0$ and $f_{22}^2 = 0$.

Thus, $f_{11} = 0$ and $f_{22} = 0$.

Therefore, the matrix of ∂ is given by:

$$\partial = \begin{bmatrix} 0 & f_{12} \\ 0 & 0 \end{bmatrix}$$

Note that, ∂ is strictly upper triangular matrix.

To show, M has a composition series:

Let $C_0 = 0$, $C_1 = \langle e_1 \rangle$ and $C_2 = \langle e_1, e_2 \rangle$.

Then C_j/C_{j-1} is free, for all $j = 1, 2$.

If $x \in C_2$, then

$x = \alpha_1 e_1 + \alpha_2 e_2$

So, $\partial(x) = \alpha_1 \partial(e_1) + \alpha_2 \partial(e_2)$

$\partial(x) = \alpha_1(0) + \alpha_2(f_{12} e_1) \ \in C_1$.

Thus, $\partial(C_2) \subseteq C_1$, and then $\partial(C_2/C_1) = 0$.

Also, if $x \in C_1$, then $x = \alpha_1 e_1$ and so,

$\partial(x) = \alpha_1 \partial(e_1) = \alpha_1(0) = 0 \in C_0$.

Hence, $\partial(C_1) \subseteq C_0$, and then $\partial(C_1/C_0) = 0$.

Therefore, $0 = C_0 \subseteq C_1 \subseteq C_2 = M$ is a composition series for M.

Hence, M is solvable.

Case (2): If $f_{21} = C \neq 0$,(constant), then the matrix of ∂ is given by:

$$\partial = \begin{bmatrix} f_{11} & f_{12} \\ C & f_{22} \end{bmatrix}$$

Since, $\partial^2 = 0$, we have that,

$f_{11}^2 + Cf_{12} = 0$ and $Cf_{12} + f_{22}^2 = 0$.

Hence, $f_{11} = f_{22}$ and $Cf_{12} = f_{11}^2$.

Now, by Lemma 7.1.5, replace $row(1)$ by $row(1) - (\frac{f_{11}}{C})row(2)$ and at the same time replace $column(2)$ by $column(2) - (\frac{f_{11}}{C})column(1)$ to get:

$$\partial = \begin{bmatrix} 0 & 0 \\ C & 0 \end{bmatrix}$$

By Lemma 7.1.5, replace $row(1)$ by $row(2)$ and at the time replace $column(2)$ by $column(1)$ to get:

$$\partial = \begin{bmatrix} 0 & C \\ 0 & 0 \end{bmatrix}$$

Therefore, M is solvable as before (Case 1). □

Proposition 7.2.2. *Let K be a field and let $R = K[x_1, x_2, \ldots, x_n]$ be a graded ring of polynomials graded in the negative way. Let M be a free finitely generated differential graded R-module with basis $S = \{e_i\}_{i=1}^3$ and with differential ∂ of degree $(P \leq -2)$. Suppose that, $dim(e_i) = k_i$ such that $1 \leq i \leq 3$ and $k_i > k_{i+1}$. If $k_i - k_{i+1} = t_i$ such that $t_i \geq -P$, then M is solvable.*

Proof. M is graded as follows:

$$
\begin{array}{l}
\qquad 0 \\
\qquad \downarrow \\
e_1 \in M_{k_1} = K \cdot e_1 \oplus 0 \cdot e_2 \oplus 0 \cdot e_3. \\
\qquad \downarrow \\
\qquad \vdots \\
\qquad \downarrow \\
e_2 \in M_{k_2} = R_{k_2-k_1} \cdot e_1 \oplus k.e_2 \oplus 0 \cdot e_3. \\
\qquad \downarrow \\
\qquad \vdots \\
\qquad \downarrow \\
e_3 \in M_{k_3} = R_{k_3-k_1} \cdot e_1 \oplus R_{k_3-k_2} \cdot e_2 \oplus k \cdot e_3. \\
\qquad \downarrow \\
\qquad \vdots \\
\qquad \downarrow \\
e_j \in M_{k_j} = R_{k_j-k_1} \cdot e_1 \oplus R_{k_j-k_2} \cdot e_2 \oplus R_{k_j-k_3} \cdot e_3 \oplus \ldots \oplus k.e_j. \\
\qquad \downarrow \\
\qquad \vdots
\end{array}
$$

Suppose that,

$$
\begin{aligned}
\partial(e_1) &= f_{11}e_1 + f_{21}e_2 + f_{31}e_3 \\
\partial(e_2) &= f_{12}e_1 + f_{22}e_2 + f_{32}e_3 \\
\partial(e_3) &= f_{13}e_1 + f_{23}e_2 + f_{33}e_3
\end{aligned}
$$

Then the matrix of ∂ with respect to the basis $\{e_1\}_{i=1}^3$ is given by:

$$
\partial = \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix}
$$

Now,

$$dim(\partial(e_1)) = dim(f_{11}) + dim(e_1),$$

$$k_1 + P = dim(f_{11}) + k_1, \text{ implies that}$$

$$dim(f_{11}) = P, \text{ and thus } deg(f_{11}) = -P.$$

So,

$$dim(\partial(e_1)) = dim(f_{21}) + dim(e_2),$$

$$k_1 + P = dim(f_{21}) + k_2, \text{ implies that}$$

$$dim(f_{21}) = P + k_1 - k_2 = P + t_1 \geq P - P = 0, \text{ and thus}$$

$$f_{21} = 0 \; or \; f_{21} = C \neq 0 \text{ (constant)}.$$

Also,

$$dim(\partial(e_1)) = dim(f_{31}) + dim(e_3),$$

$$k_1 + P = dim(f_{31}) + k_3, \text{ implies that}$$

$$dim(f_{31}) = k_1 - k_3 + P \geq -2P + P = -P, \text{ and thus } f_{31} = 0$$

Also,

$$dim(\partial(e_2)) = dim(f_{12}) + dim(e_1),$$

$$k_2 + P = dim(f_{12}) + k_1, \text{ implies that}$$

$$dim(f_{12}) = k_2 - k_1 + P \text{ and thus } deg(f_{12}) = -(k_2 - k_1 + P).$$

So,

$$dim(\partial(e_2)) = dim(f_{22}) + dim(e_2),$$

$$k_2 + P = dim(f_{22}) + k_2, \text{ implies that}$$

$$dim(f_{22}) = P + k_2 - k_2 = P, \text{ and thus } deg(f_{22}) = -P.$$

So,

$$dim(\partial(e_2)) = dim(f_{32}) + dim(e_3),$$

$$k_2 + P = dim(f_{32}) + k_3, \text{ implies that}$$

$$dim(f_{32}) = k_2 - k_3 + P \geq -P + P = 0, \text{ and thus}$$

$$f_{32} = 0 \; or \; f_{32} = \alpha \neq 0 \text{ (constant)}.$$

Also,

$$dim(\partial(e_3)) = dim(f_{13}) + dim(e_1),$$

$$k_3 + P = dim(f_{13}) + k_1, \text{ implies that}$$

$$dim(f_{13}) = k_3 - k_1 + P < 0 \text{ and thus } deg(f_{13}) = -(k_3 - k_1 + P).$$

So,

$$dim(\partial(e_3)) = dim(f_{23}) + dim(e_2),$$

$$k_3 + P = dim(f_{23}) + k_2, \text{ implies that}$$

$$dim(f_{23}) = P + k_3 - k_2 < 0 \text{ and thus } deg(f_{23}) = -(k_3 - k_2 + P).$$

So,

$$dim(\partial(e_3)) = dim(f_{33}) + dim(e_3),$$

$$k_3 + P = dim(f_{33}) + k_3, \text{ implies that}$$

$$dim(f_{33}) = P, \; and \; thus \;\; deg(f_{33}) = -P.$$

From the previous steps we can conclude the following:

1. $f_{31} = 0$,
2. $f_{21} = 0$ or $f_{21} = C \neq 0$ (constant),
3. $f_{32} = 0$ or $f_{32} = \alpha \neq 0$ (constant),
4. $deg(f_{11}) = deg(f_{22}) = deg(f_{33}) = -P$.

Hence,

$$\partial = \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ 0 & f_{32} & f_{33} \end{bmatrix}$$

Case (1): If $f_{21} = 0$, then the matrix of ∂ is given by

$$\partial = \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ 0 & f_{22} & f_{23} \\ 0 & f_{32} & f_{33} \end{bmatrix}$$

since $\partial^2 = 0$, implies that $f_{11}^2 = 0$ and then $f_{11} = 0$.
Thus,

$$\partial = \begin{bmatrix} 0 & f_{12} & f_{13} \\ 0 & f_{22} & f_{23} \\ 0 & f_{32} & f_{33} \end{bmatrix}$$

In this case either $f_{32} = 0$ or $f_{32} = \alpha \neq 0$ (constant).

Case (1.1): If $f_{32} = 0$, then the matrix of ∂ is given by

$$\partial = \begin{bmatrix} 0 & f_{12} & f_{13} \\ 0 & f_{22} & f_{23} \\ 0 & 0 & f_{33} \end{bmatrix}$$

since $\partial^2 = 0$, implies that, $f_{22}^2 = f_{33}^2 = 0$.
So, $f_{22} = f_{33} = 0$. (since R is an integral domain).
Thus,the matrix of ∂ is given by

$$\partial = \begin{bmatrix} 0 & f_{12} & f_{13} \\ 0 & 0 & f_{23} \\ 0 & 0 & 0 \end{bmatrix}$$

To show, M has a composition series:
Let $C_0 = 0$, $C_1 = \langle e_1 \rangle$, $C_2 = \langle e_1, e_2 \rangle$, and $C_3 = \langle e_1, e_2, e_3 \rangle$.
Then C_j/C_{j-1} is free, for all $1 \leq j \leq 3$.
If $x \in C_3$, then $x = \alpha_1 e_1 + \alpha_2 e_2 + \alpha_3 e_3$,
So, $\partial(x) = \alpha_1 \partial(e_1) + \alpha_2 \partial(e_2) + \alpha_3 \partial(e_3)$,
$\partial(x) = \alpha_1(0) + \alpha_2(f_{12}e_1) + \alpha_3(f_{13}e_1 + f_{23}e_2) \in C_2$.
Hence, $\partial(C_3) \subseteq C_2$, and then $\partial(C_3/C_2) = 0$.
Also, if $x \in C_2$, then $x = \alpha_1 e_1 + \alpha_2 e_2$
So, $\partial(x) = \alpha_1 \partial(e_1) + \alpha_2 \partial(e_2)$
$\partial(x) = \alpha_1(0) + \alpha_2(f_{12}e_1) \in C_1$.
Hence, $\partial(C_2) \subseteq C_1$, and then $\partial(C_2/C_1) = 0$.
Finally, if $x \in C_1$, then $x = \alpha_1 e_1$ and so,
$\partial(x) = \alpha_1 \partial(e_1) = \alpha_1(0) = 0 \in C_0$.
Hence, $\partial(C_1) \subseteq C_0$, and then $\partial(C_1/C_0) = 0$.
Therefore, $0 = C_0 \subseteq C_1 \subseteq C_2 \subseteq C_3 = M$ is a composition series for M.
Thus, M is solvable.

Case (1.2): If $f_{32} = \alpha \neq 0$ (constant), then the matrix of ∂ is given by:

$$\partial = \begin{bmatrix} 0 & f_{12} & f_{13} \\ 0 & f_{22} & f_{23} \\ 0 & \alpha & f_{33} \end{bmatrix}$$

Now, by Lemma 7.1.5, replace $row(2)$ by $row(2) - (\frac{f_{22}}{\alpha})row(3)$ and at the same time replace $column(3)$ by $column(3) - (\frac{f_{22}}{\alpha})column(2)$ to get:

$$\partial = \begin{bmatrix} 0 & f_{12} & \frac{\alpha f_{13} - f_{22} f_{12}}{\alpha} \\ 0 & 0 & 0 \\ 0 & \alpha & 0 \end{bmatrix}$$

By Lemma 7.1.5, replace $row(2)$ by $row(3)$ and at the time replace $column(3)$ by $column(2)$ to get:

$$\partial = \begin{bmatrix} 0 & \frac{\alpha f_{13} - f_{22} f_{12}}{\alpha} & f_{12} \\ 0 & 0 & \alpha \\ 0 & 0 & 0 \end{bmatrix}$$

Therefore, M is solvable as before (Case 1.1).

Case (2): If $f_{21} = C \neq 0$ (constant), then the matrix of ∂ is given by:

$$\partial = \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ C & f_{22} & f_{23} \\ 0 & f_{32} & f_{33} \end{bmatrix}$$

In this case either $f_{32} = 0$ or $f_{32} = \alpha \neq 0$ (constant).

Case (2.1): If $f_{32} = 0$, then the matrix of ∂ is given by:

$$\partial = \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ C & f_{22} & f_{23} \\ 0 & 0 & 0 \end{bmatrix}$$

By Lemma 7.1.5, replace $row(1)$ by $row(1) - (\frac{f_{11}}{C})row(2)$ and at the same time replace $column(2)$ by $column(2) - (\frac{f_{11}}{C})column(1)$ to get:

$$\partial = \begin{bmatrix} 0 & 0 & \frac{C f_{13} - f_{11} f_{23}}{C} \\ C & 0 & f_{23} \\ 0 & 0 & f_{33} \end{bmatrix}$$

By Lemma 7.1.5, replace $row(1)$ by $row(2)$ and at the time replace $column(2)$ by $column(1)$ to get:

$$\partial = \begin{bmatrix} 0 & C & f_{23} \\ 0 & 0 & \frac{Cf_{13}-f_{11}f_{23}}{C} \\ 0 & 0 & f_{33} \end{bmatrix}$$

Therefore, M is solvable as before (Case 1.1).

Case (2.2): $f_{32} = \alpha \neq 0$ (constant),then the matrix of ∂ is given by:

$$\partial = \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ C & f_{22} & f_{23} \\ 0 & \alpha & f_{33} \end{bmatrix}$$

Since, $\partial^2 = 0$, multiply $row(3)$ by $column(1)$, we will get that $\alpha C = 0$, but this is a contradiction because $C \neq 0$ and $\alpha \neq 0$.

Therefore, this case is not possible. □

Proposition 7.2.3. *Let K be a field and let $R = K[x_1, x_2, \ldots, x_n]$ be a graded ring of polynomials graded in the negative way. Let M be a free finitely generated differential graded R-module with basis $S = \{e_i\}_{i=1}^4$ and with differential ∂ of degree $(P \leq -2)$. Suppose that, $dim(e_i) = k_i$ such that $1 \leq i \leq 4$, and $k_i > k_{i+1}$. If $k_i - k_{i+1} = t_i$ such that $t_i \geq -P$, then M is solvable.*

Proof. M is graded as follows:

$$\begin{array}{l} 0 \\ \downarrow \\ e_1 \in M_{k_1} = K \cdot e_1 \oplus 0 \cdot e_2 \oplus 0 \cdot e_3 \oplus 0 \cdot e_4. \\ \downarrow \\ \vdots \\ \downarrow \\ e_2 \in M_{k_2} = R_{k_2-k_1} \cdot e_1 \oplus k.e_2 \oplus 0 \cdot e_3 \oplus 0 \cdot e_4. \\ \downarrow \\ \vdots \\ \downarrow \\ e_3 \in M_{k_3} = R_{k_3-k_1} \cdot e_1 \oplus R_{k_3-k_2} \cdot e_2 \oplus K \cdot e_3 \oplus 0 \cdot e_4. \\ \downarrow \\ \vdots \\ \downarrow \end{array}$$

$$
\begin{array}{l}
e_4 \in M_{k_4} = R_{k_4-k_1} \cdot e_1 \oplus R_{k_4-k_2} \cdot e_2 \oplus R_{k_4-k_3} \cdot e_3 \oplus K \cdot e_4. \\
\qquad \downarrow \\
\qquad \vdots \\
\qquad \downarrow \\
e_j \in M_{k_j} = R_{k_j-k_1} \cdot e_1 \oplus R_{k_j-k_2} \cdot e_2 \oplus R_{k_j-k_3} \cdot e_3 \oplus R_{k_j-k_4} \cdot e_4 \oplus \ldots \oplus K.e_j. \\
\qquad \downarrow \\
\qquad \vdots
\end{array}
$$

Suppose that,

$$
\begin{aligned}
\partial(e_1) &= f_{11}e_1 + f_{21}e_2 + f_{31}e_3 + f_{41}e_4 \\
\partial(e_2) &= f_{12}e_1 + f_{22}e_2 + f_{32}e_3 + f_{42}e_4 \\
\partial(e_3) &= f_{13}e_1 + f_{23}e_2 + f_{33}e_3 + f_{43}e_4 \\
\partial(e_4) &= f_{14}e_1 + f_{24}e_2 + f_{34}e_3 + f_{44}e_4
\end{aligned}
$$

Then the matrix of ∂ with respect to the basis $\{e_1\}_{i=1}^{4}$ is given by:

$$
\partial = \begin{bmatrix} f_{11} & f_{12} & f_{13} & f_{14} \\ f_{21} & f_{22} & f_{23} & f_{24} \\ f_{31} & f_{32} & f_{33} & f_{34} \\ f_{41} & f_{42} & f_{43} & f_{44} \end{bmatrix}
$$

Now,

$$dim(\partial(e_1)) = dim(f_{11}) + dim(e_1),$$

$$k_1 + P = dim(f_{11}) + k_1 \text{ implies that,}$$

$$dim(f_{11}) = P, \text{ and thus } deg(f_{11}) = -P.$$

So,

$$dim(\partial(e_1)) = dim(f_{21}) + dim(e_2),$$

$$k_1 + P = dim(f_{21}) + k_2 \text{ implies that,}$$

$$dim(f_{21}) = P + k_1 - k_2 = P + t_1 \geq P - P = 0, \text{ and thus}$$

$$f_{21} = 0 \; or \; f_{21} = C_1 \neq 0 \text{ (constant).}$$

Also,

$$dim(\partial(e_1)) = dim(f_{31}) + dim(e_3),$$

$$k_1 + P = dim(f_{31}) + k_3, \text{ implies that}$$

$$dim(f_{31}) = k_1 - k_3 + P = -2P + P = -P \geq 2 \text{ and thus } f_{31} = 0, \text{ similarly } f_{41} = 0$$

Also,

$$dim(\partial(e_2)) = dim(f_{12}) + dim(e_1),$$

$$k_2 + P = dim(f_{12}) + k_1 \text{ implies that,}$$

$$dim(f_{12}) = k_2 - k_1 + P < 0 \text{ and thus } deg(f_{12}) = -(k_2 - k_1 + P).$$

So,

$$dim(\partial(e_2)) = dim(f_{22}) + dim(e_2),$$

$$k_2 + P = dim(f_{22}) + k_2 \text{ implies that,}$$

$$dim(f_{22}) = P + k_2 - k_2 = P, \text{ and thus } deg(f_{22}) = -P.$$

So,

$$dim(\partial(e_2)) = dim(f_{32}) + dim(e_3),$$

$$k_2 + P = dim(f_{32}) + k_3 \text{ implies that,}$$

$$dim(f_{32}) = k_2 - k_3 + P \geq -P + P = 0, \text{ and thus}$$

$$f_{32} = 0 \ or \ f_{32} = C_2 \neq 0 \text{ (constant).}$$

So,

$$dim(\partial(e_2)) = dim(f_{42}) + dim(e_4),$$

$$k_2 + P = dim(f_{42}) + k_4, \text{ implies that}$$

$$dim(f_{42}) = k_2 - k_4 + P > 0, \text{ and thus } f_{42} = 0 \ or \ f_{43} = C_3 \neq 0 \text{ (constant).}$$

Also,

$$dim(\partial(e_3)) = dim(f_{13}) + dim(e_1),$$

$$k_3 + P = dim(f_{13}) + k_1 \text{ implies that,}$$

$$dim(f_{13}) = k_3 - k_1 + P < 0 \text{ and thus } deg(f_{13}) = -(k_3 - k_1 + P).$$

So,

$$dim(\partial(e_3)) = dim(f_{23}) + dim(e_2),$$

$$k_3 + P = dim(f_{23}) + k_2, \text{ implies that}$$

$$dim(f_{23}) = P + k_3 - k_2 < 0 \text{ and thus } deg(f_{23}) = -(k_3 - k_2 + P).$$

So,

$$dim(\partial(e_3)) = dim(f_{33}) + dim(e_3),$$

$$k_3 + P = dim(f_{33}) + k_3, \text{ implies that}$$

$$dim(f_{33}) = P, \text{ and thus } degree\ f_{33} = -P.$$

So,

$$dim(\partial(e_3)) = dim(f_{43}) + dim(e_4),$$

$$k_3 + P = dim(f_{43}) + k_4, \text{ implies that}$$

$$dim(f_{43}) = k_3 - k_4 + P \geq 0, \text{ and thus } f_{43} = 0.$$

Also,

$$dim(\partial(e_4)) = dim(f_{14}) + dim(e_1),$$

$$k_4 + P = dim(f_{14}) + k_1, \text{ implies that}$$

$$dim(f_{14}) = k_4 - k_1 + P \text{ and thus } deg(f_{14}) = -(k_4 - k_1 + P).$$

Similarly, $degree\ f_{24} = -(P+k_4-k_2)$, $deg(f_{34}) = -(P+k_4-k_3)$, and $deg(f_{44}) = -P$.

Hence, the matrix of ∂ is given by

$$\partial = \begin{bmatrix} f_{11} & f_{12} & f_{13} & f_{14} \\ f_{21} & f_{22} & f_{23} & f_{24} \\ 0 & f_{32} & f_{33} & f_{34} \\ 0 & 0 & f_{43} & f_{44} \end{bmatrix}$$

where,

1. $f_{21} = 0$ or $f_{21} = \beta_1 \neq 0$ (constant),
2. $f_{32} = 0$ or $f_{32} = \beta_2 \neq 0$ (constant),
3. $f_{43} = 0$ or $f_{43} = \beta_3 \neq 0$ (constant).

Case (1): If $f_{21} = 0$, then the matrix ∂ is given by

$$\partial = \begin{bmatrix} f_{11} & f_{12} & f_{13} & f_{14} \\ 0 & f_{22} & f_{23} & f_{24} \\ 0 & f_{32} & f_{33} & f_{34} \\ 0 & 0 & f_{43} & f_{44} \end{bmatrix}$$

Since $\partial^2 = 0$, this implies $f_{11}^2 = 0$ which implies $f_{11} = 0$.

Thus,

$$\partial = \begin{bmatrix} 0 & f_{12} & f_{13} & f_{14} \\ 0 & f_{22} & f_{23} & f_{24} \\ 0 & f_{32} & f_{33} & f_{34} \\ 0 & 0 & f_{43} & f_{44} \end{bmatrix}$$

In this case either $f_{32} = 0$ or $f_{32} = \beta_2 \neq 0$ (constant).

Case (1.1): If $f_{32} = 0$, then the matrix ∂ is given by

$$\partial = \begin{bmatrix} 0 & f_{12} & f_{13} & f_{14} \\ 0 & f_{22} & f_{23} & f_{24} \\ 0 & 0 & f_{33} & f_{34} \\ 0 & 0 & f_{43} & f_{44} \end{bmatrix}$$

since $\partial^2 = 0$, implies that, $f_{22}^2 = 0$ which implies $f_{22} = 0$.

Thus,the matrix of ∂ is given by

$$\partial = \begin{bmatrix} 0 & f_{12} & f_{13} & f_{14} \\ 0 & 0 & f_{23} & f_{24} \\ 0 & 0 & f_{33} & f_{34} \\ 0 & 0 & f_{43} & f_{44} \end{bmatrix}$$

In this case either $f_{43} = 0$ or $f_{43} = \beta_3 \neq 0$ (constant).

Case (1.1.a): If $f_{43} = 0$, then the matrix ∂ is given by

$$\partial = \begin{bmatrix} 0 & f_{12} & f_{13} & f_{14} \\ 0 & 0 & f_{23} & f_{24} \\ 0 & 0 & f_{33} & f_{34} \\ 0 & 0 & 0 & f_{44} \end{bmatrix}$$

Since $\partial^2 = 0$, implies that, $f_{33}^2 = f_{44}^2 = 0$ which implies $f_{33} = f_{44} = 0$.

Thus,

$$\partial = \begin{bmatrix} 0 & f_{12} & f_{13} & f_{14} \\ 0 & 0 & f_{23} & f_{24} \\ 0 & 0 & 0 & f_{34} \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

To show, M has a composition series:

Let $C_0 = 0$, $C_1 = \langle e_1 \rangle$, $C_2 = \langle e_1, e_2 \rangle$, $C_3 = \langle e_1, e_2, e_3 \rangle$ and $C_4 = \langle e_1, e_2, e_3, e_4 \rangle$.

Then C_j/C_{j-1} is free, for all $1 \leq j \leq 4$.

If $x \in C_4$, then $x = \alpha_1 e_1 + \alpha_2 e_2 + \alpha_3 e_3 + \alpha_4 e_4$.

So, $\partial(x) = \alpha_1 \partial(e_1) + \alpha_2 \partial(e_2) + \alpha_3 \partial(e_3) + \alpha_4 \partial(e_4)$,

$\partial(x) = \alpha_1(0) + \alpha_2(f_{12}e_1) + \alpha_3(f_{13}e_1 + f_{23}e_2) + \alpha_4(f_{14}e_1 + f_{24}e_2 + f_{34}e_3) \ \in C_3$.

Hence, $\partial(C_4) \ \subseteq \ C_3$, and then $\partial(C_4/C_3) = 0$.

Also, if $x \in C_3$, then $x = \alpha_1 e_1 + \alpha_2 e_2 + \alpha_3 e_3$,

So, $\partial(x) = \alpha_1 \partial(e_1) + \alpha_2 \partial(e_2) + \alpha_3 \partial(e_3)$,

$\partial(x) = \alpha_1(0) + \alpha_2(f_{12}e_1) + \alpha_3(f_{13}e_1 + f_{23}e_2) \ \in C_2$.

Hence, $\partial(C_3) \ \subseteq \ C_2$, and then $\partial(C_3/C_2) = 0$.

Also, if $x \in C_2$, then $x = \alpha_1 e_1 + \alpha_2 e_2$

So, $\partial(x) = \alpha_1 \partial(e_1) + \alpha_2 \partial(e_2)$

$\partial(x) = \alpha_1(0) + \alpha_2(f_{12}e_1) \ \in C_1$.

Hence, $\partial(C_2) \ \subseteq \ C_1$, and then $\partial(C_2/C_1) = 0$.

Finally, if $x \in C_1$, then $x = \alpha_1 e_1$ and so,

$\partial(x) = \alpha_1 \partial(e_1) = \alpha_1(0) = 0 \in C_0$.

Hence, $\partial(C_1) \ \subseteq \ C_0$, and then $\partial(C_1/C_0) = 0$.

Therefore, $0 = C_0 \subseteq C_1 \subseteq C_2 \subseteq C_3 \subseteq C_4 = M$ is a composition series for M.

Thus, M is solvable.

Case (1.1.b): If $f_{43} = \beta_3 \neq 0$ (constant) , then the matrix ∂ is given by

$$\partial = \begin{bmatrix} 0 & f_{12} & f_{13} & f_{14} \\ 0 & 0 & f_{23} & f_{24} \\ 0 & 0 & f_{33} & f_{34} \\ 0 & 0 & \beta_3 & f_{44} \end{bmatrix}$$

Since $\partial^2 = 0$, implies that, $f_{33}^2 + \beta_3 f_{34} = 0$ and $\beta_3 f_{34} + f_{44}^2 = 0$, which implies $f_{33} = f_{44}$ and $\beta_3 f_{34} = f_{44}^2$.

Now, by Lemma 7.1.5, replace $row(3)$ by $row(3) - (\frac{f_{33}}{\beta_3})row(4)$ and at the same time replace column (4) by $column(4) - (\frac{f_{33}}{\beta_3})column(3)$ to get:

$$\partial = \begin{bmatrix} 0 & f_{12} & f_{13} & \frac{\beta_3 f_{14} - f_{33} f_{13}}{\beta_3} \\ 0 & 0 & f_{23} & \frac{\beta_3 f_{24} - f_{33} f_{23}}{\beta_3} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \beta_3 & 0 \end{bmatrix}$$

By Lemma 7.1.5, replace $row(3)$ by $row(4)$ and at the time replace $column(4)$ by $column(3)$ to get:

$$\partial = \begin{bmatrix} 0 & f_{12} & \frac{\beta_3 f_{14} - f_{33} f_{13}}{\beta_3} & f_{13} \\ 0 & 0 & \frac{\beta_3 f_{24} - f_{33} f_{23}}{\beta_3} & f_{23} \\ 0 & 0 & 0 & \beta_3 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Therefore, M is solvable as before (Case 1.1.a).

Case (1.2): If $f_{32} = \beta_2 \neq 0$ (constant) , then the matrix ∂ is given by

$$\partial = \begin{bmatrix} 0 & f_{12} & f_{13} & f_{14} \\ 0 & f_{22} & f_{23} & f_{24} \\ 0 & \beta_2 & f_{33} & f_{34} \\ 0 & 0 & f_{43} & f_{44} \end{bmatrix}$$

In this case either $f_{43} = 0$ or $f_{43} = \beta_3 \neq 0$ (constant).

Case (1.2.a): If $f_{43} = 0$, then the matrix ∂ is given by:

$$\partial = \begin{bmatrix} 0 & f_{12} & f_{13} & f_{14} \\ 0 & f_{22} & f_{23} & f_{24} \\ 0 & \beta_2 & f_{33} & f_{34} \\ 0 & 0 & 0 & f_{44} \end{bmatrix}$$

Since $\partial^2 = 0$, implies that, $f_{44}^2 = 0$, which implies $f_{44} = 0$.

Thus,

$$\partial = \begin{bmatrix} 0 & f_{12} & f_{13} & f_{14} \\ 0 & f_{22} & f_{23} & f_{24} \\ 0 & \beta_2 & f_{33} & f_{34} \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Since $\partial^2 = 0$, implies that, $f_{22}^2 + \beta_2 f_{23} = 0$ and $\beta_2 f_{23} + f_{33}^2 = 0$.

Hence, $f_{22} = f_{33} = 0$ and $\beta_2 f_{23} = f_{22}^2$.

By Lemma 7.1.5, replace $row(2)$ by $row(2) - (\frac{f_{22}}{\beta_2})row(3)$ and at the same time replace $column(3)$ by $column(3) - (\frac{f_{22}}{\beta_2})column(2)$ to get:

$$\partial = \begin{bmatrix} 0 & f_{12} & \frac{\beta_2 f_{13} - f_{22} f_{12}}{\beta_2} & f_{14} \\ 0 & 0 & 0 & \frac{\beta_2 f_{24} - f_{22} f_{34}}{\beta_2} \\ 0 & \beta_2 & 0 & f_{34} \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

By Lemma 7.1.5, replace $row(2)$ by $row(3)$ and at the time replace $column(2)$ by $column(3)$ to get:

$$\partial = \begin{bmatrix} 0 & \frac{\beta_2 f_{13} - f_{22} f_{12}}{\beta_2} & f_{12} & f_{14} \\ 0 & 0 & \beta_2 & f_{34} \\ 0 & 0 & 0 & \frac{\beta_2 f_{24} - f_{22} f_{34}}{\beta_2} \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Therefore, M is solvable as before (case 1.1.a).

Case (1.2.b): If $f_{43} = \beta_3 \neq 0$ (constant), then the matrix ∂ is given by:

$$\partial = \begin{bmatrix} 0 & f_{12} & f_{13} & f_{14} \\ 0 & f_{22} & f_{23} & f_{24} \\ 0 & \beta_2 & f_{33} & f_{34} \\ 0 & 0 & \beta_3 & f_{44} \end{bmatrix}$$

Since $\partial^2 = 0$, implies that, $\beta_2 \cdot \beta_3 = 0$, but $\beta_2 \neq 0$ and $\beta_3 \neq 0$ which implies to contradiction. Thus, this case is not possible.

Case (2): If $f_{21} = \beta_1 \neq 0$, then the matrix ∂ is given by

$$\partial = \begin{bmatrix} f_{11} & f_{12} & f_{13} & f_{14} \\ \beta_1 & f_{22} & f_{23} & f_{24} \\ 0 & f_{32} & f_{33} & f_{34} \\ 0 & 0 & f_{43} & f_{44} \end{bmatrix}$$

In this case either $f_{32} = 0$ or $f_{32} = \beta_2 \neq 0$ (constant).

Case (2.1): If $f_{32} = 0$, then the matrix ∂ is given by

$$\partial = \begin{bmatrix} f_{11} & f_{12} & f_{13} & f_{14} \\ \beta_1 & f_{22} & f_{23} & f_{24} \\ 0 & 0 & f_{33} & f_{34} \\ 0 & 0 & f_{43} & f_{44} \end{bmatrix}$$

In this case either $f_{43} = 0$ or $f_{43} = \beta_3 \neq 0$ (constant).

Case (2.1.a): If $f_{43} = 0$, then the matrix ∂ is given by

$$\partial = \begin{bmatrix} f_{11} & f_{12} & f_{13} & f_{14} \\ \beta_1 & f_{22} & f_{23} & f_{24} \\ 0 & 0 & f_{33} & f_{34} \\ 0 & 0 & 0 & f_{44} \end{bmatrix}$$

Since $\partial^2 = 0$, implies that, $f_{33}^2 = f_{44}^2 = 0$, which implies $f_{33} = f_{44} = 0$.

Thus,

$$\partial = \begin{bmatrix} f_{11} & f_{12} & f_{13} & f_{14} \\ \beta_1 & f_{22} & f_{23} & f_{24} \\ 0 & 0 & 0 & f_{34} \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Since $\partial^2 = 0$, implies that, $f_{11}^2 + \beta_1 f_{12} = 0$ and $\beta_1 f_{12} + f_{22}^2 = 0$. Hence, $f_{11} = f_{22} = 0$. and $\beta_1 f_{12} = f_{11}^2$.

By Lemma 7.1.5, replace $row(1)$ by $row(1) - (\frac{f_{11}}{\beta_1})row(2)$ and at the same time replace $column(2)$ by $column(2) - (\frac{f_{11}}{\beta_1})column(1)$ to get:

$$\partial = \begin{bmatrix} 0 & 0 & \frac{\beta_1 f_{13} - f_{11} f_{23}}{\beta_1} & \frac{\beta_1 f_{14} - f_{11} f_{24}}{\beta_1} \\ \beta_1 & 0 & f_{23} & f_{24} \\ 0 & 0 & 0 & f_{34} \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

By Lemma 7.1.5, replace $row(1)$ by $row(2)$ and at the time replace $column(1)$ by $column(2)$ to get:

$$\partial = \begin{bmatrix} 0 & \beta_1 & f_{23} & f_{24} \\ 0 & 0 & \frac{\beta_1 f_{13} - f_{11} f_{23}}{\beta_1} & \frac{\beta_1 f_{14} - f_{11} f_{24}}{\beta_1} \\ 0 & 0 & 0 & f_{34} \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Therefore, M is solvable as before (Case 1.1.a).

Case (2.1.b): If $f_{43} = \beta_3 \neq 0$ (constant).

$$\partial = \begin{bmatrix} f_{11} & f_{12} & f_{13} & f_{14} \\ \beta_1 & f_{22} & f_{23} & f_{24} \\ 0 & 0 & f_{33} & f_{34} \\ 0 & 0 & \beta_3 & f_{44} \end{bmatrix}$$

Since $\partial^2 = 0$, implies that, $f_{33}^2 + \beta_3 f_{34} = 0$, and $\beta_3 f_{34} + f_{44}^2 = 0$. Hence, $f_{33} = f_{44}$ *and* $f_{33}^2 = \beta_3 f_{34}$.

By Lemma 7.1.5, replace $row(3)$ by $row(3) - (\frac{f_{33}}{\beta_3})row(4)$ and at the same time replace $column(4)$ by $column(4) - (\frac{f_{33}}{\beta_3})column(3)$ to get:

$$\partial = \begin{bmatrix} f_{11} & f_{12} & f_{13} & \frac{\beta_3 f_{14} - f_{33} f_{13}}{\beta_3} \\ \beta_1 & f_{22} & f_{23} & \frac{\beta_3 f_{24} - f_{33} f_{23}}{\beta_3} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \beta_3 & 0 \end{bmatrix}$$

By Lemma 7.1.5, replace $row(3)$ by $row(4)$ and at the time replace $column(3)$ by $column(4)$ to get:

$$\partial = \begin{bmatrix} f_{11} & f_{12} & \frac{\beta_3 f_{14} - f_{33} f_{13}}{\beta_3} & f_{13} \\ \beta_1 & f_{22} & \frac{\beta_3 f_{24} - f_{33} f_{23}}{\beta_3} & f_{23} \\ 0 & 0 & 0 & \beta_3 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Since $\partial^2 = 0$, implies that, $f_{11}^2 + \beta_1 f_{12} = 0$, and $\beta_1 f_{12} + f_{22}^2 = 0$. Hence, $f_{11} = f_{22}$ and $f_{11}^2 = \beta_1 f_{12}$.

By Lemma 7.1.5, replace $row(1)$ by $row(1) - (\frac{f_{11}}{\beta_1})row(2)$ and at the same time replace $column(2)$ by $column(2) - (\frac{f_{11}}{\beta_1})column(1)$ to get:

$$\partial = \begin{bmatrix} 0 & 0 & \frac{\beta_1 f_{13} - f_{11} f_{23}}{\beta_1} & \frac{\beta_1[\beta_3 f_{14} - f_{33} f_{13}] - \beta_3 f_{11}[\beta_3 f_{24} - f_{33} f_{23}]}{\beta_1 \beta_3} \\ \beta_1 & 0 & f_{23} & \frac{\beta_3 f_{24} - f_{33} f_{23}}{\beta_3} \\ 0 & 0 & 0 & \beta_3 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

By Lemma 7.1.5, replace $row(1)$ by $row(2)$ and at the time replace $column(1)$ by $column(2)$ to get:

$$\partial = \begin{bmatrix} 0 & \beta_1 & f_{23} & \frac{\beta_3 f_{24} - f_{33} f_{23}}{\beta_3} \\ 0 & 0 & \frac{\beta_1 f_{13} - f_{11} f_{23}}{\beta_1} & \frac{\beta_1[\beta_3 f_{14} - f_{33} f_{13}] - \beta_3 f_{11}[\beta_3 f_{24} - f_{33} f_{23}]}{\beta_1 \beta_3} \\ 0 & 0 & 0 & \beta_3 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Therefore, M is solvable as before (Case 1.1.a).

Case (2.2): If $f_{32} = \beta_2 \neq 0$ (constant), then the matrix ∂ is given by

$$\partial = \begin{bmatrix} f_{11} & f_{12} & f_{13} & f_{14} \\ \beta_1 & f_{22} & f_{23} & f_{24} \\ 0 & \beta_2 & f_{33} & f_{34} \\ 0 & 0 & f_{43} & f_{44} \end{bmatrix}$$

Since $\partial^2 = 0$, then we multiply $row(3)$ by $column(1)$ to get that $\beta_1 \beta_2 = 0$, but $\beta_1 \neq 0$ and $\beta_2 \neq 0$ which implies to contradiction. Thus, this case is not possible.

□

From the previous we conclude the following two propositions:

Proposition 7.2.4. *Let K be a field and let $R = K[x_1, x_2, \ldots, x_n]$ be a graded ring of polynomials graded in the negative way. Let M be a free finitely generated differential graded R-module with basis $S = \{e_i\}_{i=1}^{m}$, and differential ∂ of degree ($P \leq -2$). Suppose, $dim(e_i) = k_i$ such that $1 \leq i \leq m$ and $k_i > k_{i+1}$. If $k_i - k_{i+1} = t_i$ such that $t_i > -P$, then M is solvable.*

Proof. M is graded as follows:

$$\begin{array}{l} 0 \\ \downarrow \\ e_1 \in M_{k_1} = K \cdot e_1 \oplus 0 \cdot e_2 \oplus 0 \cdot e_3 \oplus \ldots \oplus 0.e_m. \\ \downarrow \\ \vdots \\ \downarrow \\ e_2 \in M_{k_2} = R_{k_2 - k_1} \cdot e_1 \oplus K.e_2 \oplus 0.e_3 \oplus 0.e_4 \oplus \ldots \oplus 0.e_m. \\ \downarrow \\ \vdots \\ \downarrow \\ e_3 \in M_{k_3} = R_{k_3 - k_1} \cdot e_1 \oplus R_{k_3 - k_2} \cdot e_2 \oplus K.e_3 \oplus 0.e_4 \oplus \ldots \oplus 0.e_m. \end{array}$$

$$\begin{array}{l}
\qquad\downarrow \\
\qquad\vdots \\
\qquad\downarrow \\
e_4 \in M_{k_4} = R_{k_4-k_1} \cdot e_1 \oplus R_{k_4-k_2} \cdot e_2 \oplus R_{k_4-k_3} \cdot e_3 \oplus K.e_4 \oplus 0.e_5 \oplus \ldots \oplus 0.e_m. \\
\qquad\downarrow \\
\qquad\vdots \\
\qquad\downarrow \\
e_j \in M_{k_j} = R_{k_j-k_1} \cdot e_1 \oplus R_{k_j-k_2} \cdot e_2 \oplus R_{k_j-k_3} \oplus \ldots \oplus K.e_j. \\
\qquad\downarrow \\
\qquad\vdots
\end{array}$$

Suppose that,

$$\begin{aligned}
\partial(e_1) &= f_{11}e_1 + \ldots + f_{m1}e_m, \\
\partial(e_2) &= f_{12}e_1 + \ldots + f_{m2}e_m, \\
&\vdots \\
\partial(e_m) &= f_{1m}e_1 + \ldots + f_{mm}e_m.
\end{aligned}$$

Then the matrix of ∂ with respect to the basis $\{e_1\}_{i=1}^{m}$ is given by:

$$\partial = \begin{bmatrix} f_{11} & f_{12} & \ldots & f_{1m} \\ f_{21} & f_{22} & \ldots & f_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ f_{m1} & f_{m2} & \ldots & f_{mm} \end{bmatrix}$$

Now,

$$dim(\partial(e_1)) = dim(f_{11}) + dim(e_1),$$

$$k_1 + P = dim(f_{11}) + k_1, \text{ implies that } deg(f_{11}) = -P.$$

So,

$$dim(\partial(e_1)) = dim(f_{i1}) + dim(e_i) \text{ for each } 2 \leq i \leq m.$$

So,

$$k_1 + P = dim(f_{i1}) + k_i \text{ and then}$$

$$dim(f_{i1}) = (k_1 - k_i) + P > 0, \text{ i.e., } f_{i1} \in R_{k_1-k_i+P} = 0.$$

Therefore,

$$f_{i1} = 0 \text{ for each } 2 \leq i \leq m.$$

Also,

$$dim(\partial(e_2)) = dim(f_{12}) + dim(e_1),$$

$$k_2 + P = dim(f_{12}) + k_1,$$

$$dim(f_{12}) = k_2 - k_1 + P < 0 \text{ implies that,}$$

$$degree f_{12} = -(P + k_2 - k_1).$$

So,

$$dim(\partial(e_2)) = dim(f_{22}) + dim(e_2),$$

$$k_2 + P = dim(f_{22}) + k_2, \text{ implies that } deg(f_{22}) = -P.$$

So,

$$dim(\partial(e_2)) = dim(f_{i2}) + dim(e_i) \ for \ each \ 3 \leq i \leq m,$$

$$k_2 + P = dim(f_{i2}) + k_i \text{ and then}$$

$$dim(f_{i2}) = P + (k_2 - k_i) > 0, \text{ i.e., } f_{i2} \in R_{P+k_2-k_i} = 0.$$

Therefore,

$$f_{i2} = 0 \text{ for each } 3 \leq i \leq m.$$

Now,

$$dim(\partial(e_{m-1})) = dim(f_{i(m-1)}) + dim(e_i) \text{ for each } 1 \leq i \leq m-1,$$

$$k_{m-1} + P = dim(f_{i(m-1))} + k_i \text{ and then}$$

$$dim(f_{i(m-1)}) = (P + k_{m-1} - k_i) < 0, \text{ i.e., } f_{i(m-1)} \in R_{P+k_{m-1}-k_i} \neq 0.$$

Therefore,

$$f_{i(m-1)} \neq 0 \text{ for each } 1 \leq i \leq m-1,$$

and,

$$dim(\partial(e_{m-1})) = dim(f_{m(m-1)}) + dim(e_m),$$

$$k_{m-1} + P = dim(f_{m(m-1)} + k_m, \text{ implies that}$$

$$dim(f_{m(m-1))} = P + k_{m-1} - k_m \geq 0 \text{ which implies that } f_{m(m-1)} = 0.$$

Also,

$$dim(\partial(e_m)) = dim(f_{im}) + dim(e_i) \text{ for each } 1 \leq i \leq m,$$

$$k_m + P = dim(f_{im}) + k_i \text{ and then}$$

$$dim(f_{i(m)}) = P + (k_m - k_i) < 0, \text{ i.e., } f_{im} \in R_{P+k_m-k_i} \neq 0.$$

Therefore,

$$f_{im} \neq 0 \text{ for each } 1 \leq i \leq m.$$

Hence, the matrix of ∂ is given by:

$$\partial = \begin{bmatrix} f_{11} & f_{12} & f_{13} & f_{14} & \dots & f_{1(m-1)} & f_{1m} \\ 0 & f_{22} & f_{23} & f_{24} & \dots & f_{2(m-1)} & f_{2m} \\ 0 & 0 & f_{33} & f_{34} & \dots & f_{3(m-1)} & f_{3m} \\ 0 & 0 & 0 & f_{44} & \dots & f_{4(m-1)} & f_{4m} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & f_{(m-1)(m-1)} & f_{(m-1)m} \\ 0 & 0 & 0 & 0 & \dots & 0 & f_{mm} \end{bmatrix}$$

Since, $\partial^2 = 0$ and R is an integral domain then we have that $f_{ii} = 0$, for each $1 \leq i \leq m$.

Thus, ∂ is given by :

$$\partial = \begin{bmatrix} 0 & f_{12} & f_{13} & f_{14} & \dots & f_{1(m-1)} & f_{1m} \\ 0 & 0 & f_{23} & f_{24} & \dots & f_{2(m-1)} & f_{2m} \\ 0 & 0 & 0 & f_{34} & \dots & f_{3(m-1)} & f_{3m} \\ 0 & 0 & 0 & 0 & \dots & f_{4(m-1)} & f_{4m} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 0 & f_{(m-1)m} \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 \end{bmatrix}$$

To show, M has a composition series:
Let $C_0 = 0$ and $C_j = \langle e_1, e_2, \dots, e_j \rangle$, for all $1 \leq j \leq m$.
Then (C_j/C_{j-1}) is free. If $x \in C_j$, then x can be written uniquely as:

$$x = \alpha_1 e_1 + \alpha_2 e_2 + \dots + \alpha_j e_j.$$

Thus,

$$\partial(x) = \alpha_1 \partial(e_1) + \alpha_2 \partial(e_2) + \dots + \alpha_j \partial(e_j)$$

$$\partial(x) = \alpha_1(0) + \alpha_2(f_{12}e_1) + \ldots + \alpha_j(f_{1j}e_1 + \ldots + f_{(j-1)j}e_{j-1}) \ \in C_{j-1}$$

Therefore,

$$\partial(C_j/C_{j-1}) = 0, \text{ for each } 1 \leq j \leq m.$$

Hence, $0 = C_0 \subseteq C_1 \subseteq C_2 \subseteq \ldots \subseteq C_m = M$ is a composition series for M.

Thus, M is solvable. □

Proposition 7.2.5. *Let K be a field and let $R = K[x_1, x_2, \ldots, x_n]$ be a graded ring of polynomials graded in the negative way. Let M be a free finitely generated differential graded R-module with basis $S = \{e_i\}_{i=1}^m$, and with differential ∂ of degree $(p \leq -2)$. Suppose, $dim(e_i) = k_i$ such that $1 \leq i \leq m$ and $k_i > k_{i+1}$. If $k_i - k_{i+1} = t_i$ such that $t_i \geq -p$, then M is solvable.*

Proof. M is graded as in (Proposition 7.2.4):

Suppose that,

$$\begin{aligned}
\partial(e_1) &= f_{11}e_1 + \ldots + f_{m1}e_m, \\
\partial(e_2) &= f_{12}e_1 + \ldots + f_{m2}e_m, \\
&\vdots \\
\partial(e_m) &= f_{1m}e_1 + \ldots + f_{mm}e_m.
\end{aligned}$$

Then the matrix of ∂ with respect to the basis $\{e_1\}_{i=1}^m$ is given by:

$$\partial = \begin{bmatrix} f_{11} & f_{12} & \ldots & f_{1m} \\ f_{21} & f_{22} & \ldots & f_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ f_{m1} & f_{m2} & \ldots & f_{mm} \end{bmatrix}$$

Now,

$$dim(\partial(e_1)) = dim(f_{11}) + dim(e_1),$$

$$k_1 + P = dim(f_{11}) + k_1, \text{ implies that }, deg(f_{11}) = -P.$$

Also,

$$dim(\partial(e_1)) = dim(f_{21}) + dim(e_2),$$

$$k_1 + P = dim(f_{21}) + k_2, \text{ implies that } dim(f_{21}) = (k_1 - k_2) + P = t_1 \geq 0,$$

which implise, $f_{21} = 0$ or $f_{21} = C_1 \neq 0$ (constant).

So,

$$dim(\partial(e_1)) = dim(f_{i1}) + dim(e_i) \text{ for each } 3 \leq i \leq m,$$

$$k_1 + P = dim(f_{i1}) + k_i \ \text{ and then}$$

$$dim(f_{i1}) = (k_1 - k_i) + P > 0, \text{ i.e., } f_{i1} \in R_{k_1 - k_i + P} = 0.$$

Therefore,

$$f_{i1} = 0 \text{ for each } 3 \leq i \leq m.$$

Also,

$$dim(\partial(e_2)) = dim(f_{12}) + dim(e_1),$$

$$k_2 + P = dim(f_{12}) + k_1,$$

$$dim(f_{12}) = k_2 - k_1 + P < 0 \text{ implies that, } deg(f_{12}) = -(k_2 - k_1 + P).$$

So,

$$dim(\partial(e_2)) = dim(f_{22}) + dim(e_2),$$

$$k_2 + P = dim(f_{22}) + k_2, \text{ implies that, } deg(f_{22}) = -P.$$

So,

$$dim(\partial(e_2)) = dim(f_{32}) + dim(e_3),$$

$$k_2 + P = dim(f_{32}) + k_3, \text{ implies that, } dim(f_{32}) = P + k_2 - k_1 \geq 0$$

$f_{32} = 0$ *or* $f_{32} = C_2 \neq 0$ (constant).

So,

$$dim(\partial(e_2)) = dim(f_{i2}) + dim(e_i) \text{ for each } 4 \leq i \leq m,$$

$$k_2 + P = dim(f_{i2}) + k_i \text{ and then}$$

$$dim(f_{i2}) = (k_2 - k_i) + P > 0, \text{ i.e., } f_{i2} \in R_{P + k_2 - k_i} = 0.$$

Therefore,

$$f_{i2} = 0 \text{ for each } 4 \leq i \leq m.$$

Now,

$$dim(\partial(e_{m-1})) = dim(f_{i(m-1)}) + dim(e_i) \text{ for each } 1 \leq i \leq m-1,$$

$$k_{m-1} + P = dim(f_{i(m-1))} + k_i \text{ and then}$$

$$dim(f_{i(m-1)}) = P + (k_{m-1} - k_i) < 0, \text{ i.e., } f_{i(m-1)} \in R_{k_{m-1}-k_i+P} \neq 0.$$

Therefore,

$$f_{i(m-1)} \neq 0 \text{ for each } 1 \leq i \leq m-1,$$

and,

$$dim(\partial(e_{m-1})) = dim(f_{m(m-1)}) + dim(e_m),$$

$$k_{m-1} + P = dim(f_{m(m-1)} + k_m, \text{ implies that, } dim(f_{m(m-1)}) = P + k_{m-1} - k_m \geq 0.$$

Thus, $f_{m(m-1)} = 0$ or $f_{m(m-1)} = C_{m-1} \neq 0$ (constant).

Also,

$$dim(\partial(e_m)) = dim(f_{im}) + dim(e_i) \text{ for each } 1 \leq i \leq m,$$

$$k_m + P = dim(f_{im} + k_i \text{ and then}$$

$$dim(f_{i(m)}) = (k_m - k_i) + P < 0, \text{ i.e., } f_{i(m)} \in R_{k_m-k_i+P} \neq 0.$$

Therefore,

$$f_{im} \neq 0 \text{ for each } 1 \leq i \leq m.$$

Hence, the matrix of ∂ is given by:

$$\partial = \begin{bmatrix} f_{11} & f_{12} & f_{13} & f_{14} & \cdots & f_{1(m-1)} & f_{1m} \\ f_{21} & f_{22} & f_{23} & f_{24} & \cdots & f_{2(m-1)} & f_{2m} \\ 0 & f_{32} & f_{33} & f_{34} & \cdots & f_{3(m-1)} & f_{3m} \\ 0 & 0 & f_{43} & f_{44} & \cdots & f_{4(m-1)} & f_{4m} \\ 0 & 0 & 0 & f_{54} & \cdots & f_{5(m-1)} & f_{5m} \\ 0 & 0 & 0 & 0 & \cdots & f_{6(m-1)} & f_{6m} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & f_{(m-2)(m-1)} & f_{(m-2)m} \\ 0 & 0 & 0 & 0 & \cdots & f_{(m-1)(m-1)} & f_{(m-1)m} \\ 0 & 0 & 0 & 0 & \cdots & f_{m(m-1)} & f_{mm} \end{bmatrix}$$

where,

$f_{21} = 0$ *or* $f_{21} = C_1 \neq 0$ (constant).
$f_{32} = 0$ *or* $f_{32} = C_2 \neq 0$ (constant).
$f_{43} = 0$ *or* $f_{43} = C_3 \neq 0$ (constant).

$f_{54} = 0$ *or* $f_{54} = C_4 \neq 0$ (constant).

$\vdots$

$f_{(m-1)(m-2)} = 0$ *or* $f_{(m-1)(m-2)} = C_{m-2} \neq 0$ (constant).

$f_{m(m-1)} = 0$ *or* $f_{m(m-1)} = C_{m-1} \neq 0$ (constant).

Case (1): If $f_{21} = 0$, then the matrix ∂ is given by

$$\partial = \begin{bmatrix} f_{11} & f_{12} & f_{13} & f_{14} & \dots & f_{1(m-1)} & f_{1m} \\ 0 & f_{22} & f_{23} & f_{24} & \dots & f_{2(m-1)} & f_{2m} \\ 0 & f_{32} & f_{33} & f_{34} & \dots & f_{3(m-1)} & f_{3m} \\ 0 & 0 & f_{43} & f_{44} & \dots & f_{4(m-1)} & f_{4m} \\ 0 & 0 & 0 & f_{54} & \dots & f_{5(m-1)} & f_{5m} \\ 0 & 0 & 0 & 0 & \dots & f_{6(m-1)} & f_{6m} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & f_{(m-2)(m-1)} & f_{(m-2)m} \\ 0 & 0 & 0 & 0 & \dots & f_{(m-1)(m-1)} & f_{(m-1)m} \\ 0 & 0 & 0 & 0 & \dots & f_{m(m-1)} & f_{mm} \end{bmatrix}$$

In this case either $f_{32} = 0$ *or* $f_{32} = C_2 \neq 0$ (constant).

Case (1.1): If $f_{32} = 0$, then the matrix ∂ is given by

$$\partial = \begin{bmatrix} f_{11} & f_{12} & f_{13} & f_{14} & \dots & f_{1(m-1)} & f_{1m} \\ 0 & f_{22} & f_{23} & f_{24} & \dots & f_{2(m-1)} & f_{2m} \\ 0 & 0 & f_{33} & f_{34} & \dots & f_{3(m-1)} & f_{3m} \\ 0 & 0 & f_{43} & f_{44} & \dots & f_{4(m-1)} & f_{4m} \\ 0 & 0 & 0 & f_{54} & \dots & f_{5(m-1)} & f_{5m} \\ 0 & 0 & 0 & 0 & \dots & f_{6(m-1)} & f_{6m} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & f_{(m-2)(m-1)} & f_{(m-2)m} \\ 0 & 0 & 0 & 0 & \dots & f_{(m-1)(m-1)} & f_{(m-1)m} \\ 0 & 0 & 0 & 0 & \dots & f_{m(m-1)} & f_{mm} \end{bmatrix}$$

In this case either $f_{43} = 0$ or $f_{43} = C_3 \neq 0$ (constant).

Case (1.1.1): If $f_{43} = 0$, then the matrix ∂ is given by

$$\partial = \begin{bmatrix} f_{11} & f_{12} & f_{13} & f_{14} & \cdots & f_{1(m-1)} & f_{1m} \\ 0 & f_{22} & f_{23} & f_{24} & \cdots & f_{2(m-1)} & f_{2m} \\ 0 & 0 & f_{33} & f_{34} & \cdots & f_{3(m-1)} & f_{3m} \\ 0 & 0 & 0 & f_{44} & \cdots & f_{4(m-1)} & f_{4m} \\ 0 & 0 & 0 & f_{54} & \cdots & f_{5(m-1)} & f_{5m} \\ 0 & 0 & 0 & 0 & \cdots & f_{6(m-1)} & f_{6m} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & f_{(m-2)(m-1)} & f_{(m-2)m} \\ 0 & 0 & 0 & 0 & \cdots & f_{(m-1)(m-1)} & f_{(m-1)m} \\ 0 & 0 & 0 & 0 & \cdots & f_{m(m-1)} & f_{mm} \end{bmatrix}$$

In this case either $f_{54} = 0$ or $f_{54} = C_4 \neq 0$ (constant).

Case (1.1.1.1): If $f_{54} = 0$, then the matrix ∂ is given by

$$\partial = \begin{bmatrix} f_{11} & f_{12} & f_{13} & f_{14} & \cdots & f_{1(m-1)} & f_{1m} \\ 0 & f_{22} & f_{23} & f_{24} & \cdots & f_{2(m-1)} & f_{2m} \\ 0 & 0 & f_{33} & f_{34} & \cdots & f_{3(m-1)} & f_{3m} \\ 0 & 0 & 0 & f_{44} & \cdots & f_{4(m-1)} & f_{4m} \\ 0 & 0 & 0 & 0 & \cdots & f_{5(m-1)} & f_{5m} \\ 0 & 0 & 0 & 0 & \cdots & f_{6(m-1)} & f_{6m} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & f_{(m-2)(m-1)} & f_{(m-2)m} \\ 0 & 0 & 0 & 0 & \cdots & f_{(m-1)(m-1)} & f_{(m-1)m} \\ 0 & 0 & 0 & 0 & \cdots & f_{m(m-1)} & f_{mm} \end{bmatrix}$$

Similarly, we arrived to the following case: either $f_{m(m-1)} = 0$ or

$f_{m(m-1)} = C_{m-1} \neq 0$.

Case (1.1.....1.a): If $f_{m(m-1)} = 0$, then the matrix ∂ is given by

$$\partial = \begin{bmatrix} f_{11} & f_{12} & f_{13} & f_{14} & \dots & f_{1(m-1)} & f_{1m} \\ 0 & f_{22} & f_{23} & f_{24} & \dots & f_{2(m-1)} & f_{2m} \\ 0 & 0 & f_{33} & f_{34} & \dots & f_{3(m-1)} & f_{3m} \\ 0 & 0 & 0 & f_{44} & \dots & f_{4(m-1)} & f_{4m} \\ 0 & 0 & 0 & 0 & \dots & f_{5(m-1)} & f_{5m} \\ 0 & 0 & 0 & 0 & \dots & f_{6(m-1)} & f_{6m} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & f_{(m-2)(m-1)} & f_{(m-2)m} \\ 0 & 0 & 0 & 0 & \dots & f_{(m-1)(m-1)} & f_{(m-1)m} \\ 0 & 0 & 0 & 0 & \dots & 0 & f_{mm} \end{bmatrix}$$

Since $\partial^2 = 0$, this implies $f_{ii}^2 = 0$ which implies $f_{ii} = 0$ for each $1 \leq i \leq m$.

(the reason is that, R is an integral domain).

Thus,

$$\partial = \begin{bmatrix} 0 & f_{12} & f_{13} & f_{14} & \dots & f_{1(m-1)} & f_{1m} \\ 0 & 0 & f_{23} & f_{24} & \dots & f_{2(m-1)} & f_{2m} \\ 0 & 0 & 0 & f_{34} & \dots & f_{3(m-1)} & f_{3m} \\ 0 & 0 & 0 & 0 & \dots & f_{4(m-1)} & f_{4m} \\ 0 & 0 & 0 & 0 & \dots & f_{5(m-1)} & f_{5m} \\ 0 & 0 & 0 & 0 & \dots & f_{6(m-1)} & f_{6m} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & f_{(m-2)(m-1)} & f_{(m-2)m} \\ 0 & 0 & 0 & 0 & \dots & 0 & f_{(m-1)m} \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 \end{bmatrix}$$

Therefore, M is solvable (by the previous proposition).

Case (1.1.. . . .1.b): If $f_{m(m-1)} = C_{m-1} \neq 0$ then the matrix ∂ is given by

$$\partial = \begin{bmatrix} f_{11} & f_{12} & f_{13} & f_{14} & \dots & f_{1(m-1)} & f_{1m} \\ 0 & f_{22} & f_{23} & f_{24} & \dots & f_{2(m-1)} & f_{2m} \\ 0 & 0 & f_{33} & f_{34} & \dots & f_{3(m-1)} & f_{3m} \\ 0 & 0 & 0 & f_{44} & \dots & f_{4(m-1)} & f_{4m} \\ 0 & 0 & 0 & 0 & \dots & f_{5(m-1)} & f_{5m} \\ 0 & 0 & 0 & 0 & \dots & f_{6(m-1)} & f_{6m} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & f_{(m-2)(m-1)} & f_{(m-2)m} \\ 0 & 0 & 0 & 0 & \dots & f_{(m-1)(m-1)} & f_{(m-1)m} \\ 0 & 0 & 0 & 0 & \dots & C_{m-1} & f_{mm} \end{bmatrix}$$

Since $\partial^2 = 0$, this implies that $f^2_{(m-1)(m-1)} + C_{m-1}f_{(m-1)m} = 0$ and $C_{m-1}f_{(m-1)m} + f^2_{mm} = 0$.

Thus, $f_{(m-1)(m-1)} = f_{mm}$ and $C_{m-1}f_{(m-1)m} = f^2_{(m-1)(m-1)}$.

By Lemma 7.1.5, replace $row(m-1)$ by $row(m-1) - (\frac{f_{(m-1)(m-1)}}{C_{m-1}})row(m)$ and at the same time replace column (m) by $column(m) - (\frac{f_{(m-1)(m-1)}}{C_{m-1}})column(m-1)$ to get:

$$\partial = \begin{bmatrix} f_{11} & f_{12} & f_{13} & f_{14} & \dots & f_{1(m-1)} & \frac{C_{m-1}f_{1m} - f_{(m-1)(m-1)}f_{1(m-1)}}{C_{m-1}} \\ 0 & f_{22} & f_{23} & f_{24} & \dots & f_{2(m-1)} & \frac{C_{m-1}f_{2m} - f_{(m-1)(m-1)}f_{2(m-1)}}{C_{m-1}} \\ 0 & 0 & f_{33} & f_{34} & \dots & f_{3(m-1)} & \frac{C_{m-1}f_{3m} - f_{(m-1)(m-1)}f_{3(m-1)}}{C_{m-1}} \\ 0 & 0 & 0 & f_{44} & \dots & f_{4(m-1)} & \frac{C_{m-1}f_{4m} - f_{(m-1)(m-1)}f_{4(m-1)}}{C_{m-1}} \\ 0 & 0 & 0 & 0 & \dots & f_{5(m-1)} & \frac{C_{m-1}f_{5m} - f_{(m-1)(m-1)}f_{5(m-1)}}{C_{m-1}} \\ 0 & 0 & 0 & 0 & \dots & f_{6(m-1)} & \frac{C_{m-1}f_{6m} - f_{(m-1)(m-1)}f_{6(m-1)}}{C_{m-1}} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & f_{(m-2)(m-1)} & \frac{C_{m-1}f_{(m-2)m} - f_{(m-1)(m-1)}f_{(m-2)(m-1)}}{C_{m-1}} \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & 0 & \dots & C_{m-1} & 0 \end{bmatrix}$$

By Lemma 7.1.5, replace $row(m-1)$ by $row(m)$ and at the same time replace $column(m-1)$ by $column(m)$ to get:

$$\partial = \begin{bmatrix} f_{11} & f_{12} & f_{13} & f_{14} & \dots & \frac{C_{m-1}f_{1m}-f_{(m-1)(m-1)}f_{1(m-1)}}{C_{m-1}} & f_{1(m-1)} \\ 0 & f_{22} & f_{23} & f_{24} & \dots & \frac{C_{m-1}f_{2m}-f_{(m-1)(m-1)}f_{2(m-1)}}{C_{m-1}} & f_{2(m-1)} \\ 0 & 0 & f_{33} & f_{34} & \dots & \frac{C_{m-1}f_{3m}-f_{(m-1)(m-1)}f_{3(m-1)}}{C_{m-1}} & f_{3(m-1)} \\ 0 & 0 & 0 & f_{44} & \dots & \frac{C_{m-1}f_{4m}-f_{(m-1)(m-1)}f_{4(m-1)}}{C_{m-1}} & f_{4(m-1)} \\ 0 & 0 & 0 & 0 & \dots & \frac{C_{m-1}f_{5m}-f_{(m-1)(m-1)}f_{5(m-1)}}{C_{m-1}} & f_{5(m-1)} \\ 0 & 0 & 0 & 0 & \dots & \frac{C_{m-1}f_{6m}-f_{(m-1)(m-1)}f_{6(m-1)}}{C_{m-1}} & f_{6(m-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & \frac{C_{m-1}f_{(m-2)m}-f_{(m-1)(m-1)}f_{(m-2)(m-1)}}{C_{m-1}} & f_{(m-2)(m-1)} \\ 0 & 0 & 0 & 0 & \dots & 0 & C_{m-1} \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 \end{bmatrix}$$

Therefore,, M is solvable (by the previous proposition).

Case (1.1.. . . .1.2): If $f_{(m-1)(m-2)} = C_{m-2} \neq 0$ then the matrix ∂ is given by

$$\partial = \begin{bmatrix} f_{11} & f_{12} & f_{13} & f_{14} & \dots & f_{1(m-2)} & f_{1(m-1)} & f_{1m} \\ 0 & f_{22} & f_{23} & f_{24} & \dots & f_{2(m-2)} & f_{2(m-1)} & f_{2m} \\ 0 & 0 & f_{33} & f_{34} & \dots & f_{3(m-2)} & f_{3(m-1)} & f_{3m} \\ 0 & 0 & 0 & f_{44} & \dots & f_{4(m-2)} & f_{4(m-1)} & f_{4m} \\ 0 & 0 & 0 & 0 & \dots & f_{5(m-2)} & f_{5(m-1)} & f_{5m} \\ 0 & 0 & 0 & 0 & \dots & f_{6(m-2)} & f_{6(m-1)} & f_{6m} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \\ 0 & 0 & 0 & 0 & \dots & f_{(m-2)(m-2)} & f_{(m-2)(m-1)} & f_{(m-2)m} \\ 0 & 0 & 0 & 0 & \dots & C_{m-2} & f_{(m-1)(m-1)} & f_{(m-1)m} \\ 0 & 0 & 0 & 0 & \dots & 0 & f_{m(m-1)} & f_{mm} \end{bmatrix}$$

In this case either $f_{m(m-1)} = 0$ *or* $f_{m(m-1)} = C_{m-1} \neq 0$.

Case (1.1.. . . .1.2.1): If $f_{m(m-1)} = 0$ then the matrix ∂ is given by

$$
\partial = \begin{bmatrix}
f_{11} & f_{12} & f_{13} & f_{14} & \cdots & f_{1(m-2)} & f_{1(m-1)} & f_{1m} \\
0 & f_{22} & f_{23} & f_{24} & \cdots & f_{2(m-2)} & f_{2(m-1)} & f_{2m} \\
0 & 0 & f_{33} & f_{34} & \cdots & f_{3(m-2)} & f_{3(m-1)} & f_{3m} \\
0 & 0 & 0 & f_{44} & \cdots & f_{4(m-2)} & f_{4(m-1)} & f_{4m} \\
0 & 0 & 0 & 0 & \cdots & f_{5(m-2)} & f_{5(m-1)} & f_{5m} \\
0 & 0 & 0 & 0 & \cdots & f_{6(m-2)} & f_{6(m-1)} & f_{6m} \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \\
0 & 0 & 0 & 0 & \cdots & f_{(m-2)(m-2)} & f_{(m-2)(m-1)} & f_{(m-2)m} \\
0 & 0 & 0 & 0 & \cdots & C_{m-2} & f_{(m-1)(m-1)} & f_{(m-1)m} \\
0 & 0 & 0 & 0 & \cdots & 0 & 0 & f_{mm}
\end{bmatrix}
$$

Since $\partial^2 = 0$, this implies $f^2_{(m-2)(m-2)} + C_{m-2}f_{(m-2)(m-1)} = 0$ and $C_{m-2}f_{(m-2)(m-1)} + f^2_{(m-1)(m-1)} = 0$.

By Lemma 7.1.5, replace $row(m-2)$ by $[row(m-2) - (\frac{f_{(m-2)(m-2)}}{C_{m-2}})row(m-1)]$ and at the same time replace $column(m-1)$ by $[column(m-1) - (\frac{f_{(m-2)(m-2)}}{C_{m-2}})column(m-2)]$ to get:

$$
\partial = \begin{bmatrix}
f_{11} & f_{12} & f_{13} & f_{14} & \cdots & f_{1(m-2)} & \frac{C_{m-2}f_{1(m-1)} - f_{(m-2)(m-2)}f_{1(m-2)}}{C_{m-2}} & f_{1m} \\
0 & f_{22} & f_{23} & f_{24} & \cdots & f_{2(m-2)} & \frac{C_{m-2}f_{2(m-1)} - f_{(m-2)(m-2)}f_{2(m-2)}}{C_{m-2}} & f_{2m} \\
0 & 0 & f_{33} & f_{34} & \cdots & f_{3(m-2)} & \frac{C_{m-2}f_{3(m-)} - f_{(m-2)(m-2)}f_{3(m-2)}}{C_{m-2}} & f_{3m} \\
0 & 0 & 0 & f_{44} & \cdots & f_{4(m-2)} & \frac{C_{m-2}f_{4(m-1)} - f_{(m-2)(m-2)}f_{4(m-2)}}{C_{m-2}} & f_{4m} \\
0 & 0 & 0 & 0 & \cdots & f_{5(m-2)} & \frac{C_{m-2}f_{5(m-1)} - f_{(m-2)(m-2)}f_{5(m-2)}}{C_{m-2}} & f_{5m} \\
0 & 0 & 0 & 0 & \cdots & f_{6(m-2)} & \frac{C_{m-2}f_{6(m-1)} - f_{(m-2)(m-2)}f_{6(m-2)}}{C_{m-2}} & f_{6m} \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \\
0 & 0 & 0 & 0 & \cdots & f_{(m-3)(m-2)} & g & f_{(m-3)m} \\
0 & 0 & 0 & 0 & \cdots & 0 & 0 & h \\
0 & 0 & 0 & 0 & \cdots & C_{m-2} & 0 & f_{(m-1)m} \\
0 & 0 & 0 & 0 & \cdots & 0 & 0 & f_{mm}
\end{bmatrix},
$$

where $g = \frac{C_{m-2}f_{(m-3)(m-1)} - f_{(m-2)(m-2)}f_{(m-3)(m-2)}}{C_{m-2}}$ and

$h = \frac{C_{m-2}f_{(m-2)m} - f_{(m-2)(m-2)}f_{(m-2)(m-2)}}{C_{m-2}}$.

By Lemma 7.1.5, replace $row(m-2)$ by $row(m-1)$ and at the same time replace $column(m-2)$ by $column(m-1)$ to get:

$$\partial = \begin{bmatrix} f_{11} & f_{12} & f_{13} & f_{14} & \dots & \frac{C_{m-2}f_{1(m-1)}-f_{(m-2)(m-2)}f_{1(m-2)}}{C_{m-2}} & f_{1(m-2)} & f_{1m} \\ 0 & f_{22} & f_{23} & f_{24} & \dots & \frac{C_{m-2}f_{2(m-1)}-f_{(m-2)(m-2)}f_{2(m-2)}}{C_{m-2}} & f_{2(m-2)} & f_{2m} \\ 0 & 0 & f_{33} & f_{34} & \dots & \frac{C_{m-2}f_{3(m-)}-f_{(m-2)(m-2)}f_{3(m-2)}}{C_{m-2}} & f_{3(m-2)} & f_{3m} \\ 0 & 0 & 0 & f_{44} & \dots & \frac{C_{m-2}f_{4(m-1)}-f_{(m-2)(m-2)}f_{4(m-2)}}{C_{m-2}} & f_{4(m-2)} & f_{4m} \\ 0 & 0 & 0 & 0 & \dots & \frac{C_{m-2}f_{5(m-1)}-f_{(m-2)(m-2)}f_{5(m-2)}}{C_{m-2}} & f_{5(m-2)} & f_{5m} \\ 0 & 0 & 0 & 0 & \dots & \frac{C_{m-2}f_{6(m-1)}-f_{(m-2)(m-2)}f_{6(m-2)}}{C_{m-2}} & f_{6(m-2)} & f_{6m} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \\ 0 & 0 & 0 & 0 & \dots & g & f_{(m-3)(m-2)} & f_{(m-3)m} \\ 0 & 0 & 0 & 0 & \dots & 0 & C_{m-2} & f_{(m-1)m} \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & h \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & f_{mm} \end{bmatrix},$$

where $g = \frac{C_{m-2}f_{(m-3)(m-1)}-f_{(m-2)(m-2)}f_{(m-3)(m-2)}}{C_{m-2}}$ and

$h = \frac{C_{m-2}f_{(m-2)m}-f_{(m-2)(m-2)}f_{(m-2)(m-2)}}{C_{m-2}}$.

Therefore, M is solvable (by the previous proposition).

Case (1.1.. . . .1.2.2): If $f_{m(m-1)} = C_{m-1} \neq 0$, then the matrix ∂ is given by

$$\partial = \begin{bmatrix} f_{11} & f_{12} & f_{13} & f_{14} & \dots & f_{1(m-2)} & f_{1(m-1)} & f_{1m} \\ 0 & f_{22} & f_{23} & f_{24} & \dots & f_{2(m-2)} & f_{2(m-1)} & f_{2m} \\ 0 & 0 & f_{33} & f_{34} & \dots & f_{3(m-2)} & f_{3(m-1)} & f_{3m} \\ 0 & 0 & 0 & f_{44} & \dots & f_{4(m-2)} & f_{4(m-1)} & f_{4m} \\ 0 & 0 & 0 & 0 & \dots & f_{5(m-2)} & f_{5(m-1)} & f_{5m} \\ 0 & 0 & 0 & 0 & \dots & f_{6(m-2)} & f_{6(m-1)} & f_{6m} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \\ 0 & 0 & 0 & 0 & \dots & f_{(m-2)(m-2)} & f_{(m-2)(m-1)} & f_{(m-2)m} \\ 0 & 0 & 0 & 0 & \dots & C_{m-2} & f_{(m-1)(m-1)} & f_{(m-1)m} \\ 0 & 0 & 0 & 0 & \dots & 0 & C_{m-1} & f_{mm} \end{bmatrix}$$

Since $\partial^2 = 0$, then we multiply $row(m)$ by $column(m-2)$ to get that , $C_{m-2} \cdot C_{m-1} = 0$, but ($C_{m-1} \neq 0$ and $C_{m-2} \neq 0$), which implies to contradiction .Thus, this case is not possible.

Similarly, we discuss the rest cases, and get that M is solvable.

□

We will discuss some other cases which the free finitely generated differential graded R-module M is solvable and then generalize them to the general case as the following:

Proposition 7.2.6. *Let K be a field and let $R = K[x_1, x_2, \dots, x_n]$ be a graded ring of polynomials graded in the negative way. Let M be a free finitely generated differential graded R-module with basis $S = \{e_i\}_{i=1}^{3}$ and with differential ∂ of degree $P \le -2$. Suppose that, $dim(e_i) = k_i$ such that $1 \le i \le 3$ and $k_i > k_{i+1}$. If $k_i - k_{i+1} = t_i$ such that $t_i < -P$, then M is solvable in some cases, if $t_i + t_{i+1} > -P$.*

Proof. M is graded as before (proposition 7.2.2).

Suppose that,

$$\begin{aligned} \partial(e_1) &= f_{11}e_1 + f_{21}e_2 + f_{31}e_3 \\ \partial(e_2) &= f_{12}e_1 + f_{22}e_2 + f_{32}e_3 \\ \partial(e_3) &= f_{13}e_1 + f_{23}e_2 + f_{33}e_3 \end{aligned}$$

Then the matrix ∂ with respect to the basis $\{e_1\}_{i=1}^{3}$ is given by:

$$\partial = \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix}$$

Now,

$$dim(\partial(e_1)) = dim(f_{11}) + dim(e_1),$$

$$k_1 + P = dim(f_{11}) + k_1, \text{ implies that}$$

$$dim(f_{11}) = P, \text{ and thus } degree\ f_{11} = -P.$$

So,

$$dim(\partial(e_1)) = dim(f_{21}) + dim(e_2),$$

$$k_1 + P = dim(f_{21}) + k_2, \text{ implies that}$$

$$dim(f_{21}) = P + k_1 - k_2 = P + t_1 < P - P = 0, \textit{ which implies } deg(f_{21}) = -(P + k_1 - k_2).$$

Also,

$$dim(\partial(e_1)) = dim(f_{31}) + dim(e_3),$$

$$k_1 + P = dim(f_{31}) + k_3, \text{ implies that}$$

$$dim(f_{31}) = P + k_1 - k_3 > P - P = 0, \text{ and thus } f_{31} = 0$$

Also,

$$dim(\partial(e_2)) = dim(f_{12}) + dim(e_1),$$

$$k_2 + P = dim(f_{12}) + k_1, \text{ implies that}$$

$$dim(f_{12}) = k_2 - k_1 + P < 0 \text{ and thus } deg(f_{12}) = -(k_2 - k_1 + P).$$

So,

$$dim(\partial(e_2)) = dim(f_{22}) + dim(e_2),$$

$$k_2 + P = dim(f_{22}) + k_2, \text{ implies that}$$

$$dim(f_{22}) = P + k_2 - k_2 = P, \text{ and thus } deg(f_{22}) = -P.$$

So,

$$dim(\partial(e_2)) = dim(f_{32}) + dim(e_3),$$

$$k_2 + P = dim(f_{32}) + k_3, \text{ implies that}$$

$$dim(f_{32}) = P + k_2 - k_3 < -P + P = 0, \text{ and thus } deg(f_{12}) = -(k_2 - k_3 + P).$$

Also,

$$dim(\partial(e_3)) = dim(f_{13}) + dim(e_1),$$

$$k_3 + P = dim(f_{13}) + k_1, \text{ implies that}$$

$$dim(f_{13}) = k_3 - k_1 + P = P + P < 0 \text{ and thus } deg(f_{13}) = -(k_3 - k_1 + P).$$

So,

$$dim(\partial(e_3)) = dim(f_{23}) + dim(e_2),$$

$$k_3 + P = dim(f_{23}) + k_2, \text{ implies that}$$

$$dim(f_{23}) = P + k_3 - k_2 < 0, \text{ and thus } deg(f_{23}) = -(k_3 - k_2 + P).$$

So,

$$dim(\partial(e_3)) = dim(f_{33}) + dim(e_3),$$

$$k_3 + P = dim(f_{33}) + k_3, \text{ implies that}$$

$$dim(f_{33}) = P, \text{ and thus } deg(f_{33}) = -P.$$

Then the matrix ∂ is given by:

$$\partial = \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ 0 & f_{32} & f_{33} \end{bmatrix}$$

Since $\partial^2 = 0$, multiply $row(3)$ by $column(1)$ to get $f_{32}f_{21} = 0$ implies that $f_{32} = 0$ or $f_{21} = 0$.

Case (1): If $f_{32} = 0$ and $f_{21} \neq 0$, then the matrix ∂ is given by

$$\partial = \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ 0 & 0 & f_{33} \end{bmatrix}$$

Since $\partial^2 = 0$, multiply $row(3)$ by $column(3)$ to get $f_{33}^2 = 0$ implies that $f_{33} = 0$. Hence, the matrix ∂ is given by

$$\partial = \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ 0 & 0 & 0 \end{bmatrix}$$

Since $\partial^2 = 0$, multiply $row(2)$ by $column(1)$ to get $f_{21}f_{11} + f_{21}f_{22} = 0$ implies that $f_{21}[f_{11} + f_{22}] = 0$. Thus, either $f_{21} = 0$ or $f_{11} + f_{22} = 0$. But, $f_{21} \neq 0$ which implies that $f_{11} + f_{22} = 0$ and so $f_{11} = f_{22}$. Hence, the matrix ∂ is given by

Thus,

$$\partial = \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{11} & f_{23} \\ 0 & 0 & 0 \end{bmatrix}$$

Since $\partial^2 = 0$, multiply $row(1)$ by $column(1)$ to get $f_{11}^2 + f_{12}f_{21} = 0$ implies that $f_{11}^2 = f_{12}f_{21}$.

Case (1.1): If $f_{11} = 0$, which implies $f_{12}f_{21} = 0$ and this implies to either $f_{12} = 0$ or $f_{21} = 0$, but, $f_{21} \neq 0$. So $f_{12} = 0$, and then the matrix of ∂ is given by

$$\partial = \begin{bmatrix} 0 & 0 & f_{13} \\ f_{21} & 0 & f_{23} \\ 0 & 0 & 0 \end{bmatrix}$$

By Lemma 7.1.5, replace $row(1)$ by $row(2)$ and at the time replace $column(1)$ by $column(2)$ to get:

$$\partial = \begin{bmatrix} 0 & f_{21} & f_{23} \\ 0 & 0 & f_{13} \\ 0 & 0 & 0 \end{bmatrix}$$

Thus, M is solvable (by proposition 7.2.4).

Case(1.2): If $f_{11} \neq 0$ then $f_{12}f_{21} \neq 0$ and this implies to $f_{12} \neq 0$ and $f_{21} \neq 0$. Therefore, we can not decide whether M is solvable or not by this method.

Case (2): If $f_{21} = 0$ and $f_{32} \neq 0$, then the matrix of ∂ is given by

$$\partial = \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ 0 & f_{22} & f_{23} \\ 0 & f_{32} & f_{33} \end{bmatrix}$$

Since $\partial^2 = 0$ multiply $row(1)$ by $column(1)$ to get $f_{11}^2 = 0$ implies that $f_{11} = 0$ (since R is an integral domain).

Thus,

$$\partial = \begin{bmatrix} 0 & f_{12} & f_{13} \\ 0 & f_{22} & f_{23} \\ 0 & f_{32} & f_{33} \end{bmatrix}$$

Since $\partial^2 = 0$, multiply $row(3)$ by $column(2)$ to get $f_{32}f_{22} + f_{23}f_{33} = 0$ implies that $f_{32}[f_{22} + f_{33}] = 0$.

Thus, either $f_{32} = 0$ or $f_{22}+f_{33} = 0$. But, $f_{32} \neq 0$ which implies that $f_{22}+f_{33} = 0$ and so $f_{22} = f_{33}$. Hence, the matrix ∂ is given by

Thus,

$$\partial = \begin{bmatrix} 0 & f_{12} & f_{13} \\ 0 & f_{22} & f_{23} \\ 0 & f_{32} & f_{22} \end{bmatrix}$$

Since $\partial^2 = 0$, multiply $row(2)$ by $column(2)$ to get $f_{22}^2 + f_{23}f_{32} = 0$ implies that $f_{22}^2 = f_{23}f_{32}$.

Case (2.1): If $f_{22} = 0$ implies that $f_{23}f_{32} = 0$ which implies that, either $f_{23} = 0$ or $f_{32} = 0$. But, $f_{32} \neq 0$ and thus $f_{23} = 0$. Hence, the matrix ∂ is given by

$$\partial = \begin{bmatrix} 0 & f_{12} & f_{13} \\ 0 & 0 & 0 \\ 0 & f_{32} & 0 \end{bmatrix}$$

Since $\partial^2 = 0$, multiply $row(1)$ by $column(1)$ to get $f_{11}^2 = 0$ implies that $f_{11}^2 = 0$ (since R is an integral domain).

By Lemma 7.1.5, replace $row(2)$ by $row(3)$ and at the time replace $column(2)$ by $column(3)$ to get:

$$\partial = \begin{bmatrix} 0 & f_{12} & f_{13} \\ 0 & 0 & f_{32} \\ 0 & 0 & 0 \end{bmatrix}$$

Thus, M is solvable (by proposition 7.2.4).

Case (2.2): If $f_{22} \neq 0$, then $f_{23}f_{32} \neq 0$, and this implies to $f_{23} \neq 0$, and $f_{32} \neq 0$, Therefore, we can not decide whether M is solvable or not by this method.

Case (3): If $f_{32} = 0$ and $f_{21} = 0$, then the matrix ∂ is given by

$$\partial = \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ 0 & f_{22} & f_{23} \\ 0 & 0 & f_{33} \end{bmatrix}$$

Since $\partial^2 = 0$, then we have $f_{11}^2 = f_{22}^2 = f_{33}^2 = 0$, which implies that, $f_{11} = f_{22} = f_{33} = 0$. Then the matrix ∂ is given by

$$\partial = \begin{bmatrix} 0 & f_{12} & f_{13} \\ 0 & 0 & f_{23} \\ 0 & 0 & 0 \end{bmatrix}$$

Thus, M is solvable (by proposition 7.2.4). $\square$

Proposition 7.2.7. *Let K be a field and let $R = K[x_1, x_2, \ldots, x_n]$ be a graded ring of polynomials graded in the negative way. Let M be a free finitely generated differential graded R-module with basis $S = \{e_i\}_{i=1}^4$ and with differential ∂ of degree $P \leq -2$. Suppose that, $dim(e_i) = k_i$ such that $1 \leq i \leq 4$ and $k_i > k_{i+1}$. If $k_i - k_{i+1} = t_i$ such that $t_i < -P$, then M is solvable in some cases, if $t_i + t_{i+1} > -P$.*

Proof. M is graded as before (proposition 7.2.3).

Suppose that,

$$\begin{aligned}
\partial(e_1) &= f_{11}e_1 + f_{21}e_2 + f_{31}e_3 + f_{41}e_4 \\
\partial(e_2) &= f_{12}e_1 + f_{22}e_2 + f_{32}e_3 + f_{42}e_4 \\
\partial(e_3) &= f_{13}e_1 + f_{23}e_2 + f_{33}e_3 + f_{43}e_4 \\
\partial(e_4) &= f_{14}e_1 + f_{24}e_2 + f_{34}e_3 + f_{44}e_4
\end{aligned}$$

Then the matrix of ∂ with respect to the basis $\{e_1\}_{i=1}^{4}$ is given by:

$$\partial = \begin{bmatrix} f_{11} & f_{12} & f_{13} & f_{14} \\ f_{21} & f_{22} & f_{23} & f_{24} \\ f_{31} & f_{32} & f_{33} & f_{34} \\ f_{41} & f_{42} & f_{43} & f_{44} \end{bmatrix}$$

Now,

$$dim(\partial(e_1)) = dim(f_{11}) + dim(e_1),$$

$$k_1 + P = dim(f_{11}) + k_1, \text{ implies that}$$

$$dim(f_{11}) = P, \text{ and thus } deg(f_{11}) = -P.$$

So,

$$dim(\partial(e_1)) = dim(f_{21}) + dim(e_2),$$

$$k_1 + P = dim(f_{21}) + k_2, \text{ implies that}$$

$$dim(f_{21}) = P + k_1 - k_2 = P + t_1 < P - P = 0, \text{ and thus } deg(f_{11}) = -(P + k_1 - k_2).$$

Also,

$$dim(\partial(e_1)) = dim(f_{31}) + dim(e_3),$$

$$k_1 + P = dim(f_{31}) + k_3, \text{ implies that}$$

$$dim(f_{31}) = k_1 - k_3 + P > -P \geq 2, \text{ and thus } f_{31} = 0 \; similarly \; f_{41} = 0$$

Also,

$$dim(\partial(e_2)) = dim(f_{12}) + dim(e_1),$$

$$k_2 + P = dim(f_{12}) + k_1, \text{ implies that}$$

$$dim(f_{12}) = k_2 - k_1 + P < 0 \text{ and thus } deg(f_{12}) = -(k_2 - k_1 + P).$$

So,

$$dim(\partial(e_2)) = dim(f_{22}) + dim(e_2),$$

$$k_2 + P = dim(f_{22}) + k_2, \text{ implies that}$$

$$dim(f_{22}) = P + k_2 - k_2 = P, \text{ and thus } deg(f_{22}) = -P.$$

So,

$$dim(\partial(e_2)) = dim(f_{32}) + dim(e_3),$$

$$k_2 + P = dim(f_{32}) + k_3, \text{ implies that}$$

$$dim(f_{32}) = k_2 - k_3 + P < -P + P = 0, \text{ and thus } deg(f_{32}) = -(P + k_2 - k_3).$$

So,

$$dim(\partial(e_2)) = dim(f_{42}) + dim(e_4),$$

$$k_2 + P = dim(f_{42}) + k_4, \text{ implies that}$$

$$dim(f_{42}) = k_2 - k_4 + P > P - P = 0, \text{ and thus } f_{42} = 0.$$

Also,

$$dim(\partial(e_3)) = dim(f_{13}) + dim(e_1),$$

$$k_3 + P = dim(f_{13}) + k_1, \text{ implies that}$$

$$dim(f_{13}) = k_3 - k_1 + P < 0 \text{ and thus } deg(f_{13}) = -(k_3 - k_1 + P).$$

So,

$$dim(\partial(e_3)) = dim(f_{23}) + dim(e_2),$$

$$k_3 + P = dim(f_{23}) + k_2, \text{ implies that}$$

$$dim(f_{23}) = P + k_3 - k_2 < 0, \text{ and thus } deg(f_{23}) = -(k_3 - k_2 + P).$$

So,

$$dim(\partial(e_3)) = dim(f_{33}) + dim(e_3),$$

$$k_3 + P = dim(f_{33}) + k_3, \text{ implies that}$$

$$dim(f_{33}) = P, \text{ and thus } degree\ f_{33} = -P.$$

So,

$$dim(\partial(e_3)) = dim(f_{43}) + dim(e_4),$$

$$k_3 + P = dim(f_{43}) + k_4, \text{ implies that}$$

$$dim(f_{43}) = k_3 - k_4 + P < 0, \text{ and thus } deg(f_{43}) = -(k_3 - k_4 + P).$$

Also,

$$dim(\partial(e_4)) = dim(f_{14}) + dim(e_1),$$

$$k_4 + P = dim(f_{14}) + k_1, \text{ implies that}$$

$$dim(f_{14}) = k_4 - k_1 + P < 0 \text{ and thus } deg(f_{14}) = -(k_4 - k_1 + P).$$

Similarly, *degree* $f_{24} = -(P+k_4-k_2)$, $deg(f_{34}) = -(P+k_4-k_3)$, and $deg(f_{24}) = -P$.

Hence, the matrix of ∂ is given by

$$\partial = \begin{bmatrix} f_{11} & f_{12} & f_{13} & f_{14} \\ f_{21} & f_{22} & f_{23} & f_{24} \\ 0 & f_{32} & f_{33} & f_{34} \\ 0 & 0 & f_{43} & f_{44} \end{bmatrix}$$

Since $\partial^2 = 0$, multiply $row(4)$ by $column(2)$ to get $f_{43}f_{32} = 0$ implies that $f_{43} = 0$ or $f_{32} = 0$.

Case (1): If $f_{43} = 0$ and $f_{32} \neq 0$, then the matrix ∂ is given by

$$\partial = \begin{bmatrix} f_{11} & f_{12} & f_{13} & f_{14} \\ f_{21} & f_{22} & f_{23} & f_{24} \\ 0 & f_{32} & f_{33} & f_{34} \\ 0 & 0 & 0 & f_{44} \end{bmatrix}$$

Since $\partial^2 = 0$, then we have $f_{44}^2 = 0$ which implies $f_{44} = 0$.

Thus,

$$\partial = \begin{bmatrix} f_{11} & f_{12} & f_{13} & f_{14} \\ f_{21} & f_{22} & f_{23} & f_{24} \\ 0 & f_{32} & f_{33} & f_{34} \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Since $\partial^2 = 0$, multiply $row(3)$ by $column(1)$ to get $f_{32}f_{21} = 0$ implies that $f_{32} = 0$ or $f_{21} = 0$. But, $f_{32} \neq 0$ implies to $f_{21} = 0$.

Thus,

$$\partial = \begin{bmatrix} f_{11} & f_{12} & f_{13} & f_{14} \\ 0 & f_{22} & f_{23} & f_{24} \\ 0 & f_{32} & f_{33} & f_{34} \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Since $\partial^2 = 0$,then we have that $f_{22}^2 + f_{23}f_{32} = 0$ and $f_{23}f_{32} + f_{33}^2 = 0$. Hence, $f_{22} = f_{33}$ and $f_{23}f_{32} = f_{22}^2$.

Case (1.1): If $f_{22} = 0$, then $f_{33} = 0$ and $f_{23}f_{32} = 0$, and this implies to either $f_{23} = 0$ or $f_{32} = 0$, but $f_{32} \neq 0$. So $f_{23} = 0$, and then the matrix of ∂ is given by

Thus,

$$\partial = \begin{bmatrix} f_{11} & f_{12} & f_{13} & f_{14} \\ 0 & 0 & 0 & f_{24} \\ 0 & f_{32} & 0 & f_{34} \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

By Lemma 7.1.5, replace $row(2)$ by $row(3)$ and at the time replace $column(2)$ by $column(3)$ to get:

Thus,

$$\partial = \begin{bmatrix} f_{11} & f_{13} & f_{12} & f_{14} \\ 0 & 0 & f_{32} & f_{34} \\ 0 & 0 & f_{23} & f_{24} \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Since $\partial^2 = 0$,then $f_{11}^2 = 0$ implies $f_{11} = 0$.

Thus,

$$\partial = \begin{bmatrix} 0 & f_{12} & f_{13} & f_{14} \\ 0 & 0 & f_{32} & f_{34} \\ 0 & 0 & 0 & f_{24} \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Therefore, M is solvable (by proposition 7.2.4).

Case (1.2): If $f_{22} \neq 0$, then $f_{33} \neq 0$ and $f_{23}f_{32} \neq 0$, which implies that $f_{23} \neq 0$ and $f_{32} \neq 0$.

Therefore, we can not decide whether M is solvable or not by this method.

Case (2): If $f_{32} = 0$ and $f_{43} \neq 0$, then the matrix of ∂ is given by

$$\partial = \begin{bmatrix} f_{11} & f_{12} & f_{13} & f_{14} \\ f_{21} & f_{22} & f_{23} & f_{24} \\ 0 & 0 & f_{33} & f_{34} \\ 0 & 0 & f_{43} & f_{44} \end{bmatrix}$$

Since $\partial^2 = 0$,then we have that $f_{11}^2 + f_{12}f_{21} = 0$ and $f_{12}f_{21} + f_{22}^2 = 0$. Hence, $f_{11} = f_{22}$ and $f_{12}f_{21} = f_{11}^2$.

Case (2.1): If $f_{11} = 0$, then $f_{22} = 0$ and $f_{12}f_{21} = 0$, and this implies to either $f_{21} = 0$ or $f_{12} = 0$.

Case (2.1.1): If $f_{21} = 0$, then the matrix of ∂ is given by

$$\partial = \begin{bmatrix} 0 & f_{12} & f_{13} & f_{14} \\ 0 & 0 & f_{23} & f_{24} \\ 0 & 0 & f_{33} & f_{34} \\ 0 & 0 & f_{43} & f_{44} \end{bmatrix}$$

Since $\partial^2 = 0$, then we have that $f_{33}^2 + f_{34}f_{43} = 0$ and $f_{34}f_{43} + f_{44}^2 = 0$. Hence, $f_{33} = f_{44}$ and $f_{34}f_{43} = f_{33}^2$.

Case (2.1.1.a): If $f_{33} = 0$, then $f_{44} = 0$ and $f_{34}f_{43} = 0$. implies, $f_{34} = 0$ or $f_{43} = 0$, but $f_{43} \neq 0$ implies to $f_{34} = 0$. Hence, the matrix of ∂ is given by

$$\partial = \begin{bmatrix} 0 & f_{12} & f_{14} & f_{13} \\ 0 & 0 & f_{24} & f_{23} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & f_{43} & 0 \end{bmatrix}$$

By Lemma 7.1.5, replace $row(3)$ by $row(4)$ and at the time replace $column(3)$ by $column(4)$ to get:

$$\partial = \begin{bmatrix} 0 & f_{12} & f_{13} & f_{14} \\ 0 & 0 & 0 & f_{24} \\ 0 & 0 & 0 & f_{43} \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Therefore, M is solvable (by proposition 7.2.4).

Case (2.1.1.b): If $f_{33} \neq 0$, then $f_{44} \neq 0$ and $f_{43}f_{34} \neq 0$. implies to $f_{34} \neq 0$. Therefore, we can not decide whether M is solvable or not by this method.

Case (2.1.2): If $f_{12} = 0$, and $f_{21} \neq 0$, then the matrix of ∂ is given by

$$\partial = \begin{bmatrix} 0 & 0 & f_{13} & f_{14} \\ f_{21} & 0 & f_{23} & f_{24} \\ 0 & 0 & f_{33} & f_{34} \\ 0 & 0 & f_{43} & f_{44} \end{bmatrix}$$

By Lemma 7.1.5, replace $row(1)$ by $row(2)$ and at the time replace $column(1)$ by $column(2)$ to get:

$$\partial = \begin{bmatrix} 0 & f_{21} & f_{23} & f_{24} \\ 0 & 0 & f_{13} & f_{14} \\ 0 & 0 & f_{33} & f_{34} \\ 0 & 0 & f_{43} & f_{44} \end{bmatrix}$$

Since $\partial^2 = 0$, then we have that $f_{33}^2 + f_{34}f_{43} = 0$ and $f_{34}f_{43} + f_{44}^2 = 0$. Hence, $f_{33} = f_{44}$ and $f_{34}f_{43} = f_{33}^2$.

- If $f_{33} = 0$,then M is solvable (Case (2.1.1.a)).
- If $f_{33} \neq 0$, then we can not decide whether M is solvable or not by this method.

Case (3): If $f_{43} = 0$,and $f_{32} = 0$, then the matrix of ∂ is given by

$$\partial = \begin{bmatrix} f_{11} & f_{12} & f_{13} & f_{14} \\ f_{21} & f_{22} & f_{23} & f_{24} \\ 0 & 0 & f_{33} & f_{34} \\ 0 & 0 & 0 & f_{44} \end{bmatrix}$$

Since $\partial^2 = 0$,then we have that $f_{11}^2 + f_{21}f_{12} = 0$ and $f_{21}f_{12} + f_{22}^2 = 0$. Hence, $f_{11} = f_{22}$ and $f_{21}f_{12} = f_{11}^2$.

Case (3.1): If $f_{11} = 0$, then $f_{22} = 0$ and $f_{21}f_{12} = 0$. implies, either $f_{21} = 0$ or $f_{12} = 0$.

Also, since ∂^2 then $f_{33}^2 = 0$ and $f_{44}^2 = 0$. Hence, $f_{33} = f_{44} = 0$.

Case (3.1.1): If $f_{21} = 0$. then the matrix of ∂ is given by

$$\partial = \begin{bmatrix} 0 & f_{12} & f_{13} & f_{14} \\ 0 & 0 & f_{23} & f_{24} \\ 0 & 0 & f_{33} & f_{34} \\ 0 & 0 & 0 & f_{44} \end{bmatrix}$$

Therefore, M is solvable (by proposition 7.2.4).

Case (3.1.2): If $f_{12} = 0$ and $f_{21} \neq 0$, then the matrix of ∂ is given by

$$\partial = \begin{bmatrix} f_{11} & 0 & f_{13} & f_{14} \\ f_{21} & f_{22} & f_{23} & f_{24} \\ 0 & 0 & f_{33} & f_{34} \\ 0 & 0 & 0 & f_{44} \end{bmatrix}$$

By Lemma 7.1.5, replace $row(1)$ by $row(2)$ and at the time replace $column(1)$ by $column(2)$ to get:

$$\partial = \begin{bmatrix} f_{11} & 0 & f_{13} & f_{14} \\ 0 & f_{22} & f_{23} & f_{24} \\ 0 & 0 & f_{33} & f_{34} \\ 0 & 0 & 0 & f_{44} \end{bmatrix}$$

Therefore, M is solvable (by proposition 7.2.4).

□

Therefore, we can generalize proposition 7.2.6 and proposition 7.2.7 to the following proposition:

Proposition 7.2.8. *Let K be a field and let $R = K[x_1, x_2, \ldots, x_n]$ be a graded ring of polynomials graded in the negative way. Let M be a free finitely generated differential graded R-module with basis $S = \{e_i\}_{i=1}^m$, and differential ∂ of degree $P \leq -2$. Suppose, $dim(e_i) = k_i$ such that $1 \leq i \leq m$ and $k_i > k_{i+1}$. If $k_i - k_{i+1} = t_i$ with $t_i < -P$ and $t_i + t_{i+1} > -P$ and the entries on the diagonal of the matrix ∂ with respect to the basis $S = \{e_i\}_{i=1}^m$ are zeros then M is solvable.*

Proof. We will proof this Proposition by using GAP system next Chapter (Section 8.5). □

Remark 7.2.9. Let K be a field and let $R = K[x_1, x_2, \ldots, x_n]$ be a graded ring of polynomials graded in the negative way. Let M be a free finitely generated differential graded R-module with basis $S = \{e_i\}_{i=1}^m$, and differential ∂ of degree $P \leq -2$. Suppose $dim(e_i) = k_i$ such that $1 \leq i \leq m$ and $k_i > k_{i+1}$. If $k_i - k_{i+1} = t_i$ with $t_i < -P$ and $t_i + t_{i+1} \leq -P$ then the module M is outside the classification so we cannot decide, using our methods, whether or not it is solvable.

Proof. M is graded as before (proposition 7.2.4). Suppose that,

$$\begin{aligned}
\partial(e_1) &= f_{11}e_1 + \ldots + f_{m1}e_m, \\
\partial(e_2) &= f_{12}e_1 + \ldots + f_{m2}e_m, \\
&\vdots \\
\partial(e_m) &= f_{1m}e_1 + \ldots + f_{mm}e_m.
\end{aligned}$$

Then the matrix ∂ with respect to the basis $\{e_i\}_{i=1}^m$ is given by:

$$\partial = \begin{bmatrix} f_{11} & f_{12} & \ldots & f_{1m} \\ f_{21} & f_{22} & \ldots & f_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ f_{m1} & f_{m2} & \ldots & f_{mm} \end{bmatrix}$$

Now,

$$dim(\partial(e_1)) = dim(f_{i1}) + dim(e_i), \text{ for each } 1 \leq i \leq 3,$$

$$k_1 + P = dim(f_{i1}) + k_i, \text{ and then,}$$

$$dim(f_{i1}) = (k_1 - k_i) + P < 0, \text{ i.e., } f_{i1} \in R_{k_1 - k_i + P} \neq 0.$$

Hence, $deg(f_{i1}) = -(P + k_1 - k_i)$. Therefore, $f_{i1} \neq 0$, for each $1 \leq i \leq 3$. So,

$$dim\ (\partial(e_1)) = dim(f_{i1}) + dim(e_i), \text{ for each } 4 \leq i \leq m,$$

$$k_1 + P = dim(f_{i1}) + k_i, \text{ and then } dim(f_{i1}) = P + k_1 - k_i.$$

Therefore, $f_{i1} = 0$ or $deg(f_{i1}) = -(P + k_1 - k_i)$, for each $4 \leq i \leq m$. Also,

$$dim(\partial(e_2)) = dim(f_{i2}) + dim(e_i), \text{ for each } 1 \leq i \leq 4,$$

$$k_2 + P = dim(f_{i2}) + k_i, \textit{ and then}$$

$$dim(f_{i2}) = P + k_2 - k_i < 0, \text{ i.e., } f_{i2} \in R_{P + k_2 - k_i} \neq 0.$$

Hence, $deg(f_{i2}) = -(P + k_2 - k_i)$. Therefore, $f_{i2} \neq 0$, for each $1 \leq i \leq 4$. Also,

$$dim(\partial(e_2)) = dim(f_{i2}) + dim(e_i), \text{ for each } 5 \leq i \leq m,$$

$$k_2 + P = dim(f_{i2}) + k_i, \text{ and then } dim(f_{i2}) = P + k_2 - k_i.$$

Therefore, $f_{i2} = 0$ or $deg(f_{i2}) = -(P + k_1 - k_i)$, for each $5 \leq i \leq m$.

Now,

$$dim(\partial(e_{m-1})) = dim(f_{i(m-1)}) + dim(e_i), \text{ for each } 1 \leq i \leq m,$$

$$k_{m-1} + P = dim(f_{i(m-1)}) + k_i, \text{ and then}$$

$$dim(f_{i(m-1)}) = P + k_{m-1} - k_i < 0, \text{ i.e., } f_{i(m-1)} \in R_{k_{m-1}-k_iP} \neq 0$$

Hence, $deg(f_{i(m-1)}) = -(k_{m-1} - k_i + P)$. Therefore, $f_{i(m-1)} \neq 0$, for each $1 \leq i \leq m$.

Now,

$$dim(\partial(e_m)) = dim(f_{im}) + dim(e_i), \text{ for each } 1 \leq i \leq m.$$

$$k_m + P = dim(f_{im}) + k_i, \text{ and then}$$

$$dim(f_{im}) = P + k_m - k_i < 0, \text{i.e., } f_{im} \in R_{P+k_m-k_1} \neq 0.$$

Hence, $deg(f_{im}) = -(P + k_m - k_i)$. Therefore, $f_{im} \neq 0$, for each $1 \leq i \leq m$.

Thus the matrix ∂ is given by:

$$\partial = \begin{bmatrix} f_{11} & f_{12} & \cdots & f_{1(m-1)} & f_{1m} \\ f_{21} & f_{22} & \cdots & f_{2(m-1)} & f_{2m} \\ f_{31} & f_{32} & \cdots & f_{3(m-1)} & f_{3m} \\ \vdots & \vdots & & \vdots & \vdots \\ f_{(m-1)1} & f_{(m-1)2} & \cdots & f_{(m-1)(m-1)} & f_{(m-1)m} \\ f_{m1} & f_{m2} & \cdots & f_{m(m-1)} & f_{mm} \end{bmatrix}$$

where,

$$f_{i1} = 0 \; or \; deg(f_{i1}) = -(P + k_1 - k_i), \forall\, 4 \leq i \leq m.$$
$$f_{i2} = 0 \; or \; deg(f_{i2}) = -(P + k_2 - k_i), \forall\, 5 \leq i \leq m.$$
$$f_{i3} = 0 \; or \; deg(f_{i3}) = -(P + k_3 - k_i), \forall\, 6 \leq i \leq m.$$

$$\vdots$$

$$f_{i(m-3)} = 0 \; or \; deg(f_{i(m-3)}) = -(P + k_{m-3} - k_i), \forall\, i = m.$$

Therefore, in this case we cannot decide, using our methods, whether or not M is solvable, because we are unable to convert the matrix ∂ to a strictly upper triangular matrix. Hence we can't forming a composition series of a free finitely generated differential graded R-submodules.

□

Proposition 7.2.10. *Let K be a field and let $R = K[x_1, x_2, \dots, x_n]$ be a graded ring of polynomials graded in the negative way. Let M be a free finitely generated differential graded R-module with basis $S = \{e_i\}_{i=1}^m$, and differential ∂ of degree $P \leq -2$. Suppose $dim(e_i) = k_i$ for $1 \leq i \leq m$, such that $k_i < k_{i+1}$. If $k_i - k_{i+1} = t_i$ with $t_i < P$, then M is solvable.*

Proof. M is graded as before (proposition 7.2.4).

Suppose that,

$$\begin{aligned}
\partial(e_1) &= f_{11}e_1 + \ldots + f_{m1}e_m, \\
\partial(e_2) &= f_{12}e_1 + \ldots + f_{m2}e_m, \\
&\vdots \\
\partial(e_m) &= f_{1m}e_1 + \ldots + f_{mm}e_m.
\end{aligned}$$

Then the matrix ∂ with respect to the basis $\{e_i\}_{i=1}^m$ is given by:

$$\partial = \begin{bmatrix} f_{11} & f_{12} & \dots & f_{1m} \\ f_{21} & f_{22} & \dots & f_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ f_{m1} & f_{m2} & \dots & f_{mm} \end{bmatrix}$$

Now,

$$dim(\partial(e_1)) = dim(f_{i1}) + dim(e_i), \text{ for each } 1 \leq i \leq m,$$

$$k_1 + P = dim(f_{i1}) + k_i, \ \textit{implies}\ dim(f_{i1}) = P - (k_i - k_1) < 0, \text{i.e., } f_{i1} \in R_{P-(k_i-k_1)} \neq 0.$$

Therefore, $f_{i1} \neq 0$, for $1 \leq i \leq m$.

Also,

$$dim(\partial(e_2)) = dim(f_{i2}) + dim(e_i), \text{ for } 1 \leq i \leq m.$$

$$k_2 + P = dim(f_{i2}) + k_i, \text{ implies to } dim(f_{i2}) = P - (k_i - k_2) < 0 \text{ for } 2 \leq i \leq m.$$

Therefore, $f_{i2} \in R_{P-k_2-k_i} \neq 0$ and so $f_{i2} \neq 0$, for $2 \leq i \leq m$. While $dim(f_{12}) = P - (k_1 - k_2) > 0$ for $i = 1$, i.e., $f_{12} \in R_{P-(k_1-k_2)} = 0$ and so $f_{12} = 0$.

Now,

$$dim(\partial(e_m)) = dim(f_{im}) + dim(e_i), \text{ for } 1 \leq i \leq m.$$

$$k_m + P = dim(f_{im}) + k_i, \text{ implies to } dim(f_{im}) = P - (k_i - k_m),$$

This implies to $dim(f_{im}) > 0$ for $1 \leq i \leq m-1$, i.e., $f_{im} \in R_{P-(k_i-k_m)} = 0$ for $1 \leq i \leq m-1$, and $dim(f_{im}) < 0$ for $i = m$, i.e., $f_{im} \in R_P \neq 0$. Hence, $f_{im} = 0$ for $1 \leq i \leq m-1$ and $f_{im} \neq 0$ for $i = m$.

Then the matrix ∂ with respect to the basis $\{e_i\}_{i=1}^{m}$ is given by:

$$\partial = \begin{bmatrix} f_{11} & 0 & 0 & 0 & \dots & 0 & 0 \\ f_{21} & f_{22} & 0 & 0 & \dots & 0 & 0 \\ f_{31} & f_{32} & f_{33} & 0 & \dots & 0 & 0 \\ f_{41} & f_{42} & f_{43} & f_{44} & \dots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \ddots & 0 & 0 \\ f_{(m-1)1} & f_{(m-1)2} & f_{(m-1)3} & f_{(m-1)4} & \dots & f_{(m-1)(m-1)} & 0 \\ f_{m1} & f_{m2} & f_{m3} & f_{m4} & \dots & f_{m(m-1)} & f_{mm} \end{bmatrix}$$

Since $\partial^2 = 0$, this implies $f_{ii} = 0$ for $1 \leq i \leq m$. So the matrix ∂ become that

$$\partial = \begin{bmatrix} 0 & 0 & 0 & 0 & \dots & 0 & 0 \\ f_{21} & 0 & 0 & 0 & \dots & 0 & 0 \\ f_{31} & f_{32} & 0 & 0 & \dots & 0 & 0 \\ f_{41} & f_{42} & f_{43} & 0 & \dots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \ddots & 0 & 0 \\ f_{(m-1)1} & f_{(m-1)2} & f_{(m-1)3} & f_{(m-1)4} & \dots & 0 & 0 \\ f_{m1} & f_{m2} & f_{m3} & f_{m4} & \dots & f_{m(m-1)} & 0 \end{bmatrix}$$

By using Lemma 7.1.5 we will convert the matrix ∂ to a strictly upper triangular matrix as follows:

$$\partial = \begin{bmatrix} 0 & f_{12} & f_{31} & f_{41} & \dots & f_{1(m-2)} & f_{(m-1)}1 & f_{m1} \\ 0 & 0 & f_{32} & f_{42} & \dots & f_{2(m-2)} & f_{(m-1)2} & f_{m2} \\ 0 & 0 & 0 & f_{43} & \dots & f_{3(m-2)} & f_{(m-1)3} & f_{m3} \\ 0 & 0 & 0 & 0 & \dots & f_{(m-2)4} & f_{(m-1)4} & f_{m4} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 0 & f_{(m-2)(m-1)} & f_{m(m-2)} \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & f_{m(m-1)} \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 \end{bmatrix}$$

Therefore, M is solvable. $\square$

Example 7.2.0.13
Let $R = K[x_1, x_2, ..., x_n]$, be the ring of polynomials in n indetrminates over a field K of characteristic two. Let

$R_j = 0$ for all $j < 0$,

$R_0 = K$, and

$R_j =$ the set of all homogeneous polynomials of *degree* j for all $j > 0$. Then R is a graded K-algebra and a graded integral domain, called the **usual grading or (positive grading)**.

Note that in R, if $dim(f) = j$, i.e., $f \in R_j$ then *degree* of $f = -j$.

Proposition 7.2.11. *Let K be a field and let $R = K[x_1, x_2, \ldots, x_n]$ be a graded polynomial ring graded in the usual way. Let M be a free finitely generated differential graded R-module with basis $S = \{e_i\}_{i=1}^m$, and differential ∂ of degree $(P \geq 2, n > 1)$. Suppose, $dim(e_i) = k_i$ such that $1 \leq i \leq m$. If $k_1 < k_2 < \ldots < k_m$ and $k_{i+1} - k_i > P$ then M is solvable.*

Proof. Suppose that $e_1 \in M_{k_1},\ e_2 \in M_{k_2}, \ldots, e_m \in M_{k_m}$.

Suppose that,

$$\begin{aligned}
\partial(e_1) &= f_{11}e_1 + \ldots + f_{m1}e_m, \\
\partial(e_2) &= f_{12}e_1 + \ldots + f_{m2}e_m, \\
&\vdots \\
\partial(e_m) &= f_{1m}e_1 + \ldots + f_{mm}e_m.
\end{aligned}$$

Then the matrix of ∂ with respect to the basis $\{e_1\}_{i=1}^m$ is given by:

$$\partial = \begin{bmatrix} f_{11} & f_{12} & \cdots & f_{1m} \\ f_{21} & f_{22} & \cdots & f_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ f_{m1} & f_{m2} & \cdots & f_{mm} \end{bmatrix}$$

Now,

$$dim(\partial(e_1)) = dim(f_{11}) + dim(e_1),$$

$$k_1 - P = dim(f_{11}) + k_1, \text{ implies that } dim(f_{11}) = -P < 0 \text{ and then } deg(f_{11}) = 0.$$

Also,

$$dim(\partial(e_1)) = dim(f_{i1}) + dim(e_i) \text{ for each } 1 \leq i \leq m.$$

So,

$$k_1 - P = dim(f_{i1}) + k_i \text{ and then}$$

$$dim(f_{i1}) = (k_1 - k_i) - P < 0, \text{ i.e., } f_{i1} \in R_{k_1 - k_i - P} = 0.$$

Therefore,

$$f_{i1} = 0 \text{ for each } 1 \leq i \leq m.$$

Also,

$$dim(\partial(e_2)) = dim(f_{12}) + dim(e_1),$$

$$k_2 - P = dim(f_{12}) + k_1,$$

$$dim(f_{12}) = k_2 - k_1 - P > 0 \text{ implies that,}$$

$$deg(f_{12}) = -(k_2 - k_1 - P).$$

So,

$$dim(\partial(e_2)) = dim(f_{22}) + dim(e_2),$$

$$k_2 - P = dim(f_{22}) + k_2, \text{ implies that } deg(f_{22}) = 0.$$

So,

$$dim(\partial(e_2)) = dim(f_{i2}) + dim(e_i) \text{ for each } 2 \leq i \leq m,$$

$$k_2 - P = dim(f_{i2}) + k_i \text{ and then}$$

$$dim(f_{i2}) = (k_2 - k_i) - P < 0, \text{ i.e., } f_{i2} \in R_{P + k_2 - k_i} = 0.$$

Therefore,

$$f_{i2} = 0 \text{ for each } 2 \leq i \leq m.$$

Now,

$$dim(\partial(e_{m-1})) = dim(f_{i(m-1)}) + dim(e_i) \text{ for each } 1 \leq i \leq m-1,$$

$$k_{m-1} - P = dim(f_{i(m-1)}) + k_i \text{ and then}$$

$$dim(f_{i(m-1)}) = (k_{m-1} - k_i - P) < 0, \text{ i.e., } f_{i(m-1)} \in R_{k_{m-1} - k_i - P} \neq 0.$$

Therefore,

$$f_{i(m-1)} \neq 0 \text{ for each } 1 \leq i \leq m-1,$$

and,

$$dim(\partial(e_{m-1})) = dim(f_{m(m-1)}) + dim(e_m),$$

$$k_{m-1} - P = dim(f_{m(m-1)} + k_m, \text{ implies that}$$

$$dim(f_{m(m-1))} = k_{m-1} - k_m - P < 0 \text{ which implies that } f_{m(m-1)} = 0.$$

Also,

$$dim(\partial(e_m)) = dim(f_{im}) + dim(e_i) \text{ for each } 1 \leq i \leq m-1,$$

$$k_m - P = dim(f_{im}) + k_i \text{ and then}$$

$$dim(f_{i(m)}) = (k_m - k_i) - P > 0, \text{ i.e., } f_{im} \in R_{k_m - k_i - P} \neq 0.$$

Therefore,

$$f_{im} \neq 0 \text{ for each } 1 \leq i \leq m-1.$$

Finally,

$$dim(\partial(e_m)) = dim(f_{mm}) + dim(e_m),$$

$k_m - P = dim(f_{mm}) + k_m$, implies that $dim(f_{mm}) = -P < 0$ and then $deg(f_{mm}) = 0$.

Hence, the matrix of ∂ is given by:

$$\partial = \begin{bmatrix} 0 & f_{12} & f_{13} & f_{14} & \dots & f_{1(m-1)} & f_{1m} \\ 0 & 0 & f_{23} & f_{24} & \dots & f_{2(m-1)} & f_{2m} \\ 0 & 0 & 0 & f_{34} & \dots & f_{3(m-1)} & f_{3m} \\ 0 & 0 & 0 & 0 & \dots & f_{4(m-1)} & f_{4m} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 0 & f_{(m-1)m} \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 \end{bmatrix}$$

To show, M has a composition series:
Let $C_0 = 0$ and $C_j = \langle e_1, e_2, \dots, e_j \rangle$, for all $1 \leq j \leq m$.
Then (C_j/C_{j-1}) is free. If $x \in C_j$, then x can be written uniquely as:

$$x = \alpha_1 e_1 + \alpha_2 e_2 + \dots + \alpha_j e_j.$$

Thus,

$$\partial(x) = \alpha_1 \partial(e_1) + \alpha_2 \partial(e_2) + \dots + \alpha_j \partial(e_j)$$

$$\partial(x) = \alpha_1(0) + \alpha_2(f_{12}e_1) + \dots + \alpha_j(f_{1j}e_1 + \dots + f_{(j-1)j}e_{j-1}) \in C_{j-1}$$

Therefore,

$$\partial(C_j/C_{j-1}) = 0, \text{ for each } 1 \leq j \leq m.$$

Hence, $0 = C_0 \subseteq C_1 \subseteq C_2 \subseteq \ldots \subseteq C_m = M$ is a composition series for M. Thus, M is solvable. $\square$

Chapter 8

GAP Algorithm for Solvable Differential Graded Modules

We have established a classification for some types of differential graded R-modules. This classification gives a partial algorithm to test whether such modules are solvable. For modules outside the classification we cannot decide, using our methods, whether or not they are solvable. In this Chapter we present an algorithm and written a *GAP* package *SDGM* (**Solvable Differential Graded R-Modules**), for all the cases mentioned in Chapter 7 (Propositions 7.2.4, 7.2.5, 7.2.8, 7.2.10, 7.2.11 and Remark 7.2.9). The classification described in Chapter 7 depends on two basic parameters; the dimensions $D = [k_1, \ldots, k_n]$ of the module M, such that $dim(e_i) = k_i$, and the degree P of the differential on the module M where $(n > 1)$. These two parameters represent the input for the main function `IsSolvableModuleWithProof` of our algorithm. The output of `IsSolvableModuleWithProof` is either "true" if M is a solvable module, and in this case a proof that M is solvable is also output; or "fail" if we cannot convert the matrix d of the differential ∂ with respect to the basis $S = \{e_i\}_{i=1}^{m}$ to a strictly upper triangular matrix. The function `IsSolvableModuleWithProof` contains many other functions: in the following we describe all the functions used.

8.1 `SwapRowsColumns` Function

The input of the function `SwapRowsColumns`$(degf, x, y)$ is a matrix $degf$ of size $m \times m$ and two numbers $x \neq y$, with $1 \leq x \leq m$, $1 \leq y \leq m$. It exchanges $row(x)$

and $row(y)$, and at the same time exchange, $column(x)$ and $column(y)$. It returns the matrix $degf$ after the replacement. The function works as follows:

SwapRowsColumns($degf, x, y$)

$Temp5 \leftarrow$ StructuralCopy($degf$) $\triangleright$ $Temp5$ was empty list
$degf[x] \leftarrow Temp5[y]$
$degf[y] \leftarrow Temp5[x]$
$degf \leftarrow$ TransposedMatDestructive($degf$)
$Temp6 \leftarrow$ StructuralCopy($degf$) $\triangleright$ $Temp6$ was empty list
$degf[x] \leftarrow Temp6[y]$
$degf[y] \leftarrow Temp6[x]$
$degf \leftarrow$ TransposedMatDestructive($degf$)
return *degf*

8.2 `Solveindic1WithProof` Function

The function `Solveindic1WithProof`($m, dimf, f$) is called only if the conditions of Propositions 7.2.4,7.2.5 hold. The inputs of this function are the dimension m of the vector of dimensions, the matrix $dimf$ of dimensions and the identity matrix f of size $m \times m$ which are output by the main function `IsSolvableModuleWithProof`. The function outputs a proof that M is solvable. The function works as follows:

Solveindic1WithProof($m, dimf, f$)

for j **in** $\{1, \ldots, m\}$
  do for i **in** $\{1, \ldots, m\}$
    do if $i > j$
      then if $dimf[i][j] \geq 0$
        then $0 \leftarrow f[i][j]$
        else $f[i][j] = dimf[i][j]$
      else $f[i][j] = dimf[i][j]$
if f is an upper triangular matrix
  then for j **in** $\{1, \ldots, m\}$
    do Compute matrix d of ∂ with respect to the basis $S = \{e_i\}_{i=1}^{m}$
      using the fact that $\partial^2 = 0$ and R is an integral domain
  else Return f is not upper triangluar matrix

CONSTRUCT a proof that M is solvable
return M *is solvable*

8.3 `Solveindic2WithProof` Function

The function `Solveindic2WithProof`$(dimf, m)$ is called only if the conditions of Remark 7.2.9 or the first case of Proposition 7.2.8 (as in Remark 8.5.1(i)) hold. The inputs of this function are the matrix $dimf$ of dimensions, the dimension m of the vector of dimensions and the matrix 'degf' of size $m \times m$ which are output by the main function `IsSolvableModuleWithProof`. The function is called if the modules M is outside the classification or if (i) of Remark 8.5.1 hold. The function works as follows:

SOLVEINDIC2WITHPROOF$(dimf, m)$

$f \leftarrow dim$
for j **in** $\{1, \ldots, m-2\}$
  do for i **in** $\{1, \ldots, m\}$
    do if $i < j+2$
      then if $dimf[i][j] < 0$
        then $f[i][j] = dimf[i][j]$
        else $0 \leftarrow f[i][j]$
          $\triangleright$ since $\partial^2 = 0$ and R is an integral domain
      else if $dimf[i][j] < 0$
        then $f[i][j] = dimf[i][j]$
        else $0 \leftarrow f[i][j]$
COMPUTE matrix d of the differential ∂ with respect to the basis $S = \{e_i\}_{i=1}^{m}$
return M is outside the classification

8.4 `Solveindic3WithProof` Function

The function `Solveindic3WithProof`$(m, dimf, f)$ is called only if the conditions of Proposition 7.2.10 hold. The inputs of this function are the dimension m of the vector of dimensions, the matrix $dimf$ of dimensions and the identity matrix f of size $m \times m$ which are output by the main function `IsSolvableModuleWithProof`. The function outputs a proof that M is solvable. The function works as follows:

Solveindic3WithProof$(m, dimf, f)$

for j **in** $\{1, \ldots, m\}$
  do for i **in** $\{1, \ldots, m\}$
    do if $i > j$
      then if $dimf[i][j] \geq 0$
        then $0 \leftarrow f[i][j]$
        else $f[i][j] = dimf[i][j]$
      else $f[i][j] = dimf[i][j]$
for i **in** $\{1, \ldots, m\}$
  do $0 \leftarrow f[i][j]$ $\triangleright$ since $\partial^2 = 0$ and R is an integral domain
$Tranf \leftarrow$ TransposedMatDestructive(f)
if $Tranf$ is an upper triangular matrix
  then Compute matrix d of ∂ with respect to the basis $S = \{e_i\}_{i=1}^{m}$
Construct a proof that M is solvable
return M *is solvable*

8.5 `Solveindic4WithProof` Function

The function `Solveindic4WithProof`$(degf)$ is called only if the conditions of Proposition 7.2.8 hold. The input of this function is a matrix $degf$ of size $m \times m$ which is output by the main function `IsSolvableModuleWithProof`. It calls the following functions: `Solveindic4Size3by3`$(degf)$, `Solveindic4Size4by4A`$(degf)$, `Solveindic4Size4by4B`$(degf)$, `Solveindic4Size5by5`$(degf)$, `Solveindic4Size6by6`$(degf)$, `Solveindic4Size6by6Above`$(degf)$ and `Solveindic4Sizembym`$(degf)$ (which will be described later in Section 8.5.1, …, Section 8.5.8 respectively.) The function outputs a proof that M is solvable.

Remark 8.5.1. When we run the main function `IsSolvableModuleWithProof` with input that satisfies the conditions of Proposition 7.2.8, we will at some stage get the matrix $degf$ of size $m \times m$ with $m \geq 2$. In this case `IsSolvableModuleWithProof` calls the function `Solveindic4`; (which calls the following functions: `Solveindic4Size3by3`, `Solveindic4Size4by4A`, `Solveindic4Size4by4B`, `Solveindic4Size5by5`, `Solveindic4Size6by6`, `Solveindic4Size6by6Above` and `Solveindic4Sizembym`.

(i) If $degf = \begin{pmatrix} f_{11} & f_{12} \\ f_{21} & f_{22} \end{pmatrix}$, that is in the case $f_{11} = f_{22} = 0$ then the function $Mysolve2a(degf)$ is called.

(ii) If $degf = \begin{pmatrix} 0 & f_{12} & f_{13} \\ f_{21} & 0 & f_{23} \\ 0 & f_{32} & 0 \end{pmatrix}$, that is in the case $f_{32} = 0$ and $f_{12} = 0$ then the function $Mysolve3a(degf)$ is called.

(iii) If $degf = \begin{pmatrix} 0 & f_{12} & f_{13} & f_{14} \\ f_{21} & 0 & f_{23} & f_{24} \\ 0 & 0 & f_{32} & 0 \\ 0 & 0 & f_{43} & 0 \end{pmatrix}$, that is in the case $f_{32} = 0$ with either $f_{43} = 0$ or $f_{43} \neq 0$ (these are encoded as $b = [0]$ and $b = [1]$ respectively) then the function $Mysolve4a(degf)$ is called.

(iv) If $degf = \begin{pmatrix} 0 & f_{12} & f_{13} & f_{14} \\ f_{21} & 0 & 0 & f_{24} \\ 0 & f_{32} & 0 & 0 \\ 0 & 0 & f_{43} & 0 \end{pmatrix}$, that is in the case $f_{32} \neq 0$ and $f_{43} = 0$ (this is encoded as $b = [0]$) then the function $Mysolve4b(degf)$ is called.

(v) If $degf = \begin{pmatrix} 0 & f_{12} & 0 \\ 0 & 0 & 0 \\ 0 & f_{32} & 0 \end{pmatrix}$, that is in the case $f_{32} \neq 0$ then $Mysolve3b(degf)$ or $Mysolgeneral(degf)$ is calld when $m = 3$.

(vi) If $degf = \begin{pmatrix} 0 & f_{12} & f_{13} & f_{14} & f_{15} \\ f_{21} & 0 & f_{23} & f_{24} & f_{25} \\ 0 & f_{32} & 0 & 0 & f_{35} \\ 0 & 0 & f_{43} & 0 & 0 \\ 0 & 0 & 0 & f_{54} & 0 \end{pmatrix}$, that is in the case $f_{32} = 0$ and $f_{43} = f_{54} \neq 0$ (this is encoded as $b = [1, 1]$) then the function $Mysolve5a-$ $(degf)$ is called.

(vii) If $degf = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ f_{21} & 0 & f_{23} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & f_{43} & 0 & 0 & 0 \\ 0 & 0 & 0 & f_{54} & 0 & 0 \\ 0 & 0 & 0 & 0 & f_{65} & 0 \end{pmatrix}$, that is in the case $f_{32} = 0$, $f_{12} = 0$ and $f_{43} = f_{54} = f_{65} \neq 0$ (this is encoded as $b = [1, 1, 1]$) then the function $Mysolvable6$ is called.

(viii) If $degf = \begin{pmatrix} 0 & f_{12} & f_{13} & f_{14} & f_{15} & \cdots & f_{1m} \\ f_{21} & 0 & f_{23} & f_{24} & f_{25} & \cdots & f_{2m} \\ 0 & f_{32} & 0 & 0 & f_{35} & \cdots & f_{3m} \\ 0 & 0 & f_{43} & 0 & 0 & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & f_{(m-2)m} \\ 0 & 0 & 0 & 0 & f_{(m-1)(m-2)} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & f_{m(m-1)} & 0 \end{pmatrix}$, that is in the case $f_{32} = 0$, $f_{12} = 0$ and $f_{43} = f_{54} = f_{65} = \ldots = f_{m(m-1)} \neq 0$ (this is encoded as $b = [1, 1, \ldots, 1]$) then the function $Mysolvable1$ is called when $m \geq 6$.

1. (ix) If $degf = \begin{pmatrix} 0 & f_{12} & f_{13} & f_{14} & f_{15} & \cdots & f_{1m} \\ f_{21} & 0 & f_{23} & f_{24} & f_{25} & \cdots & f_{2m} \\ 0 & f_{32} & 0 & 0 & f_{35} & \cdots & f_{3m} \\ 0 & 0 & f_{43} & 0 & 0 & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & f_{(m-2)m} \\ 0 & 0 & 0 & 0 & f_{(m-1)(m-2)} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & f_{m(m-1)} & 0 \end{pmatrix}$, that is in the case $f_{32} \neq 0$, $f_{21} = 0$ and $f_{43} = f_{54} = f_{65} = \ldots = f_{m(m-1)} \neq 0$ (this is encoded as $b = [1, 1, \ldots, 1]$) then the function $Mysolgene-$ $ral(degf)$ is called when $m \geq 3$.

In detail the function works as follows:

SOLVEINDIC4WITHPROOF($degf$)

$m \leftarrow$ SIZE($degf$)
if $m = 2$
    then SOLVEINDIC4SIZE2BY2($degf$)
for i **in** $\{1, \ldots, 2^{m-3}\}$
        do $b \leftarrow$ CONVERTTOBINARY($i - 1$)
            for j **in** $\{1, \ldots, m-3\}$ and $j1 = j + 3$
                do if $b[j] = 0$
                      then $0 \leftarrow degf[j1][j1-1] = degf[j1][j1]$
                    if $b[j] = 1$
                      then $0 \leftarrow degf[j1][j1] = degf[j1-1][j1]$

```
        0 ← degf[i][i] for i = 1, 2, 3 ▷ by the hypothesis of Proposition 7.2.8
        Temp4 ← StructuralCopy(degf) after set Temp4 to empty list
        g ← Sum(b)
        degf ← StructuralCopy(Temp4)
        if g = 0
          then if m = 3
                 then degf ← Solveindic4Size3by3(degf)
               if m ≥ 4
                 then degf ← Solveindic4Size4by4A(degf)
                      degf ← StructuralCopy(Temp4)
                      degf ← Solveindic4Size4by4B(degf)
        if g = m − 3
          then if m = 3
                 then degf ← Solveindic4SizembyM(degf)
               if m = 4
                 then degf ← Solveindic4Size4by4A(degf)
                      degf ← StructuralCopy(Temp4)
                      degf ← Solveindic4SizembyM(degf)
               if m = 5
                 then degf ← Solveindic4Size5by5(degf)
                      degf ← StructuralCopy(Temp4)
                      degf ← Solveindic4SizembyM(degf)
               if m ≥ 6
                 then degf ← Solveindic4Size6by6Above(degf)
                      degf ← StructuralCopy(Temp4)
                      degf ← Solveindic4SizembyM(degf)
return degf
```

8.5.1 `Solveindic4Size2by2` Function

The input of the function `Solveindic4Size2by2`$(degf)$ is a matrix $degf$ of size 2×2 as in Remark 8.5.1(i). `Solveindic4Size2by2` convertes the matrix $degf$ to an upper Triangular matrix. It returns the matrix $degf$ after finishing all the replacements. The function works as follows:

Solveindic4Size2by2($degf$)

$degf[1][1] = degf[2][2] = 0$ ▷ by the hypothesis of Proposition 7.2.8
$0 \leftarrow degf[1][2]$ ▷ since $\partial^2 = 0$ and R is an integral domain
$degf \leftarrow$ StructuralCopy($degf$)
$degf \leftarrow$ SwapRowsColumns($degf, 1, 2$)
if $degf$ is not an upper triangular matrix
  then $degf \leftarrow$ Print($degf$) with some comments
  else $degf \leftarrow$ Print($degf$) with some comments
return *degf*

8.5.2 `Solveindic4Size3by3` Function

The input of the function `Solveindic4Size3by3`($degf$) is a matrix $degf$ of size 3×3 as in Remark 8.5.1(ii) (it is Case 1 of 3×3 matrix). `Solveindic4Size3by3` convertes the matrix $degf$ to an upper Triangular matrix. It returns the matrix $degf$ after replacement and tests whether it is a strictly upper triangular matrix or not. The function works as follows:

Solveindic4Size3by3($degf$)

$degf[3][2] = degf[1][2] = 0$ ▷ by the hypothesis of Proposition 7.2.8
$degf \leftarrow$ StructuralCopy($degf$)
$degf \leftarrow$ SwapRowsColumns($degf, 1, 2$)
if $degf$ is not an upper triangular matrix
  then $degf \leftarrow$ Print($degf$) with some comments
  else $degf \leftarrow$ Print($degf$) with some comments
return *degf*

8.5.3 `Solveindic4Size4by4A` Function

The input of the function `Solveindic4Size4by4`($degf$) is a matrix $degf$ of size $m \times m$ where $m \geq 4$ and $f_{ii} = 0$, $i = 1, \ldots, m$ and $f_{32} = 0$ with $Sum(b) = 0$ as in Remark 8.5.1(iii). `Solveindic4Size4by4A` convertes the matrix $degf$ to an upper Triangular matrix. It returns the matrix $degf$ after replacement and tests whether it is a strictly upper triangular matrix or not. The function works as follows:

SOLVEINDIC4SIZE4BY4A($degf$)

```
degf[3][2] = degf[1][2] = 0   ▷ by the hypothesis of Proposition 7.2.8
degf ← STRUCTURALCOPY(degf)
degf ← SWAPROWSCOLUMNS(degf, 1, 2)
if degf is an upper triangular matrix
    then degf ← PRINT(degf) with some comments
    else  degf ← PRINT(degf) with some comments
return degf
```

8.5.4 `Solveindic4Size4by4B` Function

The input of the function `Solveindic4Size4by4B`($degf$) is a matrix $degf$ of size $m \times m$ where $m \geq 4$ and $f_{32} \neq 0$ with zeros on the diagonal and $Sum(b) = 0$. The matrix $degf$ of Remark 8.5.1(iv) is one example of the input of `Solveindic4Size4by4B`. `Mysolve4b` convertes the matrix $degf$ to an upper triangular matrix. It returns the matrix $degf$ after replacement and tests whether it is a strictly upper triangular matrix or not. The function works as follows:

SOLVEINDIC4SIZE4BY4B($degf$)

```
degf ← SIZE(degf)
degf[2][1] = degf[2][3] = 0   ▷ by the hypothesis of Proposition 7.2.8
degf ← SWAPROWSCOLUMNS(degf, 2, 3)
if degf is an upper triangular matrix
    then degf ← PRINT(degf) with some comments
    else  degf ← SWAPROWSCOLUMNS(degf, 3, 4)
          degf[1][3] = 0
          for i in {4, ..., m}
              do degf[1][i] = degf[2][i] = 0
                          ▷ using ∂² = 0 and R is an integral domain
          degf ← SWAPROWSCOLUMNS(degf, 3, 4)
          degf ← SWAPROWSCOLUMNS(degf, 2, 3)
          degf ← SWAPROWSCOLUMNS(degf, 3, 4)
          degf ← PRINT(degf) with some comments
return degf
```

8.5.5 `Solveindic4Size5by5` Function

The input of the function `Solveindic4Size5by5`$(degf)$ is a matrix $degf$ of size 5×5 with $f_{32} = 0$ and $Sum(b) = 2$ as in Remark 8.5.1(vi). `Solveindic4Size5by5` convertes the matrix $degf$ to an upper triangular matrix. It returns the matrix $degf$ after replacement and tests whether it is a strictly upper triangular matrix or not. The function works as follows:

Solveindic4Size5by5$(degf)$

$m \leftarrow$ Size$(degf)$
$degf[1][2] = degf[3][2] = 0$ $\triangleright$ since $\partial^2 = 0$ and R is an integral domain
for i **in** $\{1, \ldots, m\}$
    do for j **in** $\{1, \ldots, m\}$
        if $j \geq i + 2$
            then $f[i][j] = 0$ $\triangleright$ since $\partial^2 = 0$ and R is an integral domain
$degf \leftarrow$ StructuralCopy$(degf)$
$degf \leftarrow$ SwapRowsColumns$(degf, 1, 2)$
if $degf$ is not an upper triangular matrix
    then $degf \leftarrow$ SwapRowsColumns$(degf, 3, 4)$
if $degf$ is not an upper triangular matrix
    then $degf \leftarrow$ SwapRowsColumns$(degf, 4, 5)$
if $degf$ is not an upper triangular matrix
    then $degf \leftarrow$ SwapRowsColumns$(degf, 3, 4)$
if $degf$ is not an upper triangular matrix
    then $degf \leftarrow$ Print$(degf)$ with some comments
    else $degf \leftarrow$ Print$(degf)$ with some comments
return *degf*

8.5.6 `Solveindic4Size6by6` Function

The input of the function `Solveindic4Size6by6`$(degf)$ is a matrix $degf$ of size 6×6 as in Remark 8.5.1(vii). This function is to convert a matrix $degf$ to a strictly upper triangular matrix. It is the first case of size 6×6 where $f_{32} = 0$ and $b = [1, 1, 1]$. It runs the function `SwapRowsColumns` five times swapping rows and columns until $degf$ is upper triangular matrix. In fact the matrix $degf$ in the input of the $(n+1)^{st}$ run of the function `SwapRowsColumns` it will be the matrix $degf$ output by the n^{th}

run. It returns the matrix $degf$ after finishing all the replacements. The function works as follows:

SOLVEINDIC4SIZE6BY6($degf$)
$degf \leftarrow$ SWAPROWSCOLUMNS($degf, 1, 2$)
$degf \leftarrow$ SWAPROWSCOLUMNS($degf, 2, 6$)
$degf \leftarrow$ SWAPROWSCOLUMNS($degf, 3, 4$)
$degf \leftarrow$ SWAPROWSCOLUMNS($degf, 4, 5$)
$degf \leftarrow$ SWAPROWSCOLUMNS($degf, 3, 4$)
return *degf*

8.5.7 `Solveindic4Size6by6Above` Function

The input of the function `Solveindic4Size6by6Above`($degf$) is a matrix $degf$ of size $m \times m$ with $m \geq 6$ as in Remark 8.5.1(viii). `Solveindic4Size6by6Above` convertes the matrix $degf$ to an upper triangular matrix. It outputs a proof that M is solvable for this case. The function works as follows:

SOLVEINDIC4SIZE6BY6ABOVE($degf$)
$mysize \leftarrow$ SIZE($degf$)
$degf[1][2] = degf[3][2] = 0$
for i **in** $\{1, \ldots, mysize\}$
    do for j **in** $\{1, \ldots, mysize\}$
        if $j \geq i + 2$
            then $f[i][j] = 0$ $\triangleright$ since $\partial^2 = 0$ and R is an integral domain
if $mysize < 6$
    then return that $mysize$ must be greater than 6
    else
if $mysize = 6$
    then $degf \leftarrow$ SOLVEINDIC4SIZE6BY6($degf$)
    else
        if $mysize = 7$ or $mysize = 8$
            then $mycounter \leftarrow mysize - 6$
                $degf \leftarrow$ SOLVEINDIC4SIZE6BY6($degf$)
                for i **in** $\{1, \ldots, mycounter\}$
                    do if $i = 1$

```
            then degf ← SwapRowsColumns(degf, 4 + i, 6 + i)
                 degf ← SwapRowsColumns(degf, 3 + i, 4 + i)
                 degf ← SwapRowsColumns(degf, 1, 3 + i)
                 if i > 1
                    then degf ← SwapRowsColumns(degf, 4 + i, 6 + i)
                         degf ← SwapRowsColumns(degf, 3 + i, 4 + i)
                         degf ← SwapRowsColumns(degf, 1 + i, 3 + i)
                         degf ← SwapRowsColumns(degf, 1, 1 + i)
                         degf ← SwapRowsColumns(degf, 2, 1 + i)
if mysize ≥ 9
   then mycounter ← mysize − 6
degf ← Solveindic4Size6by6(degf)
for i in {1, . . . , mycounter}
      do if i = 1
            then degf ← SwapRowsColumns(degf, 4 + i, 6 + i)
                 degf ← SwapRowsColumns(degf, 3 + i, 4 + i)
                 degf ← SwapRowsColumns(degf, 1, 3 + i)
                 if i > 1
                    then degf ← SwapRowsColumns(degf, 4 + i, 6 + i)
                         degf ← SwapRowsColumns(degf, 3 + i, 4 + i)
                         degf ← SwapRowsColumns(degf, 1 + i, 3 + i)
                         degf ← SwapRowsColumns(degf, 1, 1 + i)
                         degf ← SwapRowsColumns(degf, 2, 1 + i)
degf ← StructuralCopy(degf)
mycounter1 ← mysize − 8
for mycounter2 in {1, . . . , mycounter1}
      do for i in {1, . . . , mycounter2}
            mycounter3 ← mycounter2 − i + 1
            degf ← SwapRowsColumns(degf, 2 + mycounter3, 3 + mycounter3)
if degf is not an upper triangular matrix
   then degf ← Print(degf) with some comments
   else degf ← Print(degf) with some comments
return degf
```

8.5.8 `Solveindic4Sizembym` Function

The input of the function `Solveindic4Sizembym`($degf$) is a matrix $degf$ of size $m \times m$ with $m \geq 3$ as in Remark 8.5.1(ix). It convertes the matrix $degf$ to an upper triangular matrix. The function outputs a proof that M is solvable for this case. The algorithm works as follows:

```
Solveindic4Sizembym(degf)
m ← Size(degf)
degf[2][1] = degf[2][3] = 0
for i in {1, ..., m}
      do for j in {1, ..., m}
            if j ≥ i + 2
               then f[i][j] = 0 ▷ since ∂² = 0 and R is an integral domain
2 ← i
m ← j
while i < j
      do degf ← SwapRowsColumns(degf, i, j)
i ← i + 1
j ← j − 1
if degf is an upper triangular matrix
   then degf ← Print(degf) with some comments
   else  degf ← Print(degf) with some comments
return degf
```

8.6 `SolvableModuleByUsualGradedWithProof` Function

The function `SolvableModuleByUsualGradedWithProof`(D, P) is called only if the conditions of Proposition 7.2.11 hold. The inputs of this function are the list of dimensions of the modules $D = [k_1, \ldots, k_n]$ where $dim(e_i) = k_i$ and the degree P of the differential on the module M. (The same inputs as the main function `IsSolvableModuleWithProof`.) `SolvableModuleByUsualGradedWithProof` outputs a proof that M is solvable. The algorithm works as follows:

```
SolvableModuleByUsualGraded(D, P)
m ← Size(D)
D[1] ← k1
0 ← j
dimf ← IdentityMat(m)
degf ← IdentityMat(m)
degf2 ← IdentityMat(m)
f ← IdentityMat(m)
for i in {1, ..., m}
     do D[j] ← dimej
          for i in {1, ..., m}
               do D[i] ← dimei
                    dimej − dimei − P ← dimf[i][j]
                                                  ▷ by definition
                    if dimf[i][j] < 0
                       then f[i][j] = 0           ▷ usual graded
          −dimf[i][j] ← degf[i][j]                ▷ by the properties
for j in {1, ..., m}
     do for i in {1, ..., m}
               do Rewrite f after setting some of its entries to zero
if f is an upper triangular matrix
   then for i in {1, ..., m}
               do 0 ← f[i][i]         ▷ since ∂² = 0 and R is an integral domain
                    Compute the matrix d of the differential ∂ with respect
                    to the basis S = {e_i}_{i=1}^m
   else  return f is not upper triangluar matrix
Construct a proof that M is solvable if f is an upper triangular matrix
return M is solvable
```

8.7 `IsSolvableModuleWithProof` Function

The function `IsSolvableModuleWithProof`(D, P) is the main function of our algorithm. It checks which of the conditions of the Propositions 7.2.4, 7.2.5, 7.2.8, 7.2.10, 7.2.11 and Remark 7.2.9 hold. Then it calls one of the functions: `Solveindic1WithProof`, `Solveindic2WithProof`, `Solveindic3WithProof`, `Solveindic4` and `Solva-`

bleModuleByUsualGradedWithProof according to the condition that matches the function. The inputs of this function are the list of dimensions of the modules $D = [k_1, \ldots, k_n]$ where $dim(e_i) = k_i$ and the degree P of the differential on the module M. The function outputs the dimension m of the vector of dimensions, the matrix $dimf$ of dimensions, the identity matrix f of size $m \times m$, the matrix $degf$ of degrees, the flags $indic$ and x_i; $i = 1, 2, 3$ to determine which of Solveindic(n) function to run. The algorithm works as follows:

```
IsSolvableModuleWithProof(D, P)
m ← Size(D)
if P = 1 or −1
   then return M is solvable (by Carlsson, 1983)
if P ≤ −2
   then k1 ← D[1]
        j ← 0
        dimf ← IdentityMat(m)
        degf ← IdentityMat(m)
        degf2 ← IdentityMat(m)
        f ← IdentityMat(m)
        for i in {2, ..., m}
            do j ← j + 1
               k2 ← D[i]
               diffk ← k1 − k2
               if k1 > k2
                  then t[j] ← diffk      ▷ t was empty
                       if diffk ≥ −P
                          then indic ← 1 ▷ indic was zero
                               x1 ← x1 + 1
                                  ▷ x1 was zero
                       elseif diffk < −P
                          then indic ← 2
                               x2 ← x2 + 1
                                  ▷ x2 was zero
                       elseif diffk < P
```

```
                    then indic ← 3
                         x3 ← x3 + 1
CHECK the conditions of the input of the two cases above
FOLLOWING the same strategy for indic = 1 and indic = 3
to construct indic = 2 if t_i + t_{i+1} ≤ −P and indic = 4
if t_i + t_{i+1} > −P
for j in {1, ..., m}
     do dimej ← D[j]
     for i in {1, ..., m}
     do dimei ← D[i]
        dimf[i][j] ← dimej − dimei + P
                              ▷ by definition
        if dimf[i][j] > 0
           then f[i][j] = 0            ▷ negative graded
      degf[i][j] ← −dimf[i][j]         ▷ by the properties
if indic = 1
   then CALL FUNCTION SOLVEINDIC1WITHPROOF
if indic = 2 or (indic = 4 and m = 2)
   then if m = 2
   then CALL FUNCTION SOLVEINDIC4SIZE2BY2
   else  CALL FUNCTION SOLVEINDIC2WITHPROOF
if indic = 3
   then CALL FUNCTION SOLVEINDIC3WITHPROOF
if indic = 4
   then CALL FUNCTION SOLVEINDIC4WITHPROOF
if indic = 1
   then return true
if indic = 2 and m ≠ 2
   then return fail
if indic = 3
   then return true
if indic = 4
   then return true
```

if $P \geq 2$ and the conditions of Proposition 7.2.11 are hold
    then CALL FUNCTION SOLVABLEMODULEBYUSUALGRADEDWITHPROOF
return *true*

We will give some examples for the function `IsSolvableModuleWithProof` as follows:

Example(1):

```
gap> C:=IsSolvableModuleWithProof([30,20,10],-3);
 diffk=10
 diffk=10
 indic=1
 dimf=[ [ -3, -13, -23 ], [ 0, -3, -13 ], [ 0, 0, -3 ] ]
 degf=[ [ 3, 13, 23 ], [ 0, 3, 13 ], [ 0, 0, 3 ] ]
 f=[ [ -3, -13, -23 ], [ 0, -3, -13 ], [ 0, 0, -3 ] ]
 Newf=[ [ 0, -13, -23 ], [ 0, 0, -13 ], [ 0, 0, 0 ] ]
 d=[ [ 0, "f12", "f13" ], [ 0, 0, "f23" ], [ 0, 0, 0 ] ],
 ( Since d^2=0 and R is an integral domain ).
Let C0=0 and C1=<e1> , C2=<e1,e2> , C3=<e1,e2,e3>
 C1/C0 is free, C2/C1 is free, C3/C2 is free
If x in C1, then x can be written uniquely as:
x=a1*e1
d(x)=a1*d(e1)
d(x)=a1(0)  in  C0
Hence d(C1) subset of C0 and then d(C1/C0)=0.
If x in C2, then x can be written uniquely as:
x=a1*e1+a2*e2
d(x)=a1*d(e1)+a2*d(e2)
d(x)=a1(0)+a2(f12*e1)  in  C1
Hence d(C2) subset of C1 and then d(C2/C1)=0.
If x in C3, then x can be written uniquely as:
x=a1*e1+a2*e2+a3*e3
d(x)=a1*d(e1)+a2*d(e2)+a3*d(e3)
d(x)=a1(0)+a2(f12*e1)+a3(f13*e1+f23*e2)  in  C2
Hence d(C3) subset of C2 and then d(C3/C2)=0.
Hence, 0=C0 subset of C1 subset of C2 subset of C3= M is
```

```
a composition series for M.
true
```

Example(2):

```
gap> C:=IsSolvableModuleWithProof([30,20,10],-30);
 diffk=10
 diffk=10
 indic=2
 dimf=[ [ -30, -40, -50 ], [ -20, -30, -40 ], [ -10, -20, -30 ] ]
 degf=[ [ 30, 40, 50 ], [ 20, 30, 40 ], [ 10, 20, 30 ] ]
 f=[ [ -30, -40, -50 ], [ -20, -30, -40 ], [ -10, -20, -30 ] ]
 d=[ [ "f11", "f12", "f13" ], [ "f21", "f22", "f23" ],
     [ "f31", "f32", "f33" ] ]
fail
```

Example(3):

```
gap> C:=IsSolvableModuleWithProof([-20,-10,-5],-3);
 diffk=-10
 diffk=-5
 indic=3
 dimf=[ [ -3, 0, 0 ], [ -13, -3, 0 ], [ -18, -8, -3 ] ]
 degf=[ [ 3, 0, 0 ], [ 13, 3, 0 ], [ 18, 8, 3 ] ]
 f=[ [ 0, 0, 0 ], [ -13, 0, 0 ], [ -18, -8, 0 ] ]
 Tranf=[ [ 0, -13, -18 ], [ 0, 0, -8 ], [ 0, 0, 0 ] ]
 d=[ [ 0, "f12", "f13" ], [ 0, 0, "f23" ], [ 0, 0, 0 ] ] ,
   ( Since d^2=0 and R is an integral domain ).
Let C0=0 and C1=<e1> , C2=<e1,e2> , C3=<e1,e2,e3>
 C1/C0 is free, C2/C1 is free, C3/C2 is free
If x in C1, then x can be written uniquely as:
x=a1*e1
d(x)=a1*d(e1)
d(x)=a1(0)  in  C0
Hence d(C1) subset of C0 and then d(C1/C0)=0.
If x in C2, then x can be written uniquely as:
x=a1*e1+a2*e2
```

```
d(x)=a1*d(e1)+a2*d(e2)
d(x)=a1(0)+a2(f12*e1)  in  C1
Hence d(C2) subset of C1 and then d(C2/C1)=0.
If x in C3, then x can be written uniquely as:
x=a1*e1+a2*e2+a3*e3
d(x)=a1*d(e1)+a2*d(e2)+a3*d(e3)
d(x)=a1(0)+a2(f12*e1)+a3(f13*e1+f23*e2)  in  C2
Hence d(C3) subset of C2 and then d(C3/C2)=0.
Hence, 0=C0 subset of C1 subset of C2 subset of C3= M is
a composition series for M.
true
```

Example(4):

```
gap> C:=IsSolvableModuleWithProof([40,30,20,10],-11);
 diffk=10
 diffk=10
 diffk=10
 indic=4
 dimf=[ [ -11, -21, -31, -41], [ -1, -11, -21, -31], [ 0, -1, -11,
          -21 ], [ 0, 0, -1, -11 ] ]
 degf=[ [ 11, 21, 31, 41 ], [ 1, 11, 21, 31 ], [ 0, 1, 11, 21 ],
        [ 0, 0, 1, 11 ] ]
 b=[ 0 ]
 i=1
 degf Original Case_after setting some elements to Zero is [[ 0, 21,
 31, 41 ], [ 1, 0, 21, 31 ], [ 0, 1, 0, 21 ], [ 0, 0, 0, 0 ] ]

 degf=[ [ 0, 1, 21, 31 ], [ 0, 0, 31, 41 ], [ 0, 0, 0, 21 ],
        [ 0, 0, 0, 0 ] ]
 Thus for the First case, degf is a strictly upper Triangular
 matrix, so M is solvable.

 degf=[ [ 0, 31, 21, 41 ], [ 0, 0, 1, 21 ], [ 0, 0, 0, 31 ],
        [ 0, 0, 0, 0 ] ]
 Thus for the second case, degf is a strictly upper triangular
```

```
 matrix, so M is solvable.

 b=[ 1 ]
 i=2
 degf Original Case_after setting some elements to Zero is [[ 0, 21,
 31, 41 ], [ 1, 0, 21, 31 ], [ 0, 1, 0, 0 ], [ 0, 0, 1, 0 ] ]

 degf=[ [ 0, 1, 31, 21 ], [ 0, 0, 41, 31 ], [ 0, 0, 0, 1 ],
        [ 0, 0, 0, 0 ] ]
 Thus for the First case, degf is a strictly upper Triangular
 matrix, so M is solvable.

 degf=[[ 0, 0, 0, 21 ],[ 0, 0, 1, 0],[ 0, 0, 0, 1 ],[ 0, 0, 0, 0 ]]
 Thus for the second case, degf is a strictly upper triangular
 matrix, so M is solvable.
true
```

Example(5):

```
gap> C:=IsSolvableModuleWithProof([10,20,30],7);
 diffk=10
 diffk=10
 dimf=[ [ 0, 3, 13 ], [ 0, 0, 3 ], [ 0, 0, 0 ] ]
 degree=[ [ 0, -3, -13 ], [ 0, 0, -3 ], [ 0, 0, 0 ] ]
 f=[ [ 0, 3, 13 ], [ 0, 0, 3 ], [ 0, 0, 0 ] ]
 d=[ [ 0, "f12", "f13" ], [ 0, 0, "f23" ], [ 0, 0, 0 ] ] ,
     ( Since d^2=0 and R is an integral domain ).
Let C0=0 and C1=<e1> , C2=<e1,e2> , C3=<e1,e2,e3>
 C1/C0 is free, C2/C1 is free, C3/C2 is free.
If x in C1, then x can be written uniquely as:
x=a1*e1
d(x)=a1*d(e1)
d(x)=a1(0)  in  C0
Hence d(C1) subset of C0 and then d(C1/C0)=0.
If x in C2, then x can be written uniquely as:
x=a1*e1+a2*e2
```

```
d(x)=a1*d(e1)+a2*d(e2)
d(x)=a1(0)+a2(f12*e1)  in  C1
Hence d(C2) subset of C1 and then d(C2/C1)=0.
If x in C3, then x can be written uniquely as:
x=a1*e1+a2*e2+a3*e3
d(x)=a1*d(e1)+a2*d(e2)+a3*d(e3)
d(x)=a1(0)+a2(f12*e1)+a3(f13*e1+f23*e2)  in  C2
Hence d(C3) subset of C2 and then d(C3/C2)=0.
Hence, 0=C0 subset of C1 subset of C2 subset of C3= M is
a composition series for M.
true
```

Appendix A

Appendix

A.1 Appendix to Chapter 2

In this appendix we will attached the codes for all the functions we have written and used in Chapter 2 as follows:

1. **IsSimpleGraph Function**

```
IsSimpleGraph:=function(V,E)
local i,j,M,sV,tempx,tempedgex,tempedgey;
##
############################################################################
##
## The input of this function is a finite simple graph zeta=(V,E), where V and
## E represents the list of vertices and the list of Edges respectively.
##
## It returns "true" if zeta is a simple graph. Otherwise, It returns an error message.
##############################################################################
##
sV:=Size(V);
M:= Length(E);
if V=[] then
      Error("The graph must be simple and not a null graph");
fi;
if IsList(V)=false then
      Error("V must be a list");
fi;
if IsList(E)=false then
     Error("E must be a graph");
fi;
for i in [1..sV] do
    if IsPosInt(V[i])=false then
        Error("The entries of V must be positive integers");
```

```
    fi;
od;
if ForAny(V, v-> [v,v] in E)=true then
       Error("The graph must be simple no loops");
fi;
if IsSubset(Cartesian(V,V),E)=false then
    Error(" Every edge [x,y] must be a pair of vertices and x,y belong to V");
fi;
for i in [1..M] do  # First loop through the list of edges E
    tempedgex:=SSortedList(E[i]);
    for j in [i+1..M] do  # Second loop through the edges E excluding the first entry of E
        tempedgey:=SSortedList(E[j]);
        if tempedgex=tempedgey then  # determine whether the specific edge
                                     # E[j] is equal to the edge tempedgex
            Error("The graph must be simple no multiple edges");
        fi;
    od;
od;
return(true);
end;
```

2. StarLinkDominateOfVertex Function

```
StarLinkDominateOfVertex:=function(V,E)
local i,j,x1,M,sV,sE,tempx,St,indx1,Lk,indx2,x,YY,Y1,Y2,tempedgex,tempedgey,L,sL,invV;
##
##############################################################################
##
## The input of this function is a finite simple graph zeta=(V,E), where V and
## E represents the list of vertices and the list of Edges respectively.
##
## It computes the star St(v) and link Lk(v) and concatenates them in two separate
## lists St and Lk respectively. Also it calculates a list Y(v), for each vertex
## v in V of those vertices u in V  such that u is less than v, and we call the
## list of all such Y(v), YY. In addition, it calculates sV, the size of the
## list of vertices V and M, the size of the list of edges E.
################################################################################
##
if IsSimpleGraph(V,E)=true then      # Call the function IsSimpleGraph to test
                                     # whether the graph zeta is simple or not
   sV:=Size(V);
   M:= Length(E);
   St:= NullMat(sV,1,0);
   Lk:= NullMat(sV,1,0);
   for i in [1..sV] do                 # loop through the vertices V
     tempx:=V[i];
     indx1:=1;                         # index for the star of specific vertex v.
     indx2:=0;                         # index for the link of specific vertex v.
     St[tempx][indx1]:=tempx;          # St: is a two dimensional matrix, the rows
                                       # indices represent the vertices and the columns
                                       # indices represent the star of a specific vertex.
```

```
      for j in [1..M] do                # loop through the edges E.
          if tempx=E[j][1] then         # This section to determine whether the specific
                                        # vertex E[j][1] is equal to the vertex tempx.
             if E[j][1]<>E[j][2] then   # excludes the isolated vertices from the calculation
                indx1:=indx1+1;
                indx2:=indx2+1;
                St[tempx][indx1]:=E[j][2];
                  # means that the vertex E[j][2] belongs to the star of a specific vertex v
                Lk[tempx][indx2]:=E[j][2];
                  # means that the vertex E[j][2] belongs to the link of a specific vertex v
             fi;
          fi;
          if tempx=E[j][2] then    # This section is the same of the first section,
                                   # above just we replaced the first coordinate of
                                   # the edge E(j) by the second coordinate.
             if E[j][1]<>E[j][2] then
                indx1:=indx1+1;
                indx2:=indx2+1;
                St[tempx][indx1]:=E[j][1];
                Lk[tempx][indx2]:=E[j][1];

             fi;
          fi;
       od;
    od;
    YY:=[];
    for i in [1..sV] do       # loop through the vertices V.
        Y1:=[];
        for j in [1..sV] do   # loop through the vertices V.
            Y2:=Set(St[j]);   # make the list of star of each vertex v as an order set
            RemoveSet(Y2,j);  # remove the vertex j from the set Y2.
    if IsSubsetSet(St[i],Y2) and j<>i then # computes a list Y(v), for each vertex v in V of
                                           # these vertices y in V such that u less than v.
        Add(Y1,j);   # Y1 represents a singleton list of Y(v) with respect to each vertex v
            fi;
        od;
        Add(YY,Y1);    # YY is a list which contains the lists of Y(v) for each,
                       # vertex v in V of these vertices u in V such that u less than v
    od;
    invV:=-V;
    L:=Concatenation(V,invV);
    sL:=Size(L);
else
    return("The graph must be a simple graph");
fi;
return([St,Lk,YY,sV,M,L,sL]);
end;
```

3. DeleteVerticesFromGraph Function

```
DeleteVerticesFromGraph:=function(St,V,E)
```

```
local NE,NV,h,v1,Ex,Vx,sStI,g,v,H1,H2,b,ExM,VxM,i,a,j,sNE,sNV,sV,M;
##
###############################################################################
##
## The input of this function are the list of stars St, the list of vertices V,
## and the list of edges E.
##
## It computes graphs zeta\St(v), for all v in V, with NV the list of all lists
## of vertices of zeta\St(v) and NE the list of all lists of edges of zeta\St(v).
###############################################################################
##
sV:=Size(V);
M:=Size(E);
NE:=[];
NV:=[];
for h in [1..sV] do       # loop through the vertices V
    v1:=St[h];            # represents star for each vertex v.
    Ex:=E;
    Vx:=V;
    sStI:=Size(v1);       # represents the size of star of each vertex v.

    for g in [1..sStI] do # loop through the elements of the star of each vertex v.
        v:=v1[g];  # v: represents the vertices which are belongs to each star v1=St[h],
                   # which we want to delete them from the graph zeta.
        H1:=[];
      H2:=[];
        b:=0;
        ExM:=Size(Ex);        # represents the size of the set of edges E.
        VxM:=Size(Vx);        # represents the size of the set of vertices V.
        for i in [1..ExM] do  # loop through the edges E.

            a:=0;
            for j in [1..2] do    # loop through inside each edge of the set of edges E.
               if Ex[i][j]=v then # determine whether v is in the list of star of each,
                                  # vertex which we wants to delete it from the graph zeta.
                  a:=1;
                  b:=1;
               fi;
            od;
            if a<>1 then  # means that the coordinates of the pair (edge) does not
                          # equal to v which we want to delete.
                 Add(H1,Ex[i]); # add that pair (edge) to list H1, which it will be the
                                # list E without those edges, which are contains vertex v.
            fi;
        od;
        Ex:=H1;
        for i in [1..VxM] do     # loop through the vertices V.
            a:=0;
            if Vx[i]=v then      # determine if this (Vx[i]) is equal to vertex v which
                                 # we need to delete.
                  a:=1;          # if yes make a=1
            fi;
            if a<>1 then         # means that this vertex is not equal to vertex v.
```

```
                Add(H2,Vx[i]);  # add this vertex to the list H2, which it will
                                # be the list of V\St[h]
            fi;
        od;
        Ex:=H1;
        Vx:=H2;
    od;
    Add(NE,H1);   # NE is the list of all lists of vertices of zeta\St(v).
    Add(NV,H2);   # NV is NE the list of all lists of edges of zeta\St(v).
od;
sNE:=Size(NE);
sNV:=Size(NV);
return([NV,NE,sNV,sNE]);
end;
```

4. ConnectedComponentsOfGraph Function

```
ConnectedComponentsOfGraph:=function(G1,G2)
local DFSVisit,i,j,u,e,N1,x1,y1,M,W,count,color,s,x2,D,k,sD,P,t,AllComps,sAllComps,
F,sF,Y1,sY1,C1,C2,Y2,Y3,L2,U2,q,sY3,Y4,L4,sY4,sG1,NonIsolatedComps,IsolatedComps;
##
#############################################################################
##
## The input of this function is the list of edges G of a graph B=(G1,G2),
## where G1 is the list of vertices and G2 is the list of edges.
##
## It returns [AllComps,sAllComps,NonIsolatedComps,D,IsolatedComps,F] where:
##
### AllComps: the list of all the connected components of the graph B,
### sAllComps: the size of AllComps,
### NonIsolatedComps: the list of all the non-isolated connected components
###                   of the graph B,
### D: the list of vertices of non-isolated connected components,
### IsolatedComps: the list of all the isolated connected components of the graph B,
### F: the list of vertices of isolated connected components.
##############################################################################
##
if IsList(G1)=false then
      return("G1 must be a list");
fi;
sG1:=Size(G1);
for i in [1..sG1] do
    if IsPosInt(G1[i])=false then
        return("The entries of G1 must be positive integers");
    fi;
od;
if IsList(G2)=false then
        return("G must be a graph");
fi;
if IsSubset(Cartesian(G1,G1),G2)=false then
     return(" Every edge [x,y] must be a pair of vertices and x,y belong to G1");
```

```
fi;
M:= Length(G2);
##
################################################################################
##
## DFSVisit implements the depth search algorithm to construct the
## connected components ( having more than one vertex ) of the graph B.
##
## The input to DfsVisit are:
### i: A vertex of graph B,
### W: the weight matrix of B,
### sD: the size of the vertex list of the graph B,
### count: is a specific number representing the vertices of each component,
### color: is a list of size sD with entries the numbers of
###        non-isolated components.
##
   DFSVisit:=function(i,W,sD,count,color)
   local j,s;
       for s in [1..sD] do
           if color[s]=0 and W[i][s]=1 then
               color[s]:=count;
               DFSVisit(s,W,sD,count,color);
           fi;
       od;
   end;;
##
################################################################################
##
## This section computes the list of vertices D of the non-isolated
## connected components of the graph B and its size sD.
##
e:=0;
u:=0;
D:= [];
for i in [1..M] do
    for t in [1..2] do
        u:=0;
        for j in [1..e] do
            if D[j]=G2[i][t] then
               u:=u+1;
            fi;
        od;
        if u=0 then
            e:=e+1;
            D[e]:=G2[i][t];
        fi;
    od;
od;
u:=0;
sD:=Size(D);
##
################################################################################
##
```

```
W:= NullMat(sD,sD,0);                      # Set W to be a null matrix of size sD x sD
count:=0;                                  # index for the number of connected components
color:= ListWithIdenticalEntries( sD, 0 );  # List "color" equal to null-vector of size sD.
s:=1;           #s^th item of color is the (number of the) component of B to which
                #the s^th vertex of B belongs (or is zero if s has not yet been processed).
for i in [1..M] do  # loop through the edges of the list G2
    for j in [1..sD] do         # loop through the list of vertices of D
        if D[j]=G2[i][1] then   # determine whether the vertex D[j] equal to G[i][1]
           x1:=j;
        fi;
    od;
    for j in [1..sD] do
        if D[j]=G2[i][2] then  # determine whether the vertex D[j] equal to G[i][2]
           y1:=j;
        fi;
    od;
    W[x1][y1]:=1;    # construct the adjacency matrix of the graph B as that:
    W[y1][x1]:=1;    # if W[x1][y1]= 1 and W[y1][x1]=1 then it means that
                     # the vertex W[x1][y1] join with the vertex W[y1][x1]
                     # otherwise W[x1][y1] and W[y1][x1] are disjoined.
od;
for i in [1..sD] do
    if color[i]=0 then   # determine whether we are done with the vertices in
                         # the same component
        count:= count+1; # we give another number for the next component
        color[i]:=count;
        DFSVisit(i,W,sD,count,color);
    fi;
od;
P:=[];
NonIsolatedComps:=[];
for k in [1..count] do      # loop through the number of connected components k
    for i in [1..sD] do     # loop through the list of vertices D
        if k=color[i] then  # determine whether these vertices k  have the same
                            # number of connected component.
            Add(P,D[i]);    # Adding the vertices D(i) which are in the same
                            # connected component to the list P.
        fi;
    od;
    for i in [1..sD] do
        if k=color[i] then
             Add(P,-D[i]);  # Adding the inverses of D(i) to the list P
        fi;
    od;
    Add(NonIsolatedComps,P);   # NonIsolatedComps: the list of all the
                               # non-isolated  components of the graph B
    P:=[];
od;
##
#################################################################
##
## In this section we compute the isolated connected components of
## the graph B and add them to the list Comps
```

```
##
IsolatedComps:=[];
F:=Difference(G1,D);
sF:=Size(F);
if sF<>0 then
    for i in [1..sF] do
        Add(IsolatedComps, [F[i],-F[i]]); # IsolatedComps: the list of all the
                                          # non-isolated  components of the graph
    od;
fi;

AllComps:=Concatenation(NonIsolatedComps,IsolatedComps); # the list of all the
                                                         # components of the graph B
sAllComps:=Size(AllComps);

##
############################################################################
##
return([AllComps,sAllComps,NonIsolatedComps,D,IsolatedComps,F]);
end;
```

5. WhiteheadAutomorphismsOfSecondType Function

```
WhiteheadAutomorphismsOfSecondType:=function(NV,NE,St,YY)
local i,j,gens2,gens,genss,Bs,MV,ME,sME,h,G1,G2,R3,Comps,sComps,sMV,sNE,UniA,
D,DD,sD,S,YYY,NYY,invNYY,DYY,sDYY,Ls,t,xn,union_element,AQ,sAQ,L3,sL3,L4,sL4,sAQ1,
L5,elms,diff,Combs1,NCombs,sNCombs,Combs2,q,L7,k,set,AA1,AA,sAA,A,sA,T,sT;
##
###########################################################################
##
## The input of this function are:
### the lists of vertices NV of the subgraph zeta\St(v)
### the list of edges NE of the subgraphs zeta\St(v)=(NV(v),NE(v)) for all v in V
### the list of stars St(v)
### list YY for each vertex v in V of these vertices u in V such that u less than v.
##
## It computes the list A of type(2) Whitehead automorphisms which forms
## the first part of the set of generators of Aut(G_zeta). Also it computes
## a list T of names of elements of A (the i^th element of T is the name of
## the i^th element of A).
############################################################################
##
gens2:=[];
gens:=[];
genss:=[];
AA:=[];
Bs:=[];
MV:=NV;
sNE:=Size(NE);
for h in [1..sNE]do      #loop through the list NE
      G1:=NV[h];
```

```
G2:=NE[h];
R3:=ConnectedComponentsOfGraph(G1,G2);
                       # computes the list of the Connected components
                       # for each subgraph (NV(h),NE(h))
Comps:=R3[3];          # Comps: list of non-isolated components of the subgraph
sComps:=Size(Comps);   # sComps: size of Comps
D:=R3[4];              # D: the list of vertices of non-isolated components
sD:=Size(D);           # sD: size of D
S:=St[h];     # S is the list of the star of the vertex h
YYY:=YY;      # YYY is a list which contains the lists of Y(v),for each vertex
              # v in V of these vertices u in V such that u less than v
NYY:=YYY[h];  # YYY is the dominate list Of the vertex h
invNYY:=-NYY; # the inverse of NYY
DYY:=Concatenation(NYY,invNYY);
sDYY:=Size(DYY);
Ls:=[[]];
for t in [1..sDYY] do  # loop through the list DYY
      xn:=DYY[t];
      union_element:=function(Ls,xn,S)
                      # Call the function union-element to construct a list
                      # called Ls of all subsets of St(v) + YY(v) + (-YY(v))
      local J,i,j,sLs;
      sLs:=Size(Ls);
      for i in [1..sLs] do
          J:=StructuralCopy(Ls[i]); # to make a structural copy  of each object Ls[i]
          if not(-xn in J) or (not(xn in S) and not(-xn in S))then
              Add(J,xn);
              Add(Ls,J); # Ls is the list of all subsets of St(v) + YY(v) + (-YY(v))
          fi;
      od;
      end;;

      union_element(Ls,xn,S);
od;
AQ:=Ls;
sAQ:=Size(AQ);
L4:=[];
L3:=[];
if sComps=0 then            # determine whether the list Comps
                            # doesn't has any connected component
    for j in [1..sAQ] do    # loop through the list Ls
        sAQ1:=Size(AQ[j]);
        if sAQ1 <> 0 then
            Add(L3,AQ[j]); # add each list (subsets) AQ(j) of AQ to new list L3
        fi;
    od;
    sMV:=Size(MV[h]); # sMV is the size of the vertex list of the subgraph (MV[h],ME[h])
    ################################################################################

    ## For any element X not in D and sMV > 1 and X<>YY[h] we add the [X], [-X] and
    ## [X,-X] to L3 (since these elements are part of isolated components)

    for j in [1..sMV] do # loop through the vertex list of the subgraph (MV[h],ME[h])
```

```
      if not (MV[h][j] in D) and sMV<>1 and MV[h]<>YY[h] then
            Add(L3,[MV[h][j]]);
            Add(L3,[-MV[h][j]]);
            Add(L3,[MV[h][j],-MV[h][j]]);
      fi;
od;
################################################################################
sL3:=Size(L3);
for k in [1..sL3] do  # loop through list L3
     Add(L3[k],h);    # we add the vertex h to each list of L3 and
     Add(L4,L3[k]);   # we add the new list L3(k) to the list L4
od;
set:=L4;
sL4:=Size(L4);
L5:=[];
################################################################################
## In this part we delete the vertex h from each list set(i) and in the same
## time we add its inverse (-h) to the list diff, then we add the new list diff
## to the list L5
for i in [1..sL4] do
      elms:=[h];;
      diff:=Difference(set[i],elms );;
      Add(diff,-h);
      Add(L5,diff);
od;
################################################################################
fi;
L3:=[];
if sComps=1 then                                 # determine whether the list Comps
                                                 # has just one connected component
     for i in [1..sComps] do                     # loop through the list Comps
          for k in [1..sAQ] do                   # loop through the list AQ
               UniA:=Union( [ AQ[k] , Comps[i]]); # we make union for this component
                                                 # with each list of of the list AQ
               Add(L3, UniA);
          od;
     od;
     sMV:=Size(MV[h]);

     for j in [1..sMV] do   ## See the previous comments on this section
           if not (MV[h][j] in D) and sMV<>1 and MV[h]<>YY[h] then
               Add(L3,[MV[h][j]]);
               Add(L3,[-MV[h][j]]);
               Add(L3,[MV[h][j],-MV[h][j]]);
           fi;
     od;
     sL3:=Size(L3);
     for k in [1..sL3] do
          Add(L3[k],h);
          Add(L4,L3[k]);
     od;
     set:=L4;
```

```
     sL4:=Size(L4);
     L5:=[];
     for i in [1..sL4] do
           elms:=[h];;
           diff:=Difference(set[i],elms );;
           Add(diff,-h);
           Add(L5,diff);
     od;
fi;
L3:=[];
if sComps >=2 then                    # determine whether the list Comps
                                      # has more than one connected component
     Combs1:=Combinations(Comps);     # Combs1 is the list of all subsets of Comps
                                      # including the empty set and Comps itself
     NCombs:=Difference(Combs1,[[]]); # we removed the empty set from Combs1
     sNCombs:=Size(NCombs);
     Combs2:=[];
     for q in [1..sNCombs] do            # loop through the elements of NCombs
           L7:=Concatenation(NCombs[q]); # to remove the extra brackets
           Add(Combs2,L7);
     od;
     for k in [1..sAQ] do              # loop through the elements of AQ
           for i in [1..sNCombs] do    # loop through the elements of NCombs
                 UniA:=Union([AQ[k] ,Combs2[i]]);
                 Add(L3, UniA);
           od;
     od;
     sMV:=Size(MV[h]);
     for j in [1..sMV] do # See the previous comments on this section
           if not (MV[h][j] in D) and sMV<>1 and MV[h]<>YY[h] then
               Add(L3,[MV[h][j]]);
               Add(L3,[-MV[h][j]]);
               Add(L3,[MV[h][j],-MV[h][j]]);
           fi;
     od;
     sL3:=Size(L3);
     for k in [1..sL3] do
         Add(L3[k],h);
         Add(L4,L3[k]);
     od;
    set:=L4;
    sL4:=Size(L4);
    L5:=[];
    for i in [1..sL4] do
          elms:=[h];;
          diff:=Difference(set[i],elms );;
          Add(diff,-h);
          Add(L5,diff);
    od;
fi;
for i in [1..sL4] do # loop through the elements of L4 and L5 in the same time
      AA1:=[];
      Add(AA1,L4[i]);
```

```
            Add(AA1,h);      # we forms type(2) Whitehead automorphisms
                             # with positive operator (h)
            Add(AA,AA1);
            AA1:=[];
            Add(AA1,L5[i]);
            Add(AA1,-h);    # we forms type(2) Whitehead automorphisms
                            # with negative operator (-h)
            Add(AA,AA1);    # AA forms the type(2) Whitehead automorphisms which are
                            # the first part of the generators of the automorphisms
      od;                   # of group of partially commutative group
od;
sAA:=Size(AA);
A:=[];
for i in [1..sAA] do          # loop through the generators set AA
    if not (AA[i] in A) then  # it helps us to rewrite the list AA without repetition
        Add(A,AA[i]);         # The elements of list A are the definitions of Type(2)
                              # Whitehead automorphisms of the generators of the
                              # presentation of Aut(G_zeta)
     fi;
od;
sA:=Size(A);
T:=[];
for i in [1..sA]do
    Add(T,Concatenation(["A",String(i)])); # Compute the list T with i^th entry A(i) where
                                           # A(i) is the name of the i^th element of A
od;
sT:=Size(T);
return ([A,T,sA]);
end;
```

6. WhiteheadAutomorphismsOfFirstType Function

```
WhiteheadAutomorphismsOfFirstType:= function(E,sV,sA,T)
local gens2,gens,genss,E1,GraphAutomorphismGroup,Gr,HH,KK,rels1,HHH,srels1,
NJK,F,sF,Gens3,i,NF1,relvalofF,srelvalofF,I1,Gens2,I2,J1,sGens2,Gens,sGens,
sgenss,sgens,zz,rels2,srels2,Rels1,sRels1;
##
############################################################################
##
## The input of this function are:
### the list of edges E
### the size of the list of vertices sV
### the size of the list A  of type(2) Whitehead automorphism of Aut(G_zeta)
### the list T  with i^th entry A(i), where A(i) is the name of the i^th element of A.
##
## It computes the list Gens of the type(1) Whitehead automorphisms which forms
## the second part of the set of generators of the automorphism group of G_zeta,
## and then computes the list of the generators gens of Aut(G_zeta) with its
## size sgens. The subgroup Aut_zeta(G_zeta) of Aut(G_zeta) consists of graph
## automorphism: that is, elements pi in Aut(G_zeta) such that  pi restrict
## to the graph zeta is a graph automorhism.
```

```
##############################################################################
##
gens2:=[];
gens:=[];
genss:=[];
E1:=E;
##############################################################################
##
## The purpose of this section is to compute the group of the graph with the size
## of vertices sV since the permutation on V is an automorphism of the graph zeta
##
GraphAutomorphismGroup := function(E1)
return SubgroupProperty(SymmetricGroup(sV),g -> Set(E1,k->OnSets(k,g)) = Set(E1));
end;
##
##############################################################################
##
Gr:=GraphAutomorphismGroup(E);
HH:=AsGroup(Gr);
KK:=IsomorphismFpGroupByGenerators(HH,GeneratorsOfGroup(HH));
                                # returns an isomorphism from the given finite group
                                # HH to a finitely presented group isomorphic to HH.
HHH:=Image(KK);                 # Call Image the function which computes a finitely
                                # presented group H on the chosen generators KK
rels1:=[];
Rels1:=[];
rels2:=RelatorsOfFpGroup(HHH); # rels2: relators set of the group automorphism of graph
srels2:=Size(rels2);
F:= GeneratorsOfGroup(HHH);    # F: generators set of the group automorphism of graph
sF:=Size(F);
for i in [1..srels2] do
      zz:=ExtRepOfObj(rels2[i]);
                     # The function ExtRepOfObj() helps us to rewrite each
                     # single relation as a vector with entires are the indces
                     # and the power of the generators which are form that relation.
                     # For example the result of ExtRepOfObj(A52*A4*A52^-1*A4^-1)
                     # is the vector [ 52, 1, 4, 1, 52, -1, 4, -1 ]
      Add(Rels1,zz);
od;
sRels1:=Size(Rels1);
Gens3:=[];
for i in [1..sF] do
      NF1:=Concatenation(["f",String(i)]);
      Add(Gens3,NF1);   # Gns3 is the first part of type(1) Whitehead automorphism
                        # which are the same F just we rewrite them to make them
                        # suitable with the other generators
od;
relvalofF:= GeneratorsOfGroup(HH); # Compute list of the definitions relvalofF of
                                   # the generators Gens3 of the group of graph HH
srelvalofF:=Size(relvalofF);
I1:=[];
Gens2:=[];
for i in [1..sV] do
```

```
      I2:=Concatenation(["A",String(sA+i),"(",String(i),")","=",String(-i)]);
               # Make a list, called I2, of type(1) Whitehead automorphisms which
               # send a generator to its inverse and add it to the leist I1
      Add(I1,I2);
      J1:=Concatenation(["A",String(sA+i)]);
                               # rewrite the elements of I1 as a string to make
                               # them compatible with the other generators and
                               # add them to Gens2
      Add(Gens2,J1);
od;
sGens2:=Size(Gens2);
Gens:=Concatenation(Gens2,Gens3);
                    # Concatenate the lists Gens2 and Gens3 in a new list called
                    # Gens which represents all type(1) Whitehead automorphisms
sGens:=Size(Gens);
for i in [1..sGens] do
     Add(gens,Gens[i]);
od;
genss:=Concatenation(T,Gens2);
                   # Concatenate the two lists T and Gens2 in a one list called
                   # genss. The list genss helps to form the relations later
gens:=Concatenation(T,Gens); # Compute set of the generators gens of Aut(G_zeta),
                             # by concatenating the two lists T and Gens.
sgenss:=Size(genss);
sgens:=Size(gens);
return([gens,sgens,sgenss,Gens3,relvalofF,srelvalofF,Rels1,sRels1,sGens2]);
end;
```

7. RelationsOfGraphAutomorphisms Function

```
RelationsOfGraphAutomorphisms:= function(sA,sgenss,relvalofF,sV,sGens2)
local rels,Rels,i,j,R6,FF,srelvalofF,d,F1,PP,R7,R11,idx1,idx2,idx3,srels,sRels;
##
#############################################################################
##
## The input of this function are:
### the size sA of the list A of definition of the second type of generator,
### the size of the list genss defined in WhiteheadAutomorphismsOfFirstType.g,
### the list of generators of the graph automorphism relvalofF defined in,
### WhiteheadAutomorphismsOfFirstType.g,
### sizes sV, and sGens2 of the lists V and Gens2 respectively.
##
## It computes the row matrix of indices Rels of the generators
## which forms the relations of this type, that related to the
## graph automorphism with its size sRels.
##############################################################################
##
Rels:=[];
for i in [sA+1..sgenss] do   # loop through the generators Gens2
      Add(Rels,[1,i]);   # [1,i] is the row matrix of indices of each relation
                         # of type R11={A^2=1 : A in Gens2} and add it to the
```

```
                        # list Rels. The first entry 1 is just a flag to let
                        # us know that here the generator is of power two
od;
for i in [sA+1..sgenss] do   # loop through the generators Gens2
   for j in [sA+1..sgenss] do
      if i<>j then
         Add(Rels,[0,-i,-j,i,j]);
                     # [0,-i,-j,i,j] is the row matrix of indices of
                     # each relation of type ( g^-1*h^-1*g*h ) such that
                     # g,h in Gens2. The first entry 0 is just a flag to
                     # let us know that here the generator without any power
        fi;
    od;
od;
FF:=[];
srelvalofF:=Size(relvalofF);
for i in [1..srelvalofF] do   # loop through the generators relvalofF
                              # of the group of graph HH
       d:=relvalofF[i];
       F1:=d^-1;              # computes F1 the inverse of each element of
                              # relvalofF and add them to the list FF
      Add(FF,F1);
od;
##################################################################################
##
## In this section we apply the function PP to (j, sigma(i)) to return the value
## sigma(i) for each i in the list of vertices { 1, ..., n } and sigma in the list
## FF above. Using these values form the relations R7: that is compute the row
## matrix  of indices [0,-idx1,idx2,idx1,idx3], for each such relation,  and
## add it to the list Rels.
##
for i in [1..srelvalofF] do   # loop through the generators relvalofF
    for j in [1..sV] do       # loop through the vertex set V
          PP:=OnPoints( j,FF[i]);
          idx1:=i+sA+sGens2;
          idx2:=sA+j;
          idx3:=sA+PP;
          Add(Rels,[0,-idx1,idx2,idx1,idx3]);
                                 # means that the idx1 refers to the location
                                 # of F in the original F Matrix
    od;
od;
##################################################################################
sRels:=Size(Rels);
return([Rels,sRels]);
end;
```

8. APCGRelationR1 Function

```
APCGRelationR1:=function(sV,A,T,Rels)
local k,j,i,diff1,UA,UAiff,R1,XX1,XX2,idx1,idx2,t,sA,srels,sRels;
```

```
##
################################################################################
##
## The input of this function are:
### sV: the size of the list of vertices of the graph zeta,
### A : the list of type(2) Whitehead automorphisms of Aut(G_zeta),
### T : the list of names of elements of A,
### Rels: the list of row matrices of indices of relations (it is one
### of the outputs of "APCGRelationR5".
### Note that in order to get just the row matrices of indices of relation (R1)
### we need to pass an empty list [] rather than the list Rels above.
##
## It computes the list of indices [0,idx1,idx2] of relators of type (R1)
## of the group Aut(G_zeta) and adds them  to the list Rels. In addition
## it calculates the size of the list Rels.
## It returns [Rels,sRels].
################################################################################
##
sA:=Size(A);
for k in [1..sV] do   # loop through the list of vertices V
  for i in [1..sA]do  # loop through the list A defined above

    if k in A[i][1] and not (-k in A[i][1]) and A[i][2]=k then

                            # Here we have satisfied the conditions,
                            # of the  Whitehead automorphism (A,a),
                            # "a" is called the multiplier

       diff1:=Difference(A[i][1],[k]); # we delete the multiplier k from each subset,
                                       # A[i][1] and add its inverse -k to this subset.
       Add(diff1,-k);
       for j in [1..sA]do          # loop through the list A defined above.
         UA:=SSortedList(A[j][1]); # Sorted lists A[j][1] to satisfy the conditions of (R1)
        UAiff:=SSortedList(diff1); # Sorted lists diff1 to satisfy the conditions of (R1)
           if UA=UAiff and A[j][2]=-k then   # Verify the inverse of each (A,a)
              ################################################################################
              ##
              ## In this section we compute the list of indices [0,idx1,idx2] of relators of
              ## type (R1) and add them to the list Rels. Note that 0 is just flag to let us
              ## know that all the generators here of power 1. idx1 represents the index of a
              ## specific generator A(i). idx2 represents the index of the inverse of A(j).
              ## For example if [0,idx1,idx2]= [0, 1, 2] then this means A1*A2=1.

              XX1:=Concatenation(["A",String(i)]);
                          # XX1: represents a specific Whitehead automorphism (A,a) of A
              XX2:=Concatenation(["A",String(j)]);
                          # XX2: represents a specific Whitehead automorphism (A,a^-1)
                          #      which is the inverse of (A,a)
              idx1:=0;
              idx2:=0;
              for t in [1..sA] do   # Verify the indices of the given Whitehead
                                    # automorphisms A(i) and A(j) in A
                  if XX1=T[t] then
```

```
                    idx1:=t;
                 fi;
                 if XX2=T[t] then
                    idx2:=t;
                 fi;
              od;
              Add(Rels,[0,idx1,idx2]);
              ##
              ##############################################################################
              ##
           fi;
        od;
      fi;
   od;
od;
sRels:=Size(Rels);

return([Rels,sRels]);
end;
```

9. APCGRelationR2 Function

```
APCGRelationR2:=function(A,T,Rels,St)
local k,j,i,IntA,UniA,NUniA,l,K,t,UA,R2,XX1,XX2,XX3,idx1,idx2,idx3,t1,
sV,sA,R2a,K1,R2b,R2c,srels,sRels;
##
#############################################################################
##
## The input of this function are:
### A: the list of type(2) generators computed in "WhiteheadAutomorphismsOfSecondType",
### T: list of the names of elements of A,
### Rels: the list of row matrices of indices of the relations (it is one
### of the outputs of the "APCGRelationR1",
### St: the list of stars computed in "StarLinkDominateOfVertex".
### Note that in order to get just the row matrices of indices of relation (R2)
### we need to pass an empty list [] rather than the list Rels above.
##
## It computes the list of indices of the generators [0,idx1,idx2,-idx3] of
## relators of type (R2) of the group Aut(G_zeta) and adds them  to the list
## Rels. In addition it calculates the size of the list Rels.
## It returns [Rels,sRels].
##############################################################################
##
sV:=Size(St); #Since the size of stars list equal to sV, the size of the vertex list
sA:=Size(A);
for k in [1..sV] do          # loop through the vertex list V
    for i in [1..sA]do       # loop through the list A defined above
        for j in [1..sA] do  # loop through the list A defined above
            IntA:=Intersection( [ A[i][1] , A[j][1] ] );
            UniA:=Union( [ A[i][1] , A[j][1] ] );
            NUniA:=[];
```

```
for l in St[k] do # In this loop if the vertex l and its inverse -l in the
                  # same time are belong to the list UniA then we delete
                  # them, because they will cancel each other.
    if l in UniA and -l in UniA then
       NUniA:=Difference(UniA,[-l,l]);
       UniA:=NUniA;
    fi;
od;
K:=[k];
if IntA=K and k in A[i][1] and not (-k in A[i][1]) and A[i][2]=k and k in A[j][1]
  and  not (-k in A[j][1]) and A[j][2]=k and k in UniA and not (-k in UniA) then
      ########################################################################
      ##
      ## Section(1): We compute the first part of the list of indices
      ## [0,idx1,idx2,-idx3] of relators of type (R2) and add them to the list
      ## Rels. Note that 0 is just flag to let us know that all generators here
      ## of power 1. idx1: represents the index of the generator A(i).
      ## idx2: represents the index of the generator A(j). -idx3: represents
      ## the index of the inverse of the  generator A(t).
      ## For example if [0,idx1,idx2,-idx3]= [0,1,3,-5],
      ## then this means A1*A3*A5^-1=1.
      ##
      for t in [1..sA]do
      UA:=SSortedList(A[t][1]);
      if A[t][2]=k then
         if UA=UniA or UA=NUniA then
            XX1:=Concatenation(["A",String(i)]);
                   # XX1: represents a specific Whitehead automorphism (A,a) of A
            XX2:=Concatenation(["A",String(j)]);
                   # XX2: represents a specific Whitehead automorphism (B,a) of A
            XX3:=Concatenation(["A",String(t)]);
                   # XX3: represents a specific Whitehead automorphism (A+B,a^-1)
                   #      which is the inverse of (A+B,a) of A
            idx1:=0;
            idx2:=0;
            idx3:=0;
            for t1 in [1..sA] do
                       # Verify the indices of the given Whitehead automorphisms
                       # A(i), A(j) and the inverse of A(t) in A
                if XX1=T[t1] then
                   idx1:=t1;
                fi;
                if XX2=T[t1] then
                   idx2:=t1;
                fi;
                if XX3=T[t1] then
                   idx3:=t1;
                fi;
            od;
            Add(Rels,[0,idx1,idx2,-idx3]);
         fi;
       fi;
   od;
```

```
   ##
   ############################################################################
   ##
   ############################################################################
   ##
   ## Section(2): Note that in some cases when we delete the vertices l and
   ## its inverse -l from the list UniA=A+B we will get a new list NUniA=[k],
   ## but this is just the identity. So we will ignore this list (subset)
   ## and we compute the second part of the list of indices [0,idx1,idx2]
   ##
   if NUniA=K then
       XX1:=Concatenation(["A",String(i)]);
              # XX1: represents a specific Whitehead automorphism (A,a) of A
       XX2:=Concatenation(["A",String(j)]);
              # XX2: represents a specific Whitehead automorphism (B,a) of A
       idx1:=0;
       idx2:=0;
       for t1 in [1..sA] do
                  # Verify the indices of the given Whitehead automorphisms
                  # A(i) and A(j) in A
           if XX1=T[t1] then
               idx1:=t1;
           fi;
           if XX2=T[t1] then
               idx2:=t1;
           fi;
        od;
        Add(Rels,[0,idx1,idx2]);
   fi;
   ##
   ############################################################################
   ##
fi;
K1:=[-k];
if IntA=K1 and -k in A[i][1] and not (k in A[i][1]) and A[i][2]=-k and -k in A[j][1]
   and not (k in A[j][1]) and A[j][2]=-k and -k in UniA and not (k in UniA)  then
    ############################################################################
    ##
    ## Section(3): we compute the third part of the list of indices [0,idx1,
    ## idx2,-idx3] of relators of type (R2). It is the same of Section(1), just
    ## we switch the  multiplier "a" (k in this code) of the Whitehead
    ## automorphism (A,a) by its inverse "a^-1" (-k in this code).
    ##
    for t in [1..sA]do
        UA:=SSortedList(A[t][1]);
        if A[t][2]=-k then
           if UA=UniA or UA=NUniA then
              XX1:=Concatenation(["A",String(i)]);
              XX2:=Concatenation(["A",String(j)]);
              XX3:=Concatenation(["A",String(t)]);
              idx1:=0;
              idx2:=0;
```

```
                    idx3:=0;
                    for t1 in [1..sA] do
                        if XX1=T[t1] then
                            idx1:=t1;
                        fi;
                        if XX2=T[t1] then
                           idx2:=t1;
                        fi;
                        if XX3=T[t1] then
                           idx3:=t1;
                        fi;
                    od;
                    Add(Rels,[0,idx1,idx2,-idx3]);
                fi;
            fi;
        od;
        ##
        ###########################################################################
        ##
        ###########################################################################
        ##
        ## We compute the fourth part of the list of indices [0,idx1,idx2] of
        ## relators of type (R2). It is the same of Section(2), just we switch the
        ## multiplier "a" (k in this code) of the Whitehead automorphism (A,a) by
        ## its inverse "a^-1" (-k in this code).
        ##
        if NUniA=K1 then
            XX1:=Concatenation(["A",String(i)]);
            XX2:=Concatenation(["A",String(j)]);
            idx1:=0;
            idx2:=0;
            for t1 in [1..sA] do
                 if XX1=T[t1] then
                     idx1:=t1;
                 fi;
                 if XX2=T[t1] then
                     idx2:=t1;
                 fi;
            od;
            Add(Rels,[0,idx1,idx2]);
         fi;
         ##
         ##########################################################################
         ##
    fi;
  od;
 od;
od;
sRels:=Size(Rels);
return([Rels,sRels]);
end;
```

10. APCGRelationR3 Function

```
APCGRelationR3:=function(A,T,Lk,Rels)
local k,j,i,sV,sA,IntA,UniA,NUniA,l,K,t,UA,R2,XX1,XX2,idx1,idx2,
t1,R3a,R3a1,K1,R3b,R3b1,srels,sRels;
##
#############################################################################
##
## The input of this function are:
### A: the list of type(2) generators computed in "WhiteheadAutomorphismsOfSecondType",
### T: list of the names of elements of A,
### Lk: the list of links computed in "StarLinkDominateOfVertex".
### Rels: the list of row matrices of indices of the relations (it is one
### of the outputs of the "APCGRelationR2",
### Note that in order to get just the row matrices of indices of relation (R3)
### we need to pass an empty list [] rather than the list Rels above.
##
## It computes the list of indices of the generators [0,idx1,idx2,-idx1,-idx2]
## of relators of types (R3a) and (R3b) of the group Aut(G_zeta) and adds them
## to the list Rels. In addition it calculates the size of the list Rels.
## It returns [Rels,sRels].
#############################################################################
##
sV:=Size(Lk);
sA:=Size(A);
#############################################################################
##
## In this section we compute the list of indices [0,idx1,idx2,-idx1,-idx2] of
## relators of type (R3a) by satisfying the conditions of this relations and add
## them to the list Rels. Note that 0 is just flag to let us know that all the
## generators here of power 1. idx1: represents the index of the generator A(i).
## idx2: represents the index of the generator A(j). -idx1: means the inverse of A(i).
## -idx2: means the inverse of A(j).
## For example if [0,idx1,idx2,-idx1,-idx2]= [ 0, 9, 3, -9, -3 ], then this means
## A9*A3*A9^-1*A3^-1=1.
##
for k in [1..sV] do              # loop through the vertex list V
    for l in [1..sV] do          # loop through the vertex list V
        for i in [1..sA]do       # loop through A the Type (2) Whitehead Automorphisms
            for j in [1..sA] do  # loop through A the Type (2) Whitehead Automorphisms
                IntA:=Intersection( [ A[i][1] , A[j][1] ] );
                if l in A[i][1] and not (-l in A[i][1]) and A[i][2]=l and k in A[j][1]
                    and not (-k in A[j][1]) and A[j][2]=k and not (k in A[i][1]) and
                    not (-k in A[i][1]) and not (l in A[j][1]) and not(-l in A[j][1])
                    and IntA=[] then
                    XX1:=Concatenation(["A",String(i)]);
                        # XX1: represents a specific Whitehead automorphism (A,a) of A
                    XX2:=Concatenation(["A",String(j)]);
                        # XX2: represents a specific Whitehead automorphism (B,b) of A
                    idx1:=0;
                    idx2:=0;
                    for t in [1..sA] do  # Verify the indices of the given Whitehead
                                         # automorphisms A(i) and A(j) in A
```

```
          if XX1=T[t] then
             idx1:=t;
          fi;
          if XX2=T[t] then
              idx2:=t;
          fi;
    od;
    Add(Rels,[0,idx1,idx2,-idx1,-idx2]);
fi;
if l in A[i][1] and not (-l in A[i][1]) and A[i][2]=l and -k in A[j][1]
   and not (k in A[j][1]) and A[j][2]=-k and not (k in A[i][1]) and
   not (-k in A[i][1]) and not (l in A[j][1]) and not(-l in A[j][1])
   and IntA=[] then
    XX1:=Concatenation(["A",String(i)]);
        # XX1: represents a specific Whitehead automorphism (A,a) of A
    XX2:=Concatenation(["A",String(j)]);
        # XX2: represents a specific Whitehead automorphism (B,b) of A
    idx1:=0;
    idx2:=0;
    for t in [1..sA] do  # Verify the indices of the given Whitehead
                         # automorphisms A(i) and A(j) in A
          if XX1=T[t] then
              idx1:=t;
          fi;
          if XX2=T[t] then
              idx2:=t;
          fi;
    od;
    Add(Rels,[0,idx1,idx2,-idx1,-idx2]);
fi;
if -l in A[i][1] and not (l in A[i][1]) and A[i][2]=-l and k in A[j][1]
    and not (-k in A[j][1]) and A[j][2]=k and not (k in A[i][1]) and
    not (-k in A[i][1]) and not (l in A[j][1]) and not(-l in A[j][1])
    and IntA=[]  then
    XX1:=Concatenation(["A",String(i)]);
        # XX1: represents a specific Whitehead automorphism (A,a) of A
    XX2:=Concatenation(["A",String(j)]);
        # XX2: represents a specific Whitehead automorphism (B,b) of A
    idx1:=0;
    idx2:=0;
    for t in [1..sA] do  # Verify the indices of the given Whitehead
                         # automorphisms A(i) and A(j) in A
          if XX1=T[t] then
              idx1:=t;
          fi;
          if XX2=T[t] then
              idx2:=t;
          fi;
    od;
    Add(Rels,[0,idx1,idx2,-idx1,-idx2]);
fi;
if -l in A[i][1] and not (l in A[i][1]) and A[i][2]=-l and -k in A[j][1]
    and not (k in A[j][1])  and A[j][2]=-k and not (k in A[i][1]) and
```

```
                    not (-k in A[i][1]) and not (l in A[j][1]) and not(-l in A[j][1])
                    and IntA=[]  then
                    XX1:=Concatenation(["A",String(i)]);
                        # XX1: represents a specific Whitehead automorphism (A,a) of A
                    XX2:=Concatenation(["A",String(j)]);
                        # XX2: represents a specific Whitehead automorphism (B,b) of A
                    idx1:=0;
                    idx2:=0;
                    for t in [1..sA] do  # Verify the indices of the given Whitehead
                                         # automorphisms A(i) and A(j) in A
                          if XX1=T[t] then
                              idx1:=t;
                          fi;
                          if XX2=T[t] then
                              idx2:=t;
                          fi;
                    od;
                    Add(Rels,[0,idx1,idx2,-idx1,-idx2]);

               fi;
           od;
       od;
    od;
od;
##
##############################################################################
##
##############################################################################
##
## In this section we compute the list of indices [0,idx1,idx2,-idx1,-idx2] of
## relators of type (R3b) by satisfying the conditions of this relations and add
## them to the list Rels. Note that 0 is just flag to let us know that all the
## generators here of power 1. idx1: represents the index of the generator A(i).
## idx2: represents the index of the generator A(j). -idx1: represents the index
## of the inverse of the generator A(i). -idx2: represents the index of the
## inverse of the generator A(j).
## For example if [0,idx1,idx2,-idx1,-idx2]= [ 0, 9, 3, -9, -3 ], then this
## means that A9*A3*A9^-1*A3^-1=1.
##
for k in [1..sV] do
    for l in [1..sV] do
        for i in [1..sA]do
            for j in [1..sA] do
                IntA:=Intersection( [ A[i][1] , A[j][1] ] );
                if l in A[i][1] and not (-l in A[i][1]) and A[i][2]=l and k in A[j][1]
                    and not (-k in A[j][1]) and A[j][2]=k and not (k in A[i][1]) and
                    not (-k in A[i][1]) and not (l in A[j][1]) and not(-l  in A[j][1])
                    and IntA<>[] and l in Lk[k]  then
                    XX1:=Concatenation(["A",String(i)]);
                        # XX1: represents a specific Whitehead automorphism (A,a) of A
                    XX2:=Concatenation(["A",String(j)]);
                        # XX2: represents a specific Whitehead automorphism (B,b) of A
                    idx1:=0;
```

```
        idx2:=0;
        for t in [1..sA] do
              if XX1=T[t] then
                  idx1:=t;
              fi;
              if XX2=T[t] then
                  idx2:=t;
              fi;
        od;
        Add(Rels,[0,idx1,idx2,-idx1,-idx2]);
    fi;
    if l in A[i][1] and not (-l in A[i][1]) and A[i][2]=l and -k in A[j][1]
        and not (k in A[j][1]) and A[j][2]=-k and not (k in A[i][1]) and
        not (-k in A[i][1]) and not (l in A[j][1]) and not(-l  in A[j][1])
        and IntA<>[] and l in Lk[k]  then
        XX1:=Concatenation(["A",String(i)]);
           # XX1: represents a specific Whitehead automorphism (A,a) of A
        XX2:=Concatenation(["A",String(j)]);
           # XX2: represents a specific Whitehead automorphism (B,b) of A
        idx1:=0;
        idx2:=0;
        for t in [1..sA] do
              if XX1=T[t] then
                  idx1:=t;
              fi;
              if XX2=T[t] then
                  idx2:=t;
              fi;
        od;
        Add(Rels,[0,idx1,idx2,-idx1,-idx2]);
    fi;
    if -l in A[i][1] and not (l in A[i][1]) and A[i][2]=-l and k in A[j][1]
        and not (-k in A[j][1])  and A[j][2]=k and not (k in A[i][1]) and
        not (-k in A[i][1]) and not (l in A[j][1]) and not(-l  in A[j][1])
        and IntA<>[] and l in Lk[k]  then
        XX1:=Concatenation(["A",String(i)]);
            # XX1: represents a specific Whitehead automorphism (A,a) of A
        XX2:=Concatenation(["A",String(j)]);
            # XX2: represents a specific Whitehead automorphism (B,b) of A
        idx1:=0;
        idx2:=0;
        for t in [1..sA] do
              if XX1=T[t] then
                  idx1:=t;
              fi;
              if XX2=T[t] then
                  idx2:=t;
              fi;
        od;
        Add(Rels,[0,idx1,idx2,-idx1,-idx2]);
     fi;
     if -l in A[i][1] and not (l in A[i][1]) and A[i][2]=-l and -k in A[j][1]
         and not (k in A[j][1])  and A[j][2]=-k and not (k in A[i][1]) and
```

```
                     not (-k in A[i][1]) and not (l in A[j][1]) and not(-l  in A[j][1])
                     and IntA<>[] and l in Lk[k]  then
                    XX1:=Concatenation(["A",String(i)]);
                        # XX1: represents a specific Whitehead automorphism (A,a) of A
                    XX2:=Concatenation(["A",String(j)]);
                        # XX2: represents a specific Whitehead automorphism (B,b) of A
                    idx1:=0;
                    idx2:=0;
                    for t in [1..sA] do
                          if XX1=T[t] then
                              idx1:=t;
                          fi;
                          if XX2=T[t] then
                              idx2:=t;
                          fi;
                    od;
                    Add(Rels,[0,idx1,idx2,-idx1,-idx2]);
                fi;
            od;
        od;
    od;
od;
##
##############################################################################
##
sRels:=Size(Rels);
return([Rels,sRels]);
end;
```

11. APCGRelationR4 Function

```
APCGRelationR4:=function(A,T,Lk,Rels)
local k,j,i,IntA,UniA,NUniA,l,K,t,UA6,R2,XX1,XX2,XX3,idx1,idx2,idx3,t1,R4a,
R4a1,R4a2,R4a3,K1,R4b,R4b1,R4b2,R4b3,srels,sRels,diff15,diff17,diff19,diff21,
diff22,diff16,diff18,diff20,UAdiff1,UAdiff15,UAdiff16,UAdiff17,UAdiff18,UAdiff19,
sV,sA,UAdiff20,UAdiff21,UAdiff22,UA7,UA8,UA9,UA10,UA11,UA12,UA13,n;
##
##############################################################################
##
## The input of this function are:
### A: the list of type(2) generators computed in "WhiteheadAutomorphismsOfSecondType",
### T: list of the names of elements of A,
### Lk: the list of links computed in "StarLinkDominateOfVertex".
### Rels: the list of row matrices of indices of the relations (it is one
### of the outputs of the "APCGRelationR3",
### Note that in order to get just the row matrices of indices of relation (R4)
### we need to pass an empty list [] rather than the list Rels above.
##
## It computes the list of indices of the generators [0,idx1,idx2,-idx1,-idx3,-idx2]
## of relators of types (R4a) and (R4b) of the group Aut(G_zeta) and adds them
## to the list Rels. In addition it calculates the size of the list Rels.
```

```
## It returns [Rels,sRels].
###############################################################################
##
sV:=Size(Lk); #Since the size of links list equal to sV, the size of the vertex list
sA:=Size(A);
##
################################################################################
##
## In this section we compute the list of indices [0,idx1,idx2,-idx1,-idx3,-idx2]
## of relators of type (R4a) by satisfying the conditions of this relations and
## add them to the list Rels. Note that 0 is just flag to let us know that all
## the generators here of power 1. idx1: represents the index of the generator A(i).
## idx2: represents the index of the generator A(j). -idx1: means the inverse of A(i).
## -idx3: means the inverse of the generator A(n). -idx2: means the inverse of A(j).
## For example if [0,idx1,idx2,-idx1,-idx3,-idx2]= [ [ 0, 1, 13, -1, -9, -13 ],
## then this means that A1*A13*A1^-1*A9^-1*A13^-1=1.
##
for k in [1..sV] do              # loop through the vertex list V
    for l in [1..sV] do          # loop through the vertex list V
        for i in [1..sA]do       # loop through A the Type (2) Whitehead Automorphisms
            for j in [1..sA] do  # loop through A the Type (2) Whitehead Automorphisms
                IntA:=Intersection( [ A[i][1] , A[j][1] ] );
                if l in A[i][1] and not (-l in A[i][1]) and A[i][2]=l and k in A[j][1]
                   and not (-k in A[j][1])  and A[j][2]=k and not (k in A[i][1]) and
                   not (-k in A[i][1]) and not (l in A[j][1]) and -l  in A[j][1]
                   and IntA=[]  then
                          diff15:=Difference(A[i][1],[l]);
                          Add(diff15,k);
                          for n in [1..sA]do
                          UA6:=SSortedList(A[n][1]);
                          UAdiff15:=SSortedList(diff15);
                          if UA6=UAdiff15 and A[n][2]=k then
                               XX1:=Concatenation(["A",String(i)]);
                                   # XX1: represents a specific automorphism (B,b) of A
                               XX2:=Concatenation(["A",String(j)]);
                                   # XX2: represents a specific automorphism (A,a) of A
                               XX3:=Concatenation(["A",String(n)]);
                                   # XX3: represents a specific automorphism (B-b+a,a) of A
                               idx1:=0;
                               idx2:=0;
                               idx3:=0;
                               for t in [1..sA] do
                                     # Verify the indices of the given Whitehead
                                     # automorphisms A(i), A(j) and A(n) in A
                                     if XX1=T[t] then
                                         idx1:=t;
                                     fi;
                                     if XX2=T[t] then
                                         idx2:=t;
                                     fi;
                                     if XX3=T[n] then
                                         idx3:=n;
                                     fi;
```

```
                od;
                Add(Rels,[0,idx1,idx2,-idx1,-idx3,-idx2]);
            fi;
        od;
    fi;
    if l in A[i][1] and not (-l in A[i][1]) and A[i][2]=l and -k in A[j][1]
        and not (k in A[j][1]) and A[j][2]=-k and not (-k in A[i][1]) and
        not (k in A[i][1]) and not (l in A[j][1]) and -l in A[j][1] and
        IntA=[]  then
        diff19:=Difference(A[i][1],[l]);
        Add(diff19,-k);
        for n in [1..sA]do
            UA10:=SSortedList(A[n][1]);
           UAdiff19:=SSortedList(diff19);
            if UA10=UAdiff19 and A[n][2]=-k then
                XX1:=Concatenation(["A",String(i)]);
                    # XX1: represents a specific automorphism (B,b) of A
                XX2:=Concatenation(["A",String(j)]);
                    # XX2: represents a specific automorphism (A,a) of A
                XX3:=Concatenation(["A",String(n)]);
                    # XX3: represents a specific automorphism (B-b+a,a) of A
                idx1:=0;
                idx2:=0;
                idx3:=0;
                for t in [1..sA] do
                      # Verify the indices of the given Whitehead
                      # automorphisms A(i), A(j) and A(n) in A
                      if XX1=T[t] then
                          idx1:=t;
                      fi;
                      if XX2=T[t] then
                          idx2:=t;
                      fi;
                      if XX3=T[n] then
                          idx3:=n;
                      fi;
                od;
                Add(Rels,[0,idx1,idx2,-idx1,-idx3,-idx2]);
            fi;
        od;
    fi;
    if -l in A[i][1] and not (l in A[i][1]) and A[i][2]=-l and -k in A[j][1]
         and not (k in A[j][1]) and A[j][2]=-k and not (-k in A[i][1])
         and not (k in A[i][1]) and not (-l in A[j][1]) and l in A[j][1]
         and IntA=[]  then
         diff16:=Difference(A[i][1],[-l]);
         Add(diff16,-k);
         for n in [1..sA]do
             UA7:=SSortedList(A[n][1]);
            UAdiff16:=SSortedList(diff16);
             if UA7=UAdiff16 and A[n][2]=-k then
               XX1:=Concatenation(["A",String(i)]);
                   # XX1: represents a specific automorphism (B,b) of A
```

```
            XX2:=Concatenation(["A",String(j)]);
                # XX2: represents a specific automorphism (A,a) of A
            XX3:=Concatenation(["A",String(n)]);
                # XX3: represents a specific automorphism (B-b+a,a) of A
            idx1:=0;
            idx2:=0;
            idx3:=0;
            for t in [1..sA] do
                  # Verify the indices of the given Whitehead
                  # automorphisms A(i), A(j) and A(n) in A
                  if XX1=T[t] then
                      idx1:=t;
                  fi;
                  if XX2=T[t] then
                      idx2:=t;
                  fi;
                  if XX3=T[n] then
                      idx3:=n;
                  fi;
            od;
            Add(Rels,[0,idx1,idx2,-idx1,-idx3,-idx2]);
        fi;
    od;
fi;
if -l in A[i][1] and not (l in A[i][1]) and A[i][2]=l and k in A[j][1]
      and not (-k in A[j][1]) and A[j][2]=k and not (k in A[i][1])
      and not (-k in A[i][1]) and not (-l in A[j][1]) and l in A[j][1]
      and IntA=[]  then
      diff20:=Difference(A[i][1],[-l]);
      Add(diff20,k);
      for n in [1..sA]do
          UA11:=SSortedList(A[n][1]);
         UAdiff1:=SSortedList(diff20);
          if UA11=UAdiff20 and A[n][2]=k then

            XX1:=Concatenation(["A",String(i)]);
                # XX1: represents a specific automorphism (B,b) of A
            XX2:=Concatenation(["A",String(j)]);
                # XX2: represents a specific automorphism (A,a) of A
            XX3:=Concatenation(["A",String(n)]);
                # XX3: represents a specific automorphism (B-b+a,a) of A
            idx1:=0;
            idx2:=0;
            idx3:=0;
            for t in [1..sA] do
                  # Verify the indices of the given Whitehead
                  # automorphisms A(i), A(j) and A(n) in A
                  if XX1=T[t] then
                      idx1:=t;
                  fi;
                  if XX2=T[t] then
                      idx2:=t;
                  fi;
```

```
                                  if XX3=T[n] then
                                      idx3:=n;
                                  fi;
                            od;
                            Add(Rels,[0,idx1,idx2,-idx1,-idx3,-idx2]);
                         fi;
                    od;
               fi;
          od;
      od;
  od;
od;
##
##############################################################################
##
##############################################################################
##
## In this section we compute the list of indices [0,idx1,idx2,-idx1,-idx3,-idx2]
## of relators of type (R4b) by satisfying the conditions of this relations and
## add them to the list Rels. Note that 0 is just flag to let us know that all
## the generators here of power 1. idx1: represents the index of the generator A(i).
## idx2: represents the index of the generator A(j). -idx1: means the inverse of A(i).
## of the inverse of the generator A(i).-idx3: means the inverse of the generator A(n).
## -idx2: means the inverse of A(j).
## For example if [0,idx1,idx2,-idx1,-idx3,-idx2]= [ 0, 25, 21, -25, -13,-21]
## then this means that A25*A21*A25^-1*A13^-1*A21^-1=1.
## The procedure use in this Section is similar to the first Section except
## IntA<>[] replaced by IntA<>[] and l in Lk[k]
##
for k in [1..sV] do
    for l in [1..sV] do
        for i in [1..sA]do
            for j in [1..sA] do
                IntA:=Intersection( [ A[i][1] , A[j][1] ] );
                if l in A[i][1] and not (-l in A[i][1]) and A[i][2]=l and k in A[j][1]
                     and not (-k in A[j][1]) and A[j][2]=k and not (k in A[i][1])
                     and not (-k in A[i][1]) and not (l in A[j][1]) and -l in A[j][1]
                     and IntA<>[] and l in Lk[k]  then
                     diff17:=Difference(A[i][1],[l]);
                     Add(diff17,k);
                     for n in [1..sA]do
                          UA8:=SSortedList(A[n][1]);
                         UAdiff17:=SSortedList(diff17);
                          if UA8=UAdiff17 and A[n][2]=k then
                             XX1:=Concatenation(["A",String(i)]);
                             XX2:=Concatenation(["A",String(j)]);
                             XX3:=Concatenation(["A",String(n)]);
                             idx1:=0;
                             idx2:=0;
                             idx3:=0;
                             for t in [1..sA] do
                                   if XX1=T[t] then
                                       idx1:=t;
```

```
                    fi;
                    if XX2=T[t] then
                        idx2:=t;
                    fi;
                    if XX3=T[n] then
                        idx3:=n;
                    fi;
              od;
              Add(Rels,[0,idx1,idx2,-idx1,-idx3,-idx2]);
          fi;
     od;
fi;
if l in A[i][1] and not (-l in A[i][1]) and A[i][2]=l and -k in A[j][1]
     and not (k in A[j][1]) and A[j][2]=-k  and not (-k in A[i][1])
     and not (k in A[i][1]) and not (l in A[j][1]) and -l in A[j][1]
     and IntA<>[] and l in Lk[k]  then
     diff21:=Difference(A[i][1],[l]);
     Add(diff21,-k);
     for n in [1..sA]do
          UA12:=SSortedList(A[n][1]);
         UAdiff21:=SSortedList(diff21);
          if UA12=UAdiff21 and A[n][2]=-k then
             XX1:=Concatenation(["A",String(i)]);
             XX2:=Concatenation(["A",String(j)]);
             XX3:=Concatenation(["A",String(n)]);
             idx1:=0;
             idx2:=0;
             idx3:=0;
             for t in [1..sA] do
                   if XX1=T[t] then
                       idx1:=t;
                   fi;
                   if XX2=T[t] then
                       idx2:=t;
                   fi;
                   if XX3=T[n] then
                       idx3:=n;
                   fi;
              od;
              Add(Rels,[0,idx1,idx2,-idx1,-idx3,-idx2]);
         fi;
     od;
fi;
if -l in A[i][1] and not (l in A[i][1]) and A[i][2]=-l and -k in A[j][1]
      and not (k in A[j][1]) and A[j][2]=-k and not (-k in A[i][1])
      and not (k in A[i][1]) and not (-l in A[j][1]) and l in A[j][1]
      and IntA<>[] and l in Lk[k]  then
      diff18:=Difference(A[i][1],[-l]);
      Add(diff18,-k);
      for n in [1..sA]do
           UA9:=SSortedList(A[n][1]);
         UAdiff18:=SSortedList(diff18);
          if UA9=UAdiff18 and A[n][2]=-k then
```

```
                    XX1:=Concatenation(["A",String(i)]);
                    XX2:=Concatenation(["A",String(j)]);
                    XX3:=Concatenation(["A",String(n)]);
                    idx1:=0;
                    idx2:=0;
                    idx3:=0;
                    for t in [1..sA] do
                          if XX1=T[t] then
                              idx1:=t;
                          fi;
                          if XX2=T[t] then
                              idx2:=t;
                          fi;
                          if XX3=T[n] then
                              idx3:=n;
                          fi;
                     od;
                     Add(Rels,[0,idx1,idx2,-idx1,-idx3,-idx2]);
                fi;
           od;
      fi;
      if -l in A[i][1] and not (l in A[i][1]) and A[i][2]=-l and k in A[j][1]
            and not (-k in A[j][1]) and A[j][2]=k and not (k in A[i][1])
            and not (-k in A[i][1]) and not (-l in A[j][1]) and l in A[j][1]
            and IntA<>[] and l in Lk[k]  then
            diff22:=Difference(A[i][1],[-l]);
            Add(diff22,k);
            for n in [1..sA]do
                 UA13:=SSortedList(A[n][1]);
               UAdiff22:=SSortedList(diff22);
                if UA13=UAdiff22 and A[n][2]=k then
                    XX1:=Concatenation(["A",String(i)]);
                    XX2:=Concatenation(["A",String(j)]);
                    XX3:=Concatenation(["A",String(n)]);
                    idx1:=0;
                    idx2:=0;
                    idx3:=0;
                    for t in [1..sA] do
                          if XX1=T[t] then
                              idx1:=t;
                          fi;
                          if XX2=T[t] then
                              idx2:=t;
                          fi;
                          if XX3=T[n] then
                              idx3:= n;
                          fi;
                    od;
                    Add(Rels,[0,idx1,idx2,-idx1,-idx3,-idx2]);
                fi;
            od;
       fi;
od;
```

```
        od;
    od;
od;
##
##########################################################################
##
sRels:=Size(Rels);
return([Rels,sRels]);
end;
```

12. APCGRelationR5 Function

```
APCGRelationR5:=function(A,St,Lk,Rels,T)
local k,j,i,m,UA,UAiff,UAiff2,IntA,UniA,NUniA,l,K,t,UA1,XX1,XX2,XX3,idx1,idx2,
sV,sA,idx3,idx4,t1,R5,srels,sRels,diff,diff1,diff2,UAdiff,UAdiff1,UAdiff2,lk,Y2;
##
############################################################################
##
### A: the list of type(2) generators computed in "WhiteheadAutomorphismsOfSecondType",
### St: the list of stars computed in "StarLinkDominateOfVertex",
### Lk: the list of links computed in "StarLinkDominateOfVertex",
### Rels: the list of row matrices of indices of the relations (it is one
### of the outputs of the "RelationsOfGraphAutomorphisms",
### Note that in order to get just the row matrices of indices of relation (R3)
### we need to pass an empty list [] rather than the list Rels above.
### T: list of the names of elements of A.
##
## It computes the list of indices of the generators [2,idx1,idx2,idx4,-idx3,j,k,j]
## of relators of type (R5) by satisfying the conditions of this relations
## and add them to the list Rels. Note that the first entry "2" in the
## list of indices above means that the idx4 refers to the location of A's
## (which are start at sA+1 and end at sA+sGens2) and this type of generators
## are automorphisms of graph that, just swap the vertex "b" (j in this code)
## to the vertex "a" (k in this code) and vice versa. idx1: represents the
## index of the generator A(l). idx2: represents the index of the generator
## A(i). -idx3: represents the the inverse of the generator A(m). j and k
## refer to the vertex or its inverse. In addition it calculates the sizes
## of the list Rels.
## For example if [2,idx1,idx2,-idx3,idx4,j,k,j]= [[2, 25, 1, 31, -3, 3, 1,3],
## then this means that A25*A1*A31*A3^-1=1.
##
## It returns [Rels,sRels].
#############################################################################
##
sV:=Size(St); #Since the size of stars list equal to sV, the size of the vertex list
sA:=Size(A);
lk:=[];
for i in [1..sV] do
    Y2:=Difference(Lk[i],[0]);
    Add(lk,Y2);
od;
```

```
for k in [1..sV] do
    for j in [1..sV] do
        for i in [1..sA]do
            ##############################################################################
            ##
            ## In this section we compute first part of the list of indices of the
            ## generators which is [2,idx1,idx2,idx4,-idx3,j,k,j] of the relators of
            ## type (R5) when the multiplier "a" (k in this code) of the automorphism (A,a)
            ## is the original vertex "a" (not the inverse of "a"), and the multiplier "b"
            ## (j in this code) of the automorphism (A-a+a^-1,b) is the original vertex "b"
            ## and k not equal to j with k~j, by satisfying the conditions of this relations.
            ## 2: means that idx4 refers to the location of A's.
            ## idx1: represents the index "l" of a specific generator A(l) of A.
            ## idx2: represents the index "i" of a specific generator A(i) of A.
            ## -idx3: represents the inverse of the specific generator A(m) of A which
            ##        corresponds to the index idx3.
            ## idx4: refers to the index of A's which starts at sA+1 and end at sA+sGens2
            ## For example if [2,idx1,idx2,idx4,-idx3,j,k,j]= [ 2, 25, 1, 31, -3, 3, 1, 3 ]
            ## then this means that A25*A1*A31*A3^-1=1.
            ##
            if k in A[i][1] and not (-k in A[i][1]) and j in A[i][1] and not (-j in A[i][1])
                 and j<>k and A[i][2]=k and IsSubset(St[k],lk[j])=true and
                 IsSubset(St[j],lk[k])=true then
                 diff1:=Difference(A[i][1],[k]);
                 Add(diff1,-k);
                 diff2:=Difference(A[i][1],[j]);
                 Add(diff2,-j);
                 for l in [1..sA]do
                     UA:=SSortedList(A[l][1]);
                    UAiff:=SSortedList(diff1);
                     for m in [1..sA]do
                         UA1:=SSortedList(A[m][1]);
                         UAiff2:=SSortedList(diff2);
                            if UA=UAiff and A[l][2]=j and UA1=UAiff2 and A[m][2]=k then
                              idx4:=sA+j;
                              XX1:=Concatenation(["A",String(l)]);
                              # XX1: represents a specific automorphism (A-a+a^-1,b) of A
                              XX2:=Concatenation(["A",String(i)]);
                              # XX2: represents a specific automorphism (A,a) of A
                              XX3:=Concatenation(["A",String(m)]);
                              # XX3: represents a specific automorphism (A-b+b^-1,a) of A
                              idx1:=0;
                              idx2:=0;
                              idx3:=0;
                              for t in [1..sA] do
                                    # Verify the indices of the given Whitehead
                                    # automorphisms A(l), A(i) and A(m) in A
                                    if XX1=T[t] then
                                        idx1:=t;
                                    fi;
                                    if XX2=T[t] then
                                        idx2:=t;
                                    fi;
```

```
                        if XX3=T[t] then
                            idx3:=t;
                        fi;
                  od;
                  Add(Rels,[2,idx1,idx2,idx4,-idx3,j,k,j]);
                    # 2: means that the idx4 refers to the location of A's
                    #    which starts at sA+1 and end at sA+sGens2,
                    # j: refers to the vertex or its inverse
               fi;
            od;
        od;
fi;
##
################################################################################
##
################################################################################
##
## In this section we compute second part of the list of indices of the
## generators which is [2,idx1,idx2,idx4,-idx3,j,k,j] of the relators of
## type (R5) when the multiplier "a" (k in this code) of the automorphism (A,a)
## is the original vertex "a", and the multiplier "b" (j in this code) of the
## automorphism (A-a+a^-1,b) is the inverse of the vertex "b" (-j in this code)
## and k not equal to -j with k~ -j, by satisfying the conditions of this
## relations.
## The procedure use in this Section is similar to the first Section above.
##
if k in A[i][1] and not (-k in A[i][1]) and -j in A[i][1] and not (j in A[i][1])
     and -j<>k and A[i][2]=k and IsSubset(St[k],lk[j])=true and
     IsSubset(St[j],lk[k])=true then
     diff1:=Difference(A[i][1],[k]);
     Add(diff1,-k);
     diff2:=Difference(A[i][1],[-j]);
     Add(diff2,j);
     for l in [1..sA]do
         UA:=SSortedList(A[l][1]);
        UAiff:=SSortedList(diff1);
         for m in [1..sA]do
             UA1:=SSortedList(A[m][1]);
             UAiff2:=SSortedList(diff2);
             if UA=UAiff and A[l][2]=-j and UA1=UAiff2 and A[m][2]=k then
                 idx4:=sA+j;
                 XX1:=Concatenation(["A",String(l)]);
                 XX2:=Concatenation(["A",String(i)]);
                 XX3:=Concatenation(["A",String(m)]);
                 idx1:=0;
                 idx2:=0;
                 idx3:=0;
                 for t in [1..sA] do
                     if XX1=T[t] then
                         idx1:=t;
                     fi;
                     if XX2=T[t] then
                         idx2:=t;
```

```
                    fi;
                    if XX3=T[t] then
                        idx3:=t;
                    fi;
                od;
                Add(Rels,[2,idx1,idx2,idx4,-idx3,-j,k,-j]);
            fi;
        od;
    od;
fi;
##
################################################################################
##
################################################################################
##
## In this section we compute third part of the list of indices of the
## generators which is [2,idx1,idx2,idx4,-idx3,j,k,j] of the relators of
## type (R5) when the multiplier "a" (k in this code) of the automorphism (A,a)
## is the inverse of the vertex "a" (-k in this code), and the multiplier "b"
## (j in this code) of the automorphism (A-a+a^-1,b) is the original vertex "b"
## and -k not equal to j with -k ~ j, by satisfying the conditions of this
## relations.
## The procedure use in this Section is similar to the first Section above.
##
if -k in A[i][1] and not (k in A[i][1]) and j in A[i][1] and not (-j in A[i][1])
      and j<>-k and A[i][2]=-k and IsSubset(St[k],lk[j])=true and
      IsSubset(St[j],lk[k])=true then
      diff1:=Difference(A[i][1],[-k]);
      Add(diff1,k);
      diff2:=Difference(A[i][1],[j]);
      Add(diff2,-j);
      for l in [1..sA]do
          UA:=SSortedList(A[l][1]);
         UAiff:=SSortedList(diff1);
          for m in [1..sA]do
              UA1:=SSortedList(A[m][1]);
              UAiff2:=SSortedList(diff2);
              if UA=UAiff and A[l][2]=j and UA1=UAiff2 and A[m][2]=-k then
                  idx4:=sA+j;
                  XX1:=Concatenation(["A",String(l)]);
                  XX2:=Concatenation(["A",String(i)]);
                  XX3:=Concatenation(["A",String(m)]);
                  idx1:=0;
                  idx2:=0;
                  idx3:=0;
                  for t in [1..sA] do
                      if XX1=T[t] then
                          idx1:=t;
                      fi;
                      if XX2=T[t] then
                          idx2:=t;
                      fi;
                      if XX3=T[t] then
```

```
                        idx3:=t;
                    fi;
                od;
                Add(Rels,[2,idx1,idx2,idx4,-idx3,j,-k,j]);
            fi;
        od;
    od;
fi;
##
##############################################################################
##
##############################################################################
##
## In this section we compute third part of the list of indices of the
## generators which is [2,idx1,idx2,idx4,-idx3,j,k,j] of the relators of
## type (R5) when the multiplier "a" (k in this code) of the automorphism
## (A,a) is the inverse of the vertex "a" (-k in this code), and the
## multiplier "b" (j in this code) of the automorphism (A-a+a^-1,b) is
## the inverse of the vertex "b" (-j in this code) and -k not equal to
## -j with -k ~ -j, by satisfying the conditions of this relations.
## The procedure use in this Section is similar to the first Section above.
##
if -k in A[i][1] and not (k in A[i][1]) and -j in A[i][1] and not (j in A[i][1])
    and -j<>-k and A[i][2]=-k and IsSubset(St[k],lk[j])=true and
    IsSubset(St[j],lk[k])=true then
    diff1:=Difference(A[i][1],[-k]);
    Add(diff1,k);
    diff2:=Difference(A[i][1],[-j]);
    Add(diff2,j);
    for l in [1..sA]do
        UA:=SSortedList(A[l][1]);
       UAiff:=SSortedList(diff1);
        for m in [1..sA]do
            UA1:=SSortedList(A[m][1]);
            UAiff2:=SSortedList(diff2);
            if UA=UAiff and A[l][2]=-j and UA1=UAiff2 and A[m][2]=-k then
                idx4:=sA+j;
                XX1:=Concatenation(["A",String(l)]);
                XX2:=Concatenation(["A",String(i)]);
                XX3:=Concatenation(["A",String(m)]);
                idx1:=0;
                idx2:=0;
                idx3:=0;
                for t in [1..sA] do
                      if XX1=T[t] then
                          idx1:=t;
                      fi;
                      if XX2=T[t] then
                          idx2:=t;
                      fi;
                      if XX3=T[t] then
                          idx3:=t;
                      fi;
```

```
                            od;
                            Add(Rels,[2,idx1,idx2,idx4,-idx3,-j,-k,-j]);
                        fi;
                    od;
                od;
            fi;

            ##
            #########################################################################
            ##
        od;
    od;
od;
sRels:=Size(Rels);
return([Rels,sRels]);
end;
```

13. APCGRelationR8 Function

```
APCGRelationR8:=function(V,A,T,Lk,Rels)
local k,j,i,IntA,UniA,NUniA,l,K,t,UA1,UA2,UA3,UA4,UA5,UA6,R2,XX1,XX2,XX3,idx1,
idx2,idx3,t1,R8,NR8,ty,invLk1,srels,sRels,diff1,diff2,diff3,diff4,diff5,diff6,
diff7,diff8,diff9,diff10,UAdiff1,UAdiff2,UAdiff3,UAdiff4,UAdiff5,UAdiff6,UAdiff7,
sV,sA,UAdiff8,UAdiff9,UAdiff10,UA7,UA8,UA13,n,invV,L,invLk,UniLk;
##
###############################################################################
##
## The input of this function are:
### V: the list of vertices of the graph zeta,
### A: the list of type(2) generators computed in "WhiteheadAutomorphismsOfSecondType",
### T: list of the names of elements of A,
### Lk: the list of links computed in "StarLinkDominateOfVertex".
### Rels: the list of row matrices of indices of the relations (it is one
### of the outputs of the "APCGRelationR4",
### Note that in order to get just the row matrices of indices of relation (R8)
### we need to pass an empty list [] rather than the list Rels above.
##
## It computes the list of indices of the generators [0,idx1,-idx3,-idx2],
## [0,idx1,-idx2], and [0,idx1] of relators of type (R8) of the group
## Aut(G_zeta) by satisfying the conditions of this relations and add them
## to the list Rels. In addition it calculates the size of the list Rels.
## It returns [Rels,sRels].
###############################################################################
##
sV:=Size(V);
sA:=Size(A);
invV:=-V;                 # invV is the inverses list of the vertex list V
L:=Concatenation(V,invV); # L is the union of the lists V and invV

for k in [1..sV] do       # loop through the vertex list V
    ##
```

```
#############################################################################
##
## In this part we compute the list of indices When Lk(k) is not empty list.
##
if Lk[k]<>[0] then
    for i in [1..sA]do  # loop throu A the Type (2) Whitehead Automorphisms
        if k in A[i][1] and not (-k in A[i][1]) and A[i][2]=k then
            diff3:=Difference(L,[-k]);
            invLk:=-Lk[k];
            UniLk:=Concatenation(Lk[k],invLk);
            diff5:=Difference(L,A[i][1]);
            diff4:=[];
            diff6:=[];
            for l in Lk[k] do # In this loop if the vertex l and its inverse -l in the
                              # same time are belong to the list diff3 then we delete
                              # them, because they will cancel each other.
                              # We do the same if l and -l belong to the list diff5
                if l in diff3 and -l in diff3 then
                   diff4:=Difference(diff3,[-l,l]);
                   diff3:=diff4;
                fi;
                if l in diff5 and -l in diff5 then
                   diff6:=Difference(diff5,[-l,l]);
                   diff5:=diff6;
                fi;
           od;
           UAdiff4:=SSortedList(diff4);
           UAdiff5:=SSortedList(diff5);
           UAdiff6:=SSortedList(diff6);
           K:=[k];
           ty:=0;
           for j in [1..sA]do     # loop through A, the Type (2) Whitehead Automorphisms
               for n in [1..sA]do # loop through A, the Type (2) Whitehead Automorphisms
                   UA2:=SSortedList(A[j][1]);
                  UA3:=SSortedList(A[n][1]);
                   #####################################################################
                   ##
                   ## In this section we compute first part of the list of indices of the
                   ## generators which is [0,idx1,-idx3,-idx2] of the relators of type
                   ## (R8) by satisfying the conditions of this relations. Note that 0
                   ## is just flag to let us know that all the generators here of power 1.
                   ## idx1: represents the index of a specific generator A(i) of A.
                   ## -idx3: represents the index of the inverse of a specific generator
                   ## A(n) of A.
                   ## -idx2: represents the index of the inverse of a specific generator
                   ## A(j) of A.
                   ## For example if [0,idx1,-idx3,-idx2]= [ 0, 1, -4, -5 ], then this
                   ## means that A1*A4^-1*A5^-1=1.
                   ##
                  if  UAdiff4=UA2 and A[j][2]=k and  UAdiff6=UA3 and A[n][2]=-k then

                        XX1:=Concatenation(["A",String(i)]);
                        # XX1: represents a specific automorphism (A,a) of A
```

```
     XX2:=Concatenation(["A",String(j)]);
     # XX2: represents a specific automorphism (L-A, a^-1) of A
     XX3:=Concatenation(["A",String(n)]);
     # XX3: represents a specific automorphism (L-a^-1, a) of A
     idx1:=0;
     idx2:=0;
     idx3:=0;
     for t in [1..sA] do # Verify the indices of the given Whitehead
                         # automorphisms A(i), A(j) and A(n) in A
           if XX1=T[t] then
               idx1:=t;
           fi;
           if XX2=T[t] then
               idx2:=t;
           fi;
           if XX3=T[n] then
               idx3:=n;
           fi;
      od;
      NR8:=[0,idx1,-idx3,-idx2];
      ty:=1;
fi;
##
###################################################################
##
###################################################################
##
## In this section we compute second part of the list of indices of
## the generators which is [0,idx1,-idx2] of the relators of type
## (R8) by satisfying the conditions of this relations. Note that 0
## is just flag to let us know that all the generators here of
## power 1.
## idx1: represents the index of a specific generator A(i) of A.
## -idx2: represents the index of the inverse of a specific
## generator A(n) of A.
## For example if [0,idx1,-idx2]= [ 0, 7, -14 ], then this means
## that A7*A14^-1=1.
## Note that we have this case, because some time L-A-[l,-l]= [k]
## which is just the identity or L-a^-1-[l,-l]= [k] which is just
## the identity.
##
if UAdiff4=K and A[j][2]=k and  UAdiff6=UA3 and A[n][2]=-k then
     XX1:=Concatenation(["A",String(i)]);
     # XX1: represents a specific automorphism (A,a) of A
     XX2:=Concatenation(["A",String(n)]);
     # XX2: represents a specific automorphism (L-a^-1, a) of A
     idx1:=0;
     idx2:=0;
     for t in [1..sA] do
           if XX1=T[t] then
               idx1:=t;
           fi;
           if XX2=T[n] then
```

```
               idx2:=n;
           fi;
     od;
     NR8:=[0,idx1,-idx2];
     ty:=1;
fi;
if UAdiff4=K and A[j][2]=k and  UAdiff6=[] and UAdiff5= UA3
    and A[n][2]=-k then
    XX1:=Concatenation(["A",String(i)]);
    XX2:=Concatenation(["A",String(n)]);
    idx1:=0;
    idx2:=0;
    for t in [1..sA] do
          if XX1=T[t] then
              idx1:=t;
          fi;
          if XX2=T[n] then
              idx2:=n;
          fi;
    od;
    NR8:=[0,idx1,-idx2];
    ty:=1;
fi;
if UAdiff4=UA2 and A[j][2]=k and  UAdiff6=-K and A[n][2]=-k then
    XX1:=Concatenation(["A",String(i)]);
    XX2:=Concatenation(["A",String(j)]);
    idx1:=0;
    idx2:=0;
    for t in [1..sA] do
          if XX1=T[t] then
              idx1:=t;
          fi;
          if XX2=T[t] then
              idx2:=t;
          fi;
    od;
    NR8:=[0,idx1,-idx2];
    ty:=1;
fi;
##
##################################################################
##
##################################################################
##
## In this section we compute third part of the list of indices of
## the generators which is [0,idx1] of the relators of type (R8)
## by satisfying the conditions of this relations. Note that 0 is
## just flag to let us know that all the generators here of power 1.
## idx1: represents the index of a specific generator A(i) of A.
## Note that we have this case, because some time
## L-A-[l,-l]= L-a^-1-[l,-l]= [k] which is just the identity.
##
if UAdiff4=K and A[j][2]=k and  UAdiff6=-K and A[n][2]=-k then
```

```
              XX1:=Concatenation(["A",String(i)]);
              # XX1: represents a specific Whitehead automorphism (A,a) of A
              idx1:=0;
              for t in [1..sA] do
                    if XX1=T[t] then
                        idx1:=t;
                    fi;
              od;
              NR8:=[0,idx1];
              ty:=1;
          fi;
          ##
          #################################################################
          ##
     od;
  od;
fi;
if -k in A[i][1] and not (k in A[i][1]) and A[i][2]=-k then
     diff7:=Difference(L,[k]);
     invLk1:=-Lk[k];
     UniLk:=Concatenation(Lk[k],invLk1);
     diff9:=Difference(L,A[i][1]);
     diff8:=[];
     diff10:=[];
     for l in Lk[k] do
         if l in diff7 and -l in diff7 then
            diff8:=Difference(diff7,[-l,l]);
            diff7:=diff8;
         fi;
         if l in diff9 and -l in diff9 then
            diff10:=Difference(diff9,[-l,l]);
            diff9:=diff10;
         fi;
     od;
     K:=[-k];
     for j in [1..sA]do
         for n in [1..sA]do
             UA4:=SSortedList(A[j][1]);
            UAdiff8:=SSortedList(diff8);
             UA5:=SSortedList(A[n][1]);
             UAdiff9:=SSortedList(diff9);
            UAdiff10:=SSortedList(diff10);
             #############################################################
             ##
             ## This section is the same first section above, just we have
             ## replace the  multiplier "a" (k) by it inverse "a^-1" (-k).
             ##
             if UAdiff8=UA4 and  UAdiff10=UA5 then
                 XX1:=Concatenation(["A",String(i)]);
                 XX2:=Concatenation(["A",String(j)]);
                 XX3:=Concatenation(["A",String(n)]);
                 idx1:=0;
                 idx2:=0;
```

```
    idx3:=0;
    for t in [1..sA] do
          if XX1=T[t] then
              idx1:=t;
          fi;
          if XX2=T[t] then
              idx2:=t;
          fi;
          if XX3=T[n] then
              idx3:=n;
          fi;
    od;
    NR8:=[0,idx1,-idx3,-idx2];
    ty:=1;
fi;
##
###############################################################
##
###############################################################
##
## This section is the same second section above, just we have
## replace the multiplier "a" (k in this code) by it inverse
## "a^-1" (-k in this code).
##
if UAdiff8=K and A[j][2]=-k and  UAdiff10=UA5 and A[n][2]=k then
    XX1:=Concatenation(["A",String(i)]);
    XX2:=Concatenation(["A",String(n)]);
    idx1:=0;
    idx2:=0;
    for t in [1..sA] do
          if XX1=T[t] then
              idx1:=t;
          fi;
          if XX2=T[n] then
              idx2:=n;
          fi;
    od;
    NR8:=[0,idx1,-idx2];
    ty:=1;
fi;
if UAdiff8=K and A[j][2]=-k and  UAdiff10=[] and UAdiff9=UA5
    and A[n][2]=k then
    XX1:=Concatenation(["A",String(i)]);
    XX2:=Concatenation(["A",String(n)]);
    idx1:=0;
    idx2:=0;
    for t in [1..sA] do
          if XX1=T[t] then
              idx1:=t;
          fi;
          if XX2=T[n] then
              idx2:=n;
          fi;
```

```
                        od;
                        NR8:=[0,idx1,-idx2];
                        ty:=1;
                    fi;
                    if  UAdiff8=UA4 and A[j][2]=-k and   UAdiff10=-K and A[n][2]=k then
                        XX1:=Concatenation(["A",String(i)]);
                        XX2:=Concatenation(["A",String(j)]);
                        idx1:=0;
                        idx2:=0;
                        for t in [1..sA] do
                              if XX1=T[t] then
                                  idx1:=t;
                              fi;
                              if XX2=T[t] then
                                  idx2:=t;
                              fi;
                        od;
                        NR8:=[0,idx1,-idx2];
                        ty:=1;
                    fi;
                    ##
                    ################################################################
                    ##
                    ################################################################
                    ##
                    ## This section is the same third section above, just we have
                    ## replace the multiplier "a" (k) by it inverse "a^-1" (-k).
                    ##
                    if UAdiff8=K and A[j][2]=-k and   UAdiff10=-K and A[n][2]=k then

                        XX1:=Concatenation(["A",String(i)]);
                        idx1:=0;
                        idx2:=0;
                        for t in [1..sA] do
                              if XX1=T[t] then
                                  idx1:=t;
                              fi;
                        od;
                        NR8:=[0,idx1];
                        ty:=1;
                    fi;
                od;
            od;
        fi;
        if ty=1 then
              Add(Rels,NR8);
              NR8:=[];
              R8:=[];
              ty:=0;
        fi;
    od;
fi;
##
```

```
################################################################################
##
################################################################################
##
## In this part we compute the list of indices When Lk(k) is empty list which
## is the same first part when Lk(k) is not empty list with some small changes.
##
if Lk[k]=[0] then
   for i in [1..sA]do
      if k in A[i][1] and not (-k in A[i][1]) and A[i][2]=k then
        diff3:=Difference(L,[-k]);
        diff5:=Difference(L,A[i][1]);
        UAdiff4:=SSortedList(diff3);
        UAdiff6:=SSortedList(diff5);
        K:=[k];
        ty:=0;
        for j in [1..sA]do
            for n in [1..sA]do
                UA2:=SSortedList(A[j][1]);
               UA3:=SSortedList(A[n][1]);
               if  UAdiff4=UA2 and A[j][2]=k and  UAdiff6=UA3
                      and A[n][2]=-k then
                       XX1:=Concatenation(["A",String(i)]);
                       XX2:=Concatenation(["A",String(j)]);
                       XX3:=Concatenation(["A",String(n)]);
                       idx1:=0;
                       idx2:=0;
                       idx3:=0;
                       for t in [1..sA] do
                             if XX1=T[t] then
                                 idx1:=t;
                             fi;
                             if XX2=T[t] then
                                 idx2:=t;
                             fi;
                             if XX3=T[n] then
                                 idx3:=n;
                             fi;
                        od;
                        NR8:=[0,idx1,-idx3,-idx2];
                       ty:=1;
                fi;
                if UAdiff4=K and A[j][2]=k and  UAdiff6=UA3 and A[n][2]=-k then
                    XX1:=Concatenation(["A",String(i)]);
                    XX2:=Concatenation(["A",String(n)]);
                    idx1:=0;
                    idx2:=0;
                    for t in [1..sA] do
                          if XX1=T[t] then
                              idx1:=t;
                          fi;
                          if XX2=T[n] then
                              idx2:=n;
```

```
                fi;
            od;
            NR8:=[0,idx1,-idx2];
            ty:=1;
        fi;
        if  UAdiff4=UA2 and A[j][2]=k and  UAdiff6=-K and A[n][2]=-k then
            XX1:=Concatenation(["A",String(i)]);
            XX2:=Concatenation(["A",String(j)]);
            idx1:=0;
            idx2:=0;
            for t in [1..sA] do
                  if XX1=T[t] then
                      idx1:=t;
                  fi;
                  if XX2=T[t] then
                      idx2:=t;
                  fi;
             od;
            NR8:=[0,idx1,-idx2];
            ty:=1;
        fi;
        if UAdiff4=K and A[j][2]=k and  UAdiff6=-K and A[n][2]=-k then
           XX1:=Concatenation(["A",String(i)]);
           idx1:=0;
           idx2:=0;
           for t in [1..sA] do
                 if XX1=T[t] then
                     idx1:=t;
                 fi;
           od;
           NR8:=[0,idx1];
           ty:=1;
        fi;
      od;
   od;
fi;
if -k in A[i][1] and not (k in A[i][1]) and A[i][2]=-k then
      diff7:=Difference(L,[k]);
      diff9:=Difference(L,A[i][1]);
      K:=[-k];
      for j in [1..sA]do
          for n in [1..sA]do
               UA4:=SSortedList(A[j][1]);
              UAdiff8:=SSortedList(diff7);
               UA5:=SSortedList(A[n][1]);
              UAdiff10:=SSortedList(diff9);
               if UAdiff8=UA4 and  UAdiff10=UA5 and  A[j][2]=-k
                   and A[n][2]=k then
                   XX1:=Concatenation(["A",String(i)]);
                   XX2:=Concatenation(["A",String(j)]);
                   XX3:=Concatenation(["A",String(n)]);
                   idx1:=0;
                   idx2:=0;
```

```
    idx3:=0;
    for t in [1..sA] do
          if XX1=T[t] then
              idx1:=t;
           fi;
           if XX2=T[t] then
               idx2:=t;
           fi;
           if XX3=T[n] then
               idx3:=n;
           fi;
    od;
    NR8:=[0,idx1,-idx3,-idx2];
    ty:=1;
fi;
if UAdiff8=K and A[j][2]=-k and  UAdiff10=UA5 and A[n][2]=k then
    XX1:=Concatenation(["A",String(i)]);
    XX2:=Concatenation(["A",String(n)]);
    idx1:=0;
    idx2:=0;
    for t in [1..sA] do
          if XX1=T[t] then
              idx1:=t;
          fi;
          if XX2=T[n] then
              idx2:=n;
          fi;
    od;
    NR8:=[0,idx1,-idx2];
    ty:=1;
fi;
if  UAdiff8=UA4 and A[j][2]=-k and   UAdiff10=-K and A[n][2]=k then
    XX1:=Concatenation(["A",String(i)]);
    XX2:=Concatenation(["A",String(j)]);
    idx1:=0;
    idx2:=0;
    for t in [1..sA] do
          if XX1=T[t] then
              idx1:=t;
          fi;
          if XX2=T[t] then
              idx2:=t;
          fi;
     od;
     NR8:=[0,idx1,-idx2];
    ty:=1;
fi;
if UAdiff8=K and A[j][2]=-k and   UAdiff10=-K and A[n][2]=k then
    XX1:=Concatenation(["A",String(i)]);
    idx1:=0;
    idx2:=0;
    for t in [1..sA] do
          if XX1=T[t] then
```

```
                                        idx1:=t;
                                    fi;
                                od;
                                NR8:=[0,idx1];
                                ty:=1;
                            fi;
                        od;
                    od;
                fi;
                if ty=1 then
                    Add(Rels,NR8);
                    R8:=[];
                    NR8:=[];
                    ty:=0;
                fi;
            od;
        fi;
        ##
        ##############################################################################
        ##
od;
sRels:=Size(Rels);
return([Rels,sRels]);
end;
```

14. APCGRelationR9 Function

```
APCGRelationR9:=function(V,A,T,Lk,Rels)
local k,j,i,zx,IntA,UniA,NUniA,l,K,t,UA13,UA14,UA16,UA23,UA24,UA25,UA26,
R2,XX1,XX2,XX3,idx1,idx2,idx3,t1,R9,R9a,R9b,R9c,invLk1,srels,sRels,diff13,
diff14,diff15,diff16,diff23,diff24,diff25,diff26,UAdiff16,UAdiff24,UAdiff23,
sV,sA,UAdiff25,UAdiff13,UAdiff14,UAdiff26,n,invV,L,invLk2,invLk3,UniLk;
##
##############################################################################
##
## The input of this function are:
### V: the list of vertices of the graph zeta,
### A: the list of type(2) generators computed in "WhiteheadAutomorphismsOfSecondType",
### T: list of the names of elements of A,
### Lk: the list of links computed in "StarLinkDominateOfVertex".
### Rels: the list of row matrices of indices of the relations (it is one
### of the outputs of the "APCGRelationR4",
### Note that in order to get just the row matrices of indices of relation (R9)
### we need to pass an empty list [] rather than the list Rels above.
##
## It computes the list of indices of the generators [0,idx1,idx2,-idx1,-idx2]
## of relators of type (R9) of the group Aut(G_zeta) by satisfying the conditions
## of this relations and add them to the list Rels. In addition it calculates
## the size of the list Rels.
## It returns [Rels,sRels].
##############################################################################
```

```
##
sV:=Size(V);
sA:=Size(A);
invV:=-V;                   # invV is the inverses list of the vertex list V
L:=Concatenation(V,invV);   # L is the union of the lists V and invV
for k in [1..sV] do         # loop through the vertex list V
    for j in [1..sV] do     # loop through the vertex list V
       ##
       ##########################################################################
       ##
       ## In this part we compute the list of indices When Lk(k) is not empty list.
       ##
       if Lk[j]<>[0] then
          for i in [1..sA]do # loop through A the Type (2) Whitehead Automorphisms
             ######################################################################
             ##
             ## In this section we compute first part of the list of indices of the
             ## generators which is [0,idx1,idx2,-idx1,-idx2] of the relators of type
             ## (R9) when the multiplier "a" (k in this code) of the automorphism
             ## (A,a) is the original vertex "a" (not the inverse of the vertex "a")
             ## and zx=L(j) as defined below by satisfying the conditions of this
             ## relations.
             ## 0: is flag to let us know that all the generators here of power 1.
             ## idx1: represents the index "i" of a specific generator A(i) of A.
             ## idx2: represents the index "n" of a specific generator A(n) of A.
             ## -idx1: represents the inverse of the specific generator A(i) of
             ## A which corresponds to the index idx1.
             ## -idx2: represents the inverse of the specific generator A(n) of
             ## A which corresponds to the index idx2.
             ## For example if [0,idx1,idx2,-idx1,-idx2]= [ 0, 9, 5, -9, -5 ] then
             ## this means that A9*A5*A9^-1*A5^-1=1.
             ##
             zx:=L[j];  # Here zx represents the vertices "b" (R9) of the graph zeta
             if k in A[i][1] and not (-k in A[i][1]) and A[i][2]=k and not
                 (zx in A[i][1]) and not (-zx in A[i][1]) then
                 diff15:=Difference(L,[-zx]);
                 invLk2:=-Lk[j];
                 UniLk:=Concatenation(Lk[j],invLk2);
                 diff16:=[];
                 for l in Lk[j] do
                     # In this loop if the vertex l and its inverse -l in the
                     # same time are belong to the list diff15 then we delete
                     # them, because they will cancel each other
                     if l in diff15 and -l in diff15 then
                        diff16:=Difference(diff15,[-l,l]);
                        diff15:=diff16;
                     fi;
                 od;
                 for n in [1..sA]do  # loop through A the Type (2) Whitehead Automorphisms
                     UA16:=SSortedList(A[n][1]);
                    UAdiff16:=SSortedList(diff16);
                     if A[n][2]=zx then
                         if UA16=UAdiff16 and diff16<>[zx] then
```

```
                XX1:=Concatenation(["A",String(i)]);
                # XX1: represents a specific automorphism (A,a) of A
                XX2:=Concatenation(["A",String(n)]);
                # XX2: represents a specific automorphism (L-b^-1, b) of A
                idx1:=0;
                idx2:=0;
                for t in [1..sA] do
                      # Verify the indices of the given Whitehead
                      # automorphisms A(i) and A(n) in A
                      if XX1=T[t] then
                          idx1:=t;
                      fi;
                      if XX2=T[t] then
                          idx2:=t;
                      fi;
                od;
                Add(Rels,[0,idx1,idx2,-idx1,-idx2]);
            fi;
   fi;
    od;
fi;
##
#######################################################################
##
#######################################################################
##
## In this section we compute second part of the list of indices of the
## generators which is [0,idx1,idx2,-idx1,-idx2] of the relators of type
## (R9) when the multiplier "a" (k in this code) of the automorphism
## (A,a) is the original vertex "a" (not the inverse of the vertex "a")
## and zx= -L(j) as
## defined below by satisfying the conditions of this relations.
## The procedure use in this Section is similar to the first Section
## above.
##
zx:=-L[j]; # Here zx represents the inverses of the vertices b above
if k in A[i][1] and not (-k in A[i][1])  and A[i][2]=k and not
     (zx in A[i][1]) and not (-zx in A[i][1]) then
     diff23:=Difference(L,[-zx]);
     invLk2:=-Lk[j];
     UniLk:=Concatenation(Lk[j],invLk2);
     diff24:=[];
     for l in Lk[j] do
         if l in diff23 and -l in diff23 then
            diff24:=Difference(diff23,[-l,l]);
            diff23:=diff24;
         fi;
     od;
     for n in [1..sA]do
         UA24:=SSortedList(A[n][1]);
        UAdiff24:=SSortedList(diff24);
         if A[n][2]=zx then
             if UA24=UAdiff24 and diff24<>[zx] then
```

```
                XX1:=Concatenation(["A",String(i)]);
                # XX1: represents a specific automorphism (A,a) of A
                XX2:=Concatenation(["A",String(n)]);
                # XX2: represents a specific automorphism (L-b^-1, b) of A
                idx1:=0;
                idx2:=0;
                for t in [1..sA] do
                    # Verify the indices of the given Whitehead
                    # automorphisms A(i) and A(n) in A
                    if XX1=T[t] then
                        idx1:=t;
                    fi;
                    if XX2=T[t] then
                        idx2:=t;
                    fi;
                od;
                Add(Rels,[0,idx1,idx2,-idx1,-idx2]);
            fi;
    fi;
    od;
fi;
##
######################################################################
##
######################################################################
##
## In this section we compute third part of the list of indices of the
## generators which is [0,idx1,idx2,-idx1,-idx2] of the relators of type
## (R9) when the multiplier "a" (k in this code) of the automorphism
## (A,a) is the inverse of the vertex "a" and zx= L(j) as defined below
## by satisfying the conditions of this relations.
## The procedure use in this Section is similar to the first Section above.
##
zx:=L[j]; # Here zx represents the vertices "b" (R9) of the graph zeta
if -k in A[i][1] and not (k in A[i][1])  and A[i][2]=-k and not
      (zx in A[i][1]) and not (-zx in A[i][1]) then
      diff13:=Difference(L,[-zx]);
      invLk3:=-Lk[j];
      UniLk:=Concatenation(Lk[j],invLk3);
      diff14:=[];
      for l in Lk[j] do
          if l in diff13 and -l in diff13 then
              diff14:=Difference(diff13,[-l,l]);
              diff13:=diff14;
          fi;
      od;
      for n in [1..sA]do
            if A[n][2]=zx then
                 UA14:=SSortedList(A[n][1]);
                UAdiff14:=SSortedList(diff14);
                 if UA14=UAdiff14 and diff14<>[zx] then
                       XX1:=Concatenation(["A",String(i)]);
                       XX2:=Concatenation(["A",String(n)]);
```

```
                    idx1:=0;
                    idx2:=0;
                    for t in [1..sA] do
                          if XX1=T[t] then
                              idx1:=t;
                          fi;
                          if XX2=T[t] then
                              idx2:=t;
                          fi;
                    od;
                    Add(Rels,[0,idx1,idx2,-idx1,-idx2]);
              fi;
         fi;
     od;
fi;
##
######################################################################
#
######################################################################
##
## In this section we compute third part of the list of indices of the
## generators which is [0,idx1,idx2,-idx1,-idx2] of the relators of type
## (R9) when the multiplier "a" (k in this code) of the automorphism
## (A,a) is the inverse of the vertex "a" and zx= -L(j) as defined
## below by satisfying the conditions of this relations.
## The procedure use in this Section is similar to the first Section above.
##
zx:=-L[j]; # Here zx represents the inverses of the vertices b above
if -k in A[i][1] and not (k in A[i][1]) and A[i][2]=-k and not
      (zx in A[i][1]) and not (-zx in A[i][1]) then
      diff25:=Difference(L,[-zx]);
      invLk3:=-Lk[j];
      UniLk:=Concatenation(Lk[j],invLk3);
      diff26:=[];
      for l in Lk[j] do
          if l in diff25 and -l in diff25 then
             diff26:=Difference(diff25,[-l,l]);
             diff25:=diff26;
          fi;
      od;
      for n in [1..sA]do
          if A[n][2]=zx then
              UA26:=SSortedList(A[n][1]);
             UAdiff26:=SSortedList(diff26);
              if UA26=UAdiff26 and diff26<>[zx] then

                    XX1:=Concatenation(["A",String(i)]);
                    XX2:=Concatenation(["A",String(n)]);
                    idx1:=0;
                    idx2:=0;
                    for t in [1..sA] do
                          if XX1=T[t] then
                              idx1:=t;
```

```
                                fi;
                                if XX2=T[t] then
                                    idx2:=t;
                                fi;
                          od;
                          Add(Rels,[0,idx1,idx2,-idx1,-idx2]);
                    fi;
                fi;
            od;
        fi;
    od;
fi;
##
########################################################################
##
## End the first part when Lk(j) is not empty list
##
########################################################################
##
########################################################################
##
## In this part we compute the list of indices When Lk(j) is empty list
## which isthe same procedure of first part when Lk(j) is not empty list
## with some changes.
##
if Lk[j]=[0] then
     for i in [1..sA]do
     zx:=L[j];
     if k in A[i][1] and not (-k in A[i][1]) and A[i][2]=k and not
          (zx in A[i][1]) and not (-zx in A[i][1]) then
          diff16:=Difference(L,[-zx]);
          for n in [1..sA]do
              UA16:=SSortedList(A[n][1]);
             UAdiff16:=SSortedList(diff16);
              if A[n][2]=zx then
                  if UA16=UAdiff16 and diff16<>[zx] then
                        XX1:=Concatenation(["A",String(i)]);
                        XX2:=Concatenation(["A",String(n)]);
                        idx1:=0;
                        idx2:=0;
                        for t in [1..sA] do
                              if XX1=T[t] then
                                  idx1:=t;
                              fi;
                              if XX2=T[t] then
                                  idx2:=t;
                              fi;
                        od;
                        Add(Rels,[0,idx1,idx2,-idx1,-idx2]);
                  fi;
          fi;
          od;
      fi;
```

```
zx:=-L[j];
if k in A[i][1] and not (-k in A[i][1]) and A[i][2]=k and not
   (zx in A[i][1]) and not (-zx in A[i][1]) then
   diff24:=Difference(L,[-zx]);
   for n in [1..sA]do
       UA24:=SSortedList(A[n][1]);
      UAdiff24:=SSortedList(diff24);
       if A[n][2]=zx then
            if UA24=UAdiff24 and diff24<>[zx] then
                  XX1:=Concatenation(["A",String(i)]);
                  XX2:=Concatenation(["A",String(n)]);
                  idx1:=0;
                  idx2:=0;
                  for t in [1..sA] do
                        if XX1=T[t] then
                            idx1:=t;
                         fi;
                         if XX2=T[t] then
                             idx2:=t;
                         fi;
                  od;
                  Add(Rels,[0,idx1,idx2,-idx1,-idx2]);
            fi;
   fi;
   od;
fi;
zx:=L[j];
if -k in A[i][1] and not (k in A[i][1])  and A[i][2]=-k and not
     (zx in A[i][1]) and not (-zx in A[i][1]) then
     diff14:=Difference(L,[-zx]);
     for n in [1..sA]do
         if A[n][2]=zx then
            UA14:=SSortedList(A[n][1]);
           UAdiff14:=SSortedList(diff14);
            if UA14=UAdiff14 and diff14<>[zx] then
                  XX1:=Concatenation(["A",String(i)]);
                  XX2:=Concatenation(["A",String(n)]);
                  idx1:=0;
                  idx2:=0;
                  for t in [1..sA] do
                        if XX1=T[t] then
                            idx1:=t;
                        fi;
                        if XX2=T[t] then
                            idx2:=t;
                        fi;
                   od;
                   Add(Rels,[0,idx1,idx2,-idx1,-idx2]);
            fi;
         fi;
     od;
 fi;
 zx:=-L[j];
```

```
            if -k in A[i][1] and not (k in A[i][1])  and A[i][2]=-k and not
                  (zx in A[i][1]) and not (-zx in A[i][1]) then
                  diff26:=Difference(L,[-zx]);
                  for n in [1..sA]do
                     if A[n][2]=zx then
                        UA26:=SSortedList(A[n][1]);
                       UAdiff26:=SSortedList(diff26);
                        if UA26=UAdiff26 and diff26<>[zx] then

                              XX1:=Concatenation(["A",String(i)]);
                              XX2:=Concatenation(["A",String(n)]);
                              idx1:=0;
                              idx2:=0;
                              for t in [1..sA] do
                                    if XX1=T[t] then
                                        idx1:=t;
                                    fi;
                                    if XX2=T[t] then
                                        idx2:=t;
                                    fi;
                               od;
                               Add(Rels,[0,idx1,idx2,-idx1,-idx2]);
                        fi;
                    fi;
                od;
            fi;
        od;
      fi;
      ##
      ## End the second part when Lk(j) is empty list
      ##
      #################################################################
      ##
   od;
od;
sRels:=Size(Rels);
return([Rels,sRels]);
end;
```

15. APCGRelationR10 Function

```
APCGRelationR10:=function(V,A,T,Lk,Rels)
local k,j,i,m,zx,IntA,UniA,NUniA,l,K,t,UA13,UA14,UA16,UA23,UA24,UA25,UA26,
UA27,UA28,R2,XX1,XX2,XX3,idx1,idx2,idx3,t1,R10,R10a,R10b,R10c,invLk1,srels,
sRels,diff13,diff14,diff15,diff16,diff23,diff24,diff25,diff26,diff27,diff28,
UAdiff16,UAdiff24,UAdiff23,UAdiff25,UAdiff13,UAdiff14,UAdiff26,UAdiff27,
sV,sA,UAdiff28,n,invV,L,invLk2,invLk3,UniLk;
##
#############################################################################
##
## The input of this function are:
```

```
### V: the list of vertices of the graph zeta,
### A: the list of type(2) generators computed in "WhiteheadAutomorphismsOfSecondType",
### T: list of the names of elements of A,
### Lk: the list of links computed in "StarLinkDominateOfVertex".
### Rels: the list of row matrices of indices of the relations (it is one
### of the outputs of the "APCGRelationR4",
### Note that in order to get just the row matrices of indices of relation (R9)
### we need to pass an empty list [] rather than the list Rels above.
##
## It computes the list of indices of the generators [0,idx1,idx2,-idx1,-idx2,-idx3]
## of relators of type (R10) of the group Aut(G_zeta) by satisfying the conditions
## of this relations and add them to the list Rels. In addition it calculates
## the size of the list Rels.
## It returns [Rels,sRels].
##############################################################################
##
sV:=Size(V);
sA:=Size(A);
invV:=-V;                 # invV is the inverses list of the vertex list V
L:=Concatenation(V,invV); # L is the union of the lists V and invV
for k in [1..sV] do       # loop through the vertex list V
   for j in [1..sV] do    # loop through the vertex list V
      ##
      ########################################################################
      ##
      ## In this part we compute the list of indices When Lk(k) is not empty list.
      ##
      if Lk[j]<>[0] then
         for i in [1..sA]do  # loop throu A the Type (2) Whitehead Automorphisms
            ##################################################################
            ##
            ## In this section we compute first part of the list of indices of the
            ## generators which is [0,idx1,idx2,-idx1,-idx2,-idx3] of the relators
            ## of type (R10) when the multiplier "a" (k in this code) of the
            ## automorphism (A,a) is the original vertex "a" (not the inverse of
            ## the vertex "a"), and the  multiplier "b" (j in this code) of the
            ## automorphism (L-b^-1, b) is the original vertex "b" and k not equal
            ## to j, by satisfying the conditions of this relations.
            ## 0: is just flag to let us know that all generators here of power 1.
            ## idx1: represents the index "i" of a specific generator A(i) of A.
            ## idx2: represents the index "n" of a specific generator A(n) of A.
            ## -idx1: represents the inverse of the specific generator A(i) of A
            ## which corresponds to the index idx1.
            ## -idx2: represents the inverse of the specific generator A(n) of A
            ## which corresponds to the index idx2.
            ## -idx3: represents the inverse of the specific generator A(m) of A
            ## which corresponds to the index idx3.
            ## For example if [0,idx1,idx2,-idx1,-idx2,-idx3]= [0,1,27,-1,-27,-5],
            ## then this means that A1*A27*A1^-1*A27^-1*A5^-1=1.
            ##
            if k in A[i][1] and not (-k in A[i][1]) and A[i][2]=k  and j in A[i][1]
                and not (-j in A[i][1]) and k<>j then
                diff15:=Difference(L,[-j]);
```

```
invLk2:=-Lk[j];
UniLk:=Concatenation(Lk[j],invLk2);
# UniLk: represents the link of the vertex "j" with respect to L
diff16:=[];
for l in Lk[j] do # In this loop if the vertex l and its inverse -l in the
                  # same time are belong to the list diff15 then we delete
                  # them, because they will cancel each other
    if l in diff15 and -l in diff15 then
       diff16:=Difference(diff15,[-l,l]);
       diff15:=diff16;
    fi;
od;
diff27:=Difference(L,[-k]);
invLk3:=-Lk[k];
UniLk:=Concatenation(Lk[k],invLk3);
diff28:=[];
for l in Lk[j] do
    if l in diff27 and -l in diff27 then
       diff28:=Difference(diff27,[-l,l]);
       diff27:=diff28;
    fi;
od;
for n in [1..sA]do
    UA16:=SSortedList(A[n][1]);
   UAdiff16:=SSortedList(diff16);
    for m in [1..sA]do
        UA28:=SSortedList(A[m][1]);
       UAdiff28:=SSortedList(diff28);
        if A[n][2]=j and A[m][2]=k  then
            if UA16=UAdiff16 and diff16<>[j] and UA28=UAdiff28 then
               XX1:=Concatenation(["A",String(i)]);
               # XX1: represents a specific automorphism (A,a) of A
               XX2:=Concatenation(["A",String(n)]);
               # XX2: represents a specific automorphism (L-b^-1, b) of A
               XX3:=Concatenation(["A",String(m)]);
               # XX3: represents a specific automorphism (L-a^-1, a) of A
               idx1:=0;
               idx2:=0;
               idx3:=0;
               for t in [1..sA] do
                   if XX1=T[t] then
                       idx1:=t;
                   fi;
                   if XX2=T[t] then
                       idx2:=t;
                   fi;
                   if XX3=T[t] then
                       idx3:=t;
                   fi;;
               od;
               Add(Rels,[0,idx1,idx2,-idx1,-idx2,-idx3]);
            fi;
    fi;
```

```
        od;
    od;
fi;
##
######################################################################
##
######################################################################
##
## In this section we compute second part of the list of indices of the
## generators which is [0,idx1,idx2,-idx1,-idx2,-idx3] of the relators
## of type (R10) when the multiplier "a" (k in this code) of the
## automorphism (A,a) is the original vertex "a" (not the inverse of
## the vertex "a") and the multiplier "b" (j in this code) of the
## automorphism (L-b^-1, b) is the the inverse of the vertex "b"
## (-j in this code) and k not equal to -j by satisfying the
## conditions of this relations.
## The procedure use in this Section is similar to the first Section above.
##
if k in A[i][1] and not (-k in A[i][1]) and A[i][2]=k and -j in A[i][1]
   and not (j in A[i][1]) and k<> -j then
   diff15:=Difference(L,[j]);
   invLk2:=-Lk[j];
   UniLk:=Concatenation(Lk[j],invLk2);
   diff16:=[];
   for l in Lk[j] do
       if l in diff15 and -l in diff15 then
          diff16:=Difference(diff15,[-l,l]);
          diff15:=diff16;
       fi;
   od;
   diff27:=Difference(L,[-k]);
   invLk3:=-Lk[k];
   UniLk:=Concatenation(Lk[k],invLk3);
   diff28:=[];
   for l in Lk[j] do
       if l in diff27 and -l in diff27 then
          diff28:=Difference(diff27,[-l,l]);
          diff27:=diff28;
       fi;
   od;
   for n in [1..sA]do
         UA16:=SSortedList(A[n][1]);
        UAdiff16:=SSortedList(diff16);
         for m in [1..sA]do
             UA28:=SSortedList(A[m][1]);
            UAdiff28:=SSortedList(diff28);
             if A[n][2]=-j and A[m][2]=k then
                   if UA16=UAdiff16 and diff16<>[-j] and UA28=UAdiff28 then
                         XX1:=Concatenation(["A",String(i)]);
                         XX2:=Concatenation(["A",String(n)]);
                         XX3:=Concatenation(["A",String(m)]);
                         idx1:=0;
                         idx2:=0;
```

```
                    idx3:=0;
                    for t in [1..sA] do
                        if XX1=T[t] then
                            idx1:=t;
                        fi;
                        if XX2=T[t] then
                            idx2:=t;
                        fi;
                        if XX3=T[t] then
                            idx3:=t;
                        fi;
                    od;
                    Add(Rels,[0,idx1,idx2,-idx1,-idx2,-idx3]);
                fi;
            fi;
        od;
    od;
fi;
##
#######################################################################
##
#######################################################################
##
## In this section we compute third part of the list of indices of the
## generators which is [0,idx1,idx2,-idx1,-idx2,-idx3] of the relators
## of type (R10) when the multiplier "a" (k in this code) of the
## automorphism (A,a) is the inverse of the vertex "a" (-k in this code)
## and the multiplier "b" (j in this code) of the automorphism (L-b^-1, b)
is the original vertex "b" and -k not equal to j by satisfying the
## conditions of this relations.
## The procedure use in this Section is similar to the first Section above.
##
if -k in A[i][1] and not (k in A[i][1]) and A[i][2]=-k
   and j in A[i][1] and not (-j in A[i][1]) and -k<>j then
   diff15:=Difference(L,[-j]);
   invLk2:=-Lk[j];
   UniLk:=Concatenation(Lk[j],invLk2);
   diff16:=[];
   for l in Lk[j] do
       if l in diff15 and -l in diff15 then
          diff16:=Difference(diff15,[-l,l]);
          diff15:=diff16;
       fi;
   od;
   diff27:=Difference(L,[k]);
   invLk3:=-Lk[k];
   UniLk:=Concatenation(Lk[k],invLk3);
   diff28:=[];
   for l in Lk[j] do
       if l in diff27 and -l in diff27 then
          diff28:=Difference(diff27,[-l,l]);
          diff27:=diff28;
       fi;
```

```
   od;
   for n in [1..sA]do
         UA16:=SSortedList(A[n][1]);
        UAdiff16:=SSortedList(diff16);
         for m in [1..sA]do
               UA28:=SSortedList(A[m][1]);
              UAdiff28:=SSortedList(diff28);
               if A[n][2]=j and A[m][2]=-k  then
                  if UA16=UAdiff16 and diff16<>[j] and UA28=UAdiff28 then
                         XX1:=Concatenation(["A",String(i)]);
                         XX2:=Concatenation(["A",String(n)]);
                         XX3:=Concatenation(["A",String(m)]);
                         idx1:=0;
                         idx2:=0;
                         idx3:=0;
                         for t in [1..sA] do
                             if XX1=T[t] then
                                 idx1:=t;
                             fi;
                             if XX2=T[t] then
                                 idx2:=t;
                             fi;
                             if XX3=T[t] then
                                 idx3:=t;
                             fi;
                         od;
                         Add(Rels,[0,idx1,idx2,-idx1,-idx2,-idx3]);
                    fi;
          fi;
         od;
    od;
fi;
##
######################################################################
##
######################################################################
##
## In this section we compute third part of the list of indices of the
## generators which is [0,idx1,idx2,-idx1,-idx2,-idx3] of the relators
## of type (R10) when the multiplier "a" (k in this code) of the
## automorphism (A,a) is the inverse of the vertex "a" (-k in this code)
## and the multiplier "b" (j in this code) of the automorphism (L-b^-1, b)
## is the inverse of the  vertex "b" (-j in this code) and -k not equal
## to -j by satisfying the conditions of this relations.
## The procedure use in this Section is similar to the first Section above.
##
if -k in A[i][1] and not (k in A[i][1]) and A[i][2]=-k  and -j in A[i][1]
   and not (j in A[i][1]) and -k <> -j then
   diff15:=Difference(L,[j]);
   invLk2:=-Lk[j];
   UniLk:=Concatenation(Lk[j],invLk2);
   diff16:=[];
   for l in Lk[j] do
```

```
            if l in diff15 and -l in diff15 then
               diff16:=Difference(diff15,[-l,l]);
               diff15:=diff16;
            fi;
        od;
        diff27:=Difference(L,[k]);
        invLk3:=-Lk[k];
        UniLk:=Concatenation(Lk[k],invLk3);
        diff28:=[];
        for l in Lk[j] do
            if l in diff27 and -l in diff27 then
               diff28:=Difference(diff27,[-l,l]);
               diff27:=diff28;
            fi;
        od;
        for n in [1..sA]do
              UA16:=SSortedList(A[n][1]);
             UAdiff16:=SSortedList(diff16);
              for m in [1..sA]do
                    UA28:=SSortedList(A[m][1]);
                   UAdiff28:=SSortedList(diff28);
                    if A[n][2]=-j and A[m][2]=-k  then
                       if UA16=UAdiff16 and diff16<>[-j] and UA28=UAdiff28 then
                              XX1:=Concatenation(["A",String(i)]);
                              XX2:=Concatenation(["A",String(n)]);
                              XX3:=Concatenation(["A",String(m)]);
                              idx1:=0;
                              idx2:=0;
                              idx3:=0;
                              for t in [1..sA] do
                                  if XX1=T[t] then
                                      idx1:=t;
                                  fi;
                                  if XX2=T[t] then
                                      idx2:=t;
                                  fi;
                                  if XX3=T[t] then
                                      idx3:=t;
                                  fi;
                              od;
                              Add(Rels,[0,idx1,idx2,-idx1,-idx2,-idx3]);
                         fi;
                fi;
              od;
         od;
     fi;
   od;
fi;
##
#######################################################################
##
## End the first part when Lk(j) is not empty list
##
```

```
#####################################################################
##
#####################################################################
##
## In this part we compute the list of indices When Lk(j) is empty list
## which is the same procedure of first part when Lk(j) is not empty list
## with some changes.
##
if Lk[j]=[0] then
  for i in [1..sA]do
     if k in A[i][1] and not (-k in A[i][1]) and A[i][2]=k
          and j in A[i][1] and not (-j in A[i][1]) and k<>j then
          diff15:=Difference(L,[-j]);
          invLk2:=-Lk[j];
          UniLk:=Concatenation(Lk[j],invLk2);
          diff16:=Difference(diff15,UniLk);
          diff27:=Difference(L,[-k]);
          invLk3:=-Lk[k];
          UniLk:=Concatenation(Lk[k],invLk3);
          diff28:=Difference(diff27,UniLk);
          for n in [1..sA]do
                UA16:=SSortedList(A[n][1]);
               UAdiff16:=SSortedList(diff16);
                for m in [1..sA]do
                      UA28:=SSortedList(A[m][1]);
                     UAdiff28:=SSortedList(diff28);
                      if A[n][2]=j and A[m][2]=k  then
                           if UA16=UAdiff16 and UA28=UAdiff28 then
                                XX1:=Concatenation(["A",String(i)]);
                                XX2:=Concatenation(["A",String(n)]);
                                XX3:=Concatenation(["A",String(m)]);
                                idx1:=0;
                                idx2:=0;
                                idx3:=0;
                                for t in [1..sA] do
                                    if XX1=T[t] then
                                        idx1:=t;
                                    fi;
                                    if XX2=T[t] then
                                        idx2:=t;
                                    fi;
                                    if XX3=T[t] then
                                        idx3:=t;
                                    fi;
                                od;
                                Add(Rels,[0,idx1,idx2,-idx1,-idx2,-idx3]);
                           fi;
                 fi;
                od;
           od;
       fi;
       if k in A[i][1] and not (-k in A[i][1]) and A[i][2]=k
          and -j in A[i][1] and not (j in A[i][1]) and k<> -j then
```

```
   diff15:=Difference(L,[j]);
   invLk2:=-Lk[j];
   UniLk:=Concatenation(Lk[j],invLk2);
   diff16:=Difference(diff15,UniLk);
   diff27:=Difference(L,[-k]);
   invLk3:=-Lk[k];
   UniLk:=Concatenation(Lk[k],invLk3);
   diff28:=Difference(diff27,UniLk);
   for n in [1..sA]do
         UA16:=SSortedList(A[n][1]);
        UAdiff16:=SSortedList(diff16);
         for m in [1..sA]do
             UA28:=SSortedList(A[m][1]);
            UAdiff28:=SSortedList(diff28);
             if A[n][2]=-j and A[m][2]=k then
                 if UA16=UAdiff16 and UA28=UAdiff28 then
                         XX1:=Concatenation(["A",String(i)]);
                         XX2:=Concatenation(["A",String(n)]);
                         XX3:=Concatenation(["A",String(m)]);
                         idx1:=0;
                         idx2:=0;
                         idx3:=0;
                         for t in [1..sA] do
                             if XX1=T[t] then
                                 idx1:=t;
                             fi;
                             if XX2=T[t] then
                                 idx2:=t;
                             fi;
                             if XX3=T[t] then
                                 idx3:=t;
                             fi;
                         od;
                         Add(Rels,[0,idx1,idx2,-idx1,-idx2,-idx3]);
                 fi;
             fi;
        od;
   od;
fi;
if -k in A[i][1] and not (k in A[i][1]) and A[i][2]=-k
   and j in A[i][1] and not (-j in A[i][1]) and -k<>j then
   diff15:=Difference(L,[-j]);
   invLk2:=-Lk[j];
   UniLk:=Concatenation(Lk[j],invLk2);
   diff16:=Difference(diff15,UniLk);
   diff27:=Difference(L,[k]);
   invLk3:=-Lk[k];
   UniLk:=Concatenation(Lk[k],invLk3);
   diff28:=Difference(diff27,UniLk);
   for n in [1..sA]do
         UA16:=SSortedList(A[n][1]);
        UAdiff16:=SSortedList(diff16);
         for m in [1..sA]do
```

```
            UA28:=SSortedList(A[m][1]);
           UAdiff28:=SSortedList(diff28);
            if A[n][2]=j and A[m][2]=-k  then
                  if UA16=UAdiff16 and UA28=UAdiff28 then
                      XX1:=Concatenation(["A",String(i)]);
                      XX2:=Concatenation(["A",String(n)]);
                      XX3:=Concatenation(["A",String(m)]);
                      idx1:=0;
                      idx2:=0;
                      idx3:=0;
                      for t in [1..sA] do
                          if XX1=T[t] then
                              idx1:=t;
                          fi;
                          if XX2=T[t] then
                              idx2:=t;
                          fi;
                          if XX3=T[t] then
                              idx3:=t;
                          fi;
                      od;
                      Add(Rels,[0,idx1,idx2,-idx1,-idx2,-idx3]);
                  fi;
      fi;
     od;
  od;
fi;
if -k in A[i][1] and not (k in A[i][1]) and A[i][2]=-k
   and -j in A[i][1] and not (j in A[i][1]) and -k <> -j then
   diff15:=Difference(L,[j]);
   invLk2:=-Lk[j];
   UniLk:=Concatenation(Lk[j],invLk2);
   diff16:=Difference(diff15,UniLk);
   diff27:=Difference(L,[k]);
   invLk3:=-Lk[k];
   UniLk:=Concatenation(Lk[k],invLk3);
   diff28:=Difference(diff27,UniLk);
   for n in [1..sA]do
         UA16:=SSortedList(A[n][1]);
        UAdiff16:=SSortedList(diff16);
         for m in [1..sA]do
               UA28:=SSortedList(A[m][1]);
              UAdiff28:=SSortedList(diff28);
               if A[n][2]=-j and A[m][2]=-k  then
                     if UA16=UAdiff16 and UA28=UAdiff28 then

                         XX1:=Concatenation(["A",String(i)]);
                         XX2:=Concatenation(["A",String(n)]);
                         XX3:=Concatenation(["A",String(m)]);
                         idx1:=0;
                         idx2:=0;
                         idx3:=0;
                         for t in [1..sA] do
```

```
                                           if XX1=T[t] then
                                               idx1:=t;
                                           fi;
                                           if XX2=T[t] then
                                               idx2:=t;
                                           fi;
                                           if XX3=T[t] then
                                               idx3:=t;
                                           fi;
                                       od;
                                       Add(Rels,[0,idx1,idx2,-idx1,-idx2,-idx3]);
                                   fi;
                         fi;
                        od;
                   od;
               fi;
           od;
        fi;
        ##
        ## End the second part when Lk(j) is empty list
        ##
        ####################################################################
        ##
   od;
od;
sRels:=Size(Rels);
return([Rels,sRels]);
end;
```

16. APCGFinalReturn Function

```
APCGFinalReturn:=function(gens,Rels,sRels,sRels1,Rels1,sgenss)
local i,j,j1,j2,C,F,rels,srels,GHK,KK,GGG,sgens,GHK1,KK1,ZZa,rels1,srels1;
##
############################################################################
##
## The input of this function are:
### gens: the list of the generators of the group Aut(G_zeta).
### Rels: the list of the indices of the relators which computed in
###       "RelationsOfGraphAutomorphisms", "APCGRelationR1",..., "APCGRelationR10"
### sRels: the size of the list Rels.
### Rels1: the list of the indices of the relators of graph group
###        which computed in "WhiteheadAutomorphismsOfFirstType".
### sRels1: the size of the list Rels1.
### sgenss: the size of the list genss which is the name of the i^th of
###         generator of the Whitehead automorphisms of Aut(G_zeta).
###         It computed in "WhiteheadAutomorphismsOfFirstType"
##
## It forms  the list of relations rels from the lists Rels and Rels1.
## In fact this function forms the output of the function
## FinitePresentationOfAutParCommGrp in the package AutParCommGrp.
```

```
##############################################################################
##
rels1:=[];
C:=gens;
F:=FreeGroup(C);            # computes the free group on gens. The generators
                            # are displayed as string.1, string.2, ..., string.n
gens:=GeneratorsOfGroup(F); # returns a list of generators gens of the free group F
sgens:=Size(gens);
#############################################################################
##
## In this section we form  the list of relations rels1 from the list Rels1
## (computed in the function WhiteheadAutomorphismsOfFirstType) and adds
## them to the list rels1, and then adds it to the list of relations rels.
##
for i in [1..sRels1] do
       GHK:=Size(Rels1[i]);
       GHK1:=GHK/2;  # To find the real length of each single relation
       j1:=1;
       for j in [1..GHK1] do
             KK:=sgenss+AbsoluteValue(Rels1[i][j1]); #function reading
             j2:=j1+1;
             KK1:=Rels1[i][j2]; # power
             if KK1 <> 1 then
               ZZa:=gens[KK]^KK1;
             else
                 ZZa:=gens[KK];
             fi;
             if j1=1 then
              rels1[i]:=ZZa;
             else
                 rels1[i]:=rels1[i]*ZZa;
             fi;
             j1:=j1+2;
       od;
od;
srels1:=Size(rels1);
##
##########################################################################
##
## In this section we form  the list of relations rels from the list Rels
## (computed in the functions RelationsOfGraphAutomorphisms, APCGRelationR1,
## APCGRelationR2,..., APCGRelationR10)
##
rels:=[];
for i in [1..sRels] do
  GHK:=Size(Rels[i]);
  KK:=AbsoluteValue(Rels[i][2]);
  if Rels[i][1] = 0 then
   rels[i]:=gens[KK];
fi;
  if Rels[i][1] = 1 then
     rels[i]:=gens[KK]^2;
fi;
```

```
  if Rels[i][1] = 2 then
     rels[i]:=gens[KK];
     GHK:=GHK-3;
fi;
  if Rels[i][2] < 0 then
   rels[i]:=rels[i]^-1;
fi;
  for j in [3..GHK] do
        KK:=AbsoluteValue(Rels[i][j]);
        if Rels[i][j] < 0 then
             rels[i]:=rels[i]*gens[KK]^-1;
        else
             rels[i]:=rels[i]*gens[KK];
    fi;
   od;
od;
srels:=Size(rels);
##
##########################################################################
##
for i in [1..srels1] do # This loop is to add the relations of graph
                        # automorphisms rels1 to final relations list rels
    j:=srels+i;
    rels[j]:=rels1[i];
od;
srels:=Size(rels);
GGG:=F/rels;            # computes the finitely presented group on
                        # the generators gens of F defined above
return([F,gens,rels,GGG,sgens,srels]);
end;
```

17. FinitePresentationOfAutParCommGrp Function

```
FinitePresentationOfAutParCommGrp:=function(V,E)
local R1,R2,R3,R4,R5,R6,R7,R8,R9,R10,R11,R12,R13,R14,St,Lk,YY,sV,
M,NV,NE,sNV,sNE,A,sA,gens,sgens,sgenss,Gens3,rels,srels,Rels,sRels,
relvalofF,srelvalofF,rels1,srels1,sGens2,F,GGG,sComps,Rels1,sRels1,
T,Q,i,j,tempedgex,tempedgey;
##
############################################################################
##
## The input of this function is  a simple graph zeta=(V,E), where V and E
## represent the set of vertices and the set of edges respectively.
##
## It returns [gens,rels,GGG], where
### gens: is a list of free generators of the automorphism group of
###       partially commutative group Aut(G_zeta).
### rels: is a list of relations in the generators of the free group.
###       Note that relations are entered as relators, i.e., as words
###       in the generators of the free group.
### GGG:=F/rels: is the automorphism group Aut(G_zeta) of G_zeta given
```

```
###             as a finite presentation group with generators gens
###             and relators rels.
##
## In fact, the main work of this function is to run all the functions
## we have read them below to give a finite presentation for automorphism
## groups Aut(G_zeta) of G_zeta.
##############################################################################
##
if IsSimpleGraph(V,E)=true then   # Call the function IsSimpleGraph to test
                                  # whether the graph zeta is simple or not
  ##
  ############################################################################
  ##
  ## This section is to compute the star St(v), link Lk(v) and the dominate
  ## list Y(v) of each pair of vertices v,u in V
  ##
  R1:=StarLinkDominateOfVertex(V,E);  #F StarLinkDominateOfVertex( <V>, <E>)
                                      ## return([St,Lk,YY,sV,M,L,sL]);
  St:=R1[1];
  Lk:=R1[2];
  YY:=R1[3];
  sV:=R1[4];
  M:=R1[5];
  ##
  ############################################################################
  ##
  ## This section is to delete the star St(v) of a specific vertex v
  ## from the graph zeta
  ##
  R2:=DeleteVerticesFromGraph(St,V,E); #F DeleteverticesFromGraph( <St>, <V>, <E>)
                                       ## return([NV,NE,sNV,sNE]);
  NV:=R2[1];
  NE:=R2[2];
  sNV:=R2[3];
  sNE:=R2[4];
  ##
  ############################################################################
  ##
  ############################################################################
  ##
  ## This section is to compute the type (2) Whitehead automorphisms
  ##
  R3:=WhiteheadAutomorphismsOfSecondType(NV,NE,St,YY);
            #F WhiteheadAutomorphismsOfSecondType( <NV>, <NE>, <St>, <YY> )
            ## return ([A,T,sA]);
  A:=R3[1];
  T:=R3[2];
  sA:=R3[3];
  ##
  ############################################################################
  ##
  ############################################################################
  ##
```

```
## This section is to compute the type (1) Whitehead automorphisms also to
## copute the generators of the group automorphism of graph and then find
## the generators of the automorphism group of partially commutative group
##
R4:=WhiteheadAutomorphismsOfFirstType(E,sV,sA,T);
            #F WhiteheadAutomorphismsOfFirstType( <E>, <sV>, <sA>, <T> )
            ## return([gens,sgens,sgenss,Gens3,relvalofF,srelvalofF,Rels1,sRels1,sGens2]);
gens:=R4[1];
sgens:=R4[2];
sgenss:=R4[3];
Gens3:=R4[4];
relvalofF:=R4[5];
srelvalofF:=R4[6];
Rels1:=R4[7];
sRels1:=R4[8];
sGens2:=R4[9];
##
############################################################################
##
############################################################################
##
## This section is to compute the relations related to the graph automorphisms
##
R5:=RelationsOfGraphAutomorphisms(sA,sgenss,relvalofF,sV,sGens2);
      #F RelationsOfGraphAutomorphisms( <sA>, <sgenss>, <relvalofF>, <sV>, <sGens2>)
      #  return([Rels,sRels]);
Rels:=R5[1];
sRels:=R5[2];
##
############################################################################
##
############################################################################
##
## This section is to compute the relation R5
##
R6:=APCGRelationR5(A,St,Lk,Rels,T);
                           #F APCGRelationR5( <A>, <St>, <Lk> <Rels>, <T> )
                           ## return([Rels,sRels]);
Rels:=R6[1];
sRels:=R6[2];
##
############################################################################
##
############################################################################
##
## This section is to compute the relation R1
##
R7:=APCGRelationR1(sV,A,T,Rels);  #F APCGRelationR1( <sV>, <A>, <T>, <Rels> )
                                  ## return([Rels,sRels]);
Rels:=R7[1];
sRels:=R7[2];
##
############################################################################
```

```
##
###########################################################################
##
## This section is to compute the relation R2
##
R8:=APCGRelationR2(A,T,Rels,St); #F APCGRelationR2( <A>, <T>, <Rels>, <St>)
                                 ## return([Rels,sRels]);
Rels:=R8[1];
sRels:=R8[2];
##
###########################################################################
##
###########################################################################
##
## This section is to compute the relation R3
##
R9:=APCGRelationR3(A,T,Lk,Rels); #F APCGRelationR3( <A>, <T>, <Lk>, <Rels>)
                                    ## return([Rels,sRels]);
Rels:=R9[1];
sRels:=R9[2];
##
###########################################################################
##
###########################################################################
##
## This section is to compute the relation R4
##

R10:=APCGRelationR4(A,T,Lk,Rels); #F APCGRelationR4( <A>, <T>, <Lk>, <Rels>)
                                  ## return([Rels,sRels]);
Rels:=R10[1];
sRels:=R10[2];
##
###########################################################################
##
###########################################################################
##
## This section is to compute the relation R8
##
R11:=APCGRelationR8(V,A,T,Lk,Rels); #F APCGRelationR8( <V>, <A>, <T>, <Lk>, <Rels>)
                                    ## return([Rels,sRels]);
Rels:=R11[1];
sRels:=R11[2];
##
###########################################################################
##
###########################################################################
##
## This section is to compute the relation R9
##
R12:=APCGRelationR9(V,A,T,Lk,Rels); #F APCGRelationR9( <V>, <A>, <T>, <Lk>, <Rels>)
                                     ## return([Rels,sRels]);
Rels:=R12[1];
```

```
  sRels:=R12[2];
  ##
  ############################################################################
  ##
  ############################################################################
  ##
  ## This section is to compute the relation R10
  ##

  R13:=APCGRelationR10(V,A,T,Lk,Rels);  #F APCGRelationR10( <V>, <A>, <T>, <Lk> <Rels> )
                                        ## return([Rels,sRels]);
  Rels:=R13[1];
  sRels:=R13[2];
  ##
  ###############################################################################
  ##
  ###############################################################################
  ##
  ## This section is to compute the final relations T from the matrix of
  ## indices of the generators and find the final return
  ##
  R14:=APCGFinalReturn(gens,Rels,sRels,sRels1,Rels1,sgenss);
               #F APCGFinalReturn( <gens>, <Rels>, <sRels>, <sRels1>, <Rels1>, <sgenss> )
               ## return([F,gens,rels,GGG,sgens,srels]);
  F:=R14[1];
  gens:=R14[2];
  rels:=R14[3];
  GGG:=R14[4];
  sgens:=R14[5];
  srels:=R14[6];
  ##
  ############################################################################
  ##
else
    return("The graph must be a simple graph");
fi;
return[gens,rels,GGG];
end;
```

18. TietzeTransformations Function

```
TietzeTransformations:=function(G)
local hom,H,R;
##
############################################################################
##
## The aim of this function is to simplify the presentation of the finitely,
## presented group G, i.e., to reduce the number of generators, the number
## of relators and the relator lengths.
## The input of this function is finite presentation of the group G.
##
```

```
## Returns a group H isomorphic to G, so that the presentation of H,
## has been simplified using Tietze transformations.
##############################################################################
##
hom:= IsomorphismSimplifiedFpGroup(G);  # To find a homomorphism (an isomorphism).
H:= Image(hom);                         # Image( map ) is the image of the general
                                        # mapping map, i.e., the subset of elements
                                        # of the range of map that are actually values
                                        # of map. Note that in this case the argument
                                        # may also be multi-valued.
R:= RelatorsOfFpGroup(H);   # returns the relators of the finitely presented group
                            # G as words in the free generators provided by the
                            # FreeGeneratorsOfFpGroup value of G.
return[H,R];
end;
```

A.2 Appendix to Chapter 3

In this appendix we will attached the codes for all the functions we have written in Chapter 3 as follows:

1. **StarLinkOfVertex Function**

```
StarLinkOfVertex:=function(V,E)
local i,j,x1,M,sV,sE,tempx,St,indx1,Lk,indx2,x,YY,Y1,Y2,tempedgex,tempedgey;
##
############################################################################
##
## The input of this function is a finite simple graph zeta=(V,E), where V and
## E represents the list of vertices and the list of Edges respectivly.
##
## It computes the star St(v) and the link Lk(v) and concatenates them in
## two separate lists St and Lk respectively.
##############################################################################
##
if IsSimpleGraph(V,E)=true then     # Call the function IsSimpleGraph to test
                                    # whether the graph zeta is simple or not
  sV:=Length(V);
  M:= Length(E);
  St:= NullMat(sV,1,0);
  for i in [1..sV] do               # loop through the vertices V
      tempx:=V[i];
      indx1:=1;                     # index for the star of specific vertex v
      St[tempx][indx1]:=tempx;      # St: is a two dimensional matrix, the rows
                                    # indices represent the vertices and the columns
                                    # indices represent the star of a specific vertex.
      for j in [1..M] do            # loop through the edges E
          if tempx=E[j][1] then     # determine whether the specific edge E[j][1]
                                    # is equal to the vertex tempx
             if E[j][1]<>E[j][2] then # excludes isolated vertices from the calculation
                indx1:=indx1+1;
                St[tempx][indx1]:=E[j][2]; # means that the vertex E[j][2] belonges to
                                           # the star of a specific vertex v
             fi;
          fi;
          if tempx=E[j][2] then       # This section is the same of the first section,
                                      # above just we replaced the first coordinate of
                                      # the edge E(j) by the second coordinate.
             if E[j][1]<>E[j][2] then
                indx1:=indx1+1;
                St[tempx][indx1]:=E[j][1];
             fi;
          fi;
      od;
  od;
  Lk:=[];
```

```
  for j in [1..sV] do   # loop through the list of vertices V
      Y2:=Set(St[j]);   # make the list of a specific star St(j) as an order set
      RemoveSet(Y2,j);  # remove the vertex v (j in this code) from the list Y2
      Add(Lk,Y2);
  od;
else
    return("The graph must be a simple graph");
fi;
return([St,Lk]);
end;
```

2. CombinationsOfConnectedComponents Function

```
CombinationsOfConnectedComponents:=function(Comps)
local i,C1,sC1,Y2,Y3,L2,U2,q,sY3,Y4,L4,sY4;
##
############################################################################
##
## The input of this function is the list of connected
## components Comps of the specified graph B.
##
## The output is the set of all combinations Y4 of the multiset Comps.
############################################################################
##
C1:=Combinations(Comps); # Call the function Combinations to construct a list
                         # called C1 of all combinations of the multiset Comps
sC1:=Size(C1);
##
############################################################################
##
## In this section: loop through the list C1 to construct a list called Y2.
## Each element l of C1 is a list of lists X1, ..., Xn. Call the Concatenation
## function to form a new list h from the element of X1, ..., Xn.
## Then add this list to Y2.
Y2:=[];
Y3:=[];
for q in [1..sC1] do
     L2:=Concatenation(C1[q]);
     U2:=SSortedList(L2); #sorting each element of L2
     Add(Y2,L2);
     Add(Y3,U2);
od;
##
############################################################################
##
sY3:=Size(Y3);
Y4:=[];
for i in [1..sY3] do # Loop through the list Y3 to construct a list Y4 by
                     # adding each element of Y3 not equal to empty set to Y4
    if Y3[i]<>[] then
       Add(Y4,Y3[i]);
```

```
    fi;
od;
sY4:=Size(Y4);
return([Y3,Y4,sY4]);
end;
```

3. GeneratorsOfSubgroupConj Function

```
GeneratorsOfSubgroupConj:=function(NE,NV,V)
local i,j,gens2,gens,genss,rels,Rels,Bs,h,G2,G1,R3,R4,Comps,sComps,sMV,
sNE,UniA,D,DD,sD,S,YYY,NYY,invNYY,DYY,sDYY,Ls,t,xn,union_element,NCxY,
sgens,gens4,sgens4,gens3,sgens3,invV,sL,Y6,xs2,Y3,Y4,sY4,xs1,diff2,Y5,
sY5,sY6,sz,Y7,sY7,sxs2,xs3,sxs3,xs,sxs,Uxs,sUxs,CxY,sCxY,y9,y8,Y,sY,sBs,
Y8,sY8,y19,x11,sxs1,k,f,sf,gens1,sgens1,CxY1,sCxY1,y10,y99,NCY,KK,HH,L;
##
##############################################################################
##
## The input of this function are:
### the list NE of all lists of edges of the subgraph zeta\St(v)
### the list NV of all lists of vertices of the subgraph zeta\St(v)
### the list V which is the list of vertices.
##
## It computes the list gens1 which form the type(1) generators
## (elementary partial conjugations) of the subgroup Conj(G_zeta)
## of the group Aut(G_zeta).
###############################################################################
##
gens:=[];
Bs:=[];
Y6:=[];
xs2:=[];
sNE:=Size(NE);
invV:=-V;                  # invV: is the inverses list of the vertex list V
L:=Concatenation(V,invV);  # L is the union of the lists V and invV
for h in [1..sNE]do #loop through the lists NV and NE since they have same size
      G2:=NE[h];
      G1:=NV[h];
      R3:=ConnectedComponentsOfGraph(G1,G2);
                       # computes the list of the Connected components
                       # for each subgraph (NV(h),NE(h))
      Comps:=R3[1];    # Comps: list of all components of (NV(h),NE(h))
      sComps:=R3[2];   # sComps: size of Comps

      R4:=CombinationsOfConnectedComponents(Comps);
                               # computes the list of the combinations
                               # of the list Comps
      Y3:=R4[1]; # Y3: list of all combinations of the list Comps (it will be list of list)
      Y4:=R4[2]; # Y4: it is Y3 after SSorted its elements and delete the empty elements
      sY4:=R4[3]; # sY4: size of Y4
      xs1:=[];
      for i in [1..sY4] do            # loop through the list Y4
```

```
        diff2:=Difference(L,Y4[i]); # computes the difference diff2 between the list
                                    # L and each elements (list) of the list Y4
        Add(xs1,diff2);             # add each diff2 to the new list xs1
    od;
    sxs1:=Size(xs1);
    ##
    ############################################################################
    ##
    ## In this section: loop through the list Y4 to construct a list called Y6.
    ## In order to do this first find the size sz of xs1(i). For each element l
    ## of xs1(i) concatenate elements of Y4(i)  with elements of l to give a list
    ## KK. Then form a listY5 of pairs HH; with entries (KK, l), for each element
    ## l of xs1(i). Then append Y5 to the list Y6.
    ##
    Y5:=[];
    for i in [1..sY4] do
         sz:=Size(xs1[i]);
         for j in [1..sz] do
             KK:=Concatenation(Y4[i],[xs1[i][j]]);
             HH:=[KK,xs1[i][j]];
             Add(Y5,HH);
         od;
    od;
    sY5:=Size(Y5);
    Add(Y6,Y5);
    ##
    ##########################################################################
    ##
    Add(xs2,xs1); # Make new list xs2, by adding xs1 to xs2. This step and tht
                  # next one are needed because there are two inner loops
    Add(Bs,Y3);   # Make new lists Bs, by adding Y3 to Bs
od;               # ending the loop through the lists NV and NE
sY6:=Size(Y6);
Y7:=Concatenation(Y6); # Compute the list Y7 by concatenating the dense
                       # list of lists Y6
sY7:=Size(Y7);
sxs2:=Size(xs2);
xs3:=Concatenation(xs2); # Compute the list xs3 by concatenating the dense
                         # list of lists xs2
sxs3:=Size(xs3);
xs:=[];
##
##############################################################################
##
## In this section: loop through the list xs3 to construct a list called xs by
## adding each non-empty entry of xs3 to xs, and calculate the size of xs.
for i in [1..sxs3] do
    if not (xs3[i] in xs) and xs3[i]<>[] then
        Add(xs,xs3[i]);
    fi;
od;
sxs:=Size(xs);
##
```

```
################################################################################
##
Uxs:=Union(xs);  # Call the function Union to construct a list called Uxs by
sUxs:=Size(Uxs); # computing the union of xs and calculates it size sUxs
CxY1:=[];
for i in [1..sY7] do   # Loop through the list Y7 to construct a list
                       # called CxY1 by adding each non-empty entry of
                       # Y7 to CxY1, and calculate its size sCxY1
    if not (Y7[i] in CxY1) and Y7[i]<>[] then
        Add(CxY1,Y7[i]);
    fi;
od;
sCxY1:=Size(CxY1);
CxY:=[];
for j in [1..sCxY1]do  # Loop through the list CxY1 to compute a list of
                       # the definitions CxY of the partial conjugations,
                       # with its size sCxY
      y9:=CxY1[j][2];
      y10:=CxY1[j][1];
      y99:=SSortedList(y10);
      NCY:=[y99,y9];
      Add(CxY,NCY);
od;
sCxY:=Size(CxY);
Y8:=Concatenation(Bs); # Make a list Y8 by concatenating the dense
                       # list of lists Bs defined above
sBs:=Size(Bs);
sY8:=Size(Y8);
Y:=[];
for i in [1..sY8] do   # Loop through the list Y8 to construct a list Y
                       # of the non-empty unions of connected components
                       # of zeta\St(v)
    if not (Y8[i] in Y) and Y8[i]<>[] then
        Add(Y,Y8[i]);
    fi;
od;
sY:=Size(Y);
################################################################################
##
## In this section: loop through the lists CxY and Y to construct a list f
## such that each element of f represents the element of CxY of the same index,
## i.e., f(n)=CxY(n), n in N, and calculate its size sf
##
f:=[];
y19:=[];
for k in [1..sCxY]do
      x11:=CxY[k][2];
      diff2:=Difference(CxY[k][1],[x11]);
      for j in [1..sY]do
            if diff2=Y[j] then
               y19:=[j];
            fi;
      od;
```

```
      NCxY:=Concatenation(["c",String(x11),",","Y",String(y19[1])]);
      Add(f,NCxY);
od;
sf:=Size(f);
##
################################################################################
##
gens1:=[];
for j in [1..sf]do # Loop through the list f to create a list gens1 of type(1)
                   # generators of of the subgroup Conj(G_zeta), and calculate
                   # its size sgens1. Each element of gens1 represents the
                   # element of f of the same index, i.e., gens1(n)=f(n), n in N.
                   # (This make these generators compatible with GAP format.)
    Add(gens1,Concatenation(["f",String(j)]));
od;
sgens1:=Size(gens1);
return[CxY,sCxY,Y,sY,f,sf,gens1,sgens1];
end;;
```

4. APCGRelationRConj1 Function

```
APCGRelationRConj1:=function(CxY, Y, f)
local k,j,i,diff2,R1,XX1,XX2,idx1,idx2,t,y12,rels,R2a,sR2a,x8,sY,sCxY,sf;
##
################################################################################
##
## The input of this function are:
### CxY: list of elementary partial conjugations of Conj(G_zeta) or Conjv
###      computed  in "GeneratorsOfSubgroupConj" or "GeneratorsOfSubgroupConjv",
### Y:   list of the non-empty union of connected components of zeta\St(v)
###      computed  in "GeneratorsOfSubgroupConj" or "GeneratorsOfSubgroupConjv",
### f:  the list of the names of the definitions of the generators CxY
###     [f(n) = CxY(n), n in N].
##
## It computes the list of indices [0,idx1,idx2] of relations of type (C1) of
## Conj(G_zeta) or (Re1) of Conjv and adds each of them to the list R2a.
## In addition it calculates the size of the list ‘R2a’.
## It returns [R2a,sR2a].
################################################################################
##
sY:=Size(Y);
sCxY:=Size(CxY);
sf:=Size(f);
R2a:=[];
if sY<>0 then
       y12:=[];
       for k in [1..sCxY]do # loop through the list CxY
             ########################################################################
             ##
             ## In this section we compute the list of indices of the generators which
             ## is [0,idx1,idx2] of the relators of type (C1) or (Re1) by satisfying the
```

```
            ## conditions of the relation (C1) or relation(Re1).
            ## 0: is just flag to let us know that all the generators here of power 1.
            ## idx1: represents the index of a specific generator f(t) of f.
            ## idx2: represents the index of the inverse of the specific generator f(t).
            ## For example if [0,idx1,idx2]= [ 0, 1, 4] then this means f1*f4=1.
            ##
            x8:=CxY[k][2];
            diff2:=Difference(CxY[k][1],[x8]);
            for j in [1..sY]do
                  if diff2=Y[j] then
                      y12:=[j];
                  fi;
            od;
            XX1:=Concatenation(["c",String(x8),",","Y",String(y12[1])]);
            # XX1: represents a specific partial conjugations automorphism
            # alpha_Y,v of the list CxY
            XX2:=Concatenation(["c",String(-x8),",","Y",String(y12[1])]);
            # XX2: represents a specific partial conjugations automorphism
            # alpha_Y,v^-1 of the list CxY which is the inverse of alpha_Y,v
            idx1:=0;
            idx2:=0;
            for t in [1..sf] do  # loop through the list f to find the indices
                if XX1=f[t] then
                   idx1:=t;
                fi;
                if XX2=f[t] then
                   idx2:=t;
                fi;
            od;
            Add(R2a,[0,idx1,idx2]);
            ##
            #########################################################################
            ##
      od;
else
    return("sY must be greater than zero");
fi;
sR2a:=Size(R2a);
return([R2a,sR2a]);
end;
```

5. APCGRelationRConj2 Function

```
APCGRelationRConj2:=function(CxY,Y,Lk,f,R2a)
local k,m,n,j,i,q,l,diff2,diff3,diff4,R2,XX1,XX2,XX3,idx1,idx2,idx3,t,y11,
y12,y13,y16,rels,sR2a,x8,x08,x11,IntY,UniY,U3,NUniA,sLK,lk,sY,sCxY,sf;
##
############################################################################
##
## The input of this function are:
### CxY: the list of elementary partial conjugations of Conj(G_zeta) or Conjv
```

```
###      computed  in "GeneratorsOfSubgroupConj" or "GeneratorsOfSubgroupConjv",
### Y:   the list of the non-empty union of connected components of zeta\St(v)
###      computed  in "GeneratorsOfSubgroupConj" or "GeneratorsOfSubgroupConjv",
### Lk:  the list of links computed in "StarLinkDominateOfVertex"
### f:   the list of the names of the definitions of the generators CxY
###      [f(n) = CxY(n), n in N],
### R2a: the list of indices computed in "APCGRelationRConj1".
##
## It computes the list of indices [0,idx1,idx2,idx3] of relations of type (C2)
## of Conj(G_zeta) or (Re2) of Conjv and adds each of them  to the list R2a (we
## can replace R2a by [] if we need just the indices [0,idx1,idx2,idx3] of
## relations of type (C2) or (Re2)).
## In addition it calculates the size of the list R2a.
## It returns [R2a,sR2a].
##############################################################################
##
sY:=Size(Y);
sCxY:=Size(CxY);
sf:=Size(f);
if sY<>0 then
   y11:=[];
   y13:=[];
   for i in [1..sCxY-1]do # loop through the list CxY excluding the last entry in CxY
       x8:=CxY[i][2];
       x08:=AbsoluteValue(x8);
       diff2:=Difference(CxY[i][1],[x8]);
                 # diff2: represents the connected component Y(i) which is related
                 # to a specific partial conjugation "alpha_Y(i),v" (CxY in this code)
       for t in [1..sY]do
         # Verify the index of a given list (diff2) in Y which related to "alpha_Y(i),v"
           if diff2=Y[t] then
              y11:=[t];
           fi;
       od;
       for j in [i+1..sCxY]do # loop through the list CxY excluding the first entry in CxY
           if x8=CxY[j][2] then
              diff3:=Difference(CxY[j][1],[x8]);
                       # diff3: represents the connected component Y(i) which is related
                       # to a specific partial conjugation "alpha_Y(j),v" (CxY in this code)
              for m in [1..sY]do
              # Verify the index of a given list diff2 in Y which related to "alpha_Y(j),v"
                  if diff3=Y[m] then
                      y13:=[m];
                  fi;
              od;
      IntY:=Intersection( [ diff2 , diff3 ] );
              if IntY=[]  then
                  UniY:=Union( [ diff2 , diff3 ] );
                  U3:=SSortedList(UniY);
                      # U3: the sorted list of the union of the two components
                      # diff2 and diff3 (Y union Z in the relation C2)
                  NUniA:=[];
                  lk:=Lk[x08];
```

```
    sLK:=Size(lk);
    if sLK<>0 then
       for q in [1..sLK]do
           # loop through the list lk to do that: if the vertex l and its
           # inverse -l are belong to lk and U3 in the same time then we
           # delete them, because they will cancel each other.
           l:=lk[q];
           if l in U3 and -l in U3 then
              NUniA:=Difference(U3,[-l,l]);
              U3:=NUniA;
           fi;
       od;
    fi;
    for n in [1..sCxY]do
        # Verify the index of a given list diff4 in Y which is related
        # to the automorphism "alpha_Y(i)+Y(j),v^-1" as in the relation (C2)
        x11:=CxY[i][2];
        diff4:=Difference(CxY[n][1],[x11]);
        if U3=diff4 and  CxY[n][2]=x8 then
            y16:=[];
            for t in [1..sY]do
                if diff4=Y[t] then
                    y16:=[t];
                fi;
            od;
            XX1:=Concatenation(["c",String(x8),",","Y",String(y11[1])]);
            ## XX1: represents a specific partial conjugations automorphism
            ## "alpha_Y(i),v" of the list CxY
            XX2:=Concatenation(["c",String(x8),",","Y",String(y13[1])]);
            ## XX2: represents a specific partial conjugations automorphism
            ## "alpha_Y(j),v" of the list CxY
            XX3:=Concatenation(["c",String(-x8 ),",","Y",String( y16[1])]);
            ## XX3: represents a specific partial conjugations  automorphism
            ## "alpha_Y(i)+Y(j),v^-1" of the list CxY   which is the inverse
            ## of "alpha_Y+Z,v"
            idx1:=0;
            idx2:=0;
            idx3:=0;
            for t in [1..sf] do
                if XX1=f[t] then
                    idx1:=t;
                fi;
                if XX2=f[t] then
                    idx2:=t;
                fi;
                if XX3=f[t] then
                    idx3:=t;
                fi;
            od;
            Add(R2a,[0,idx1,idx2,idx3]);
        fi;
    od;
fi;
```

```
            fi;
        od;
    od;
else
    return("sY must be greater than zero");
fi;
sR2a:=Size(R2a);
return([R2a,sR2a]);
end;
```

6. APCGRelationRConj3 Function

```
APCGRelationRConj3:=function(CxY,Y,Lk,f,R2a)
local k,m,n,j,i,q,l,diff2,diff3,diff4,R3,XX1,XX2,XX3,XX4,idx1,idx2,
idx3,idx4,t,y9,y10,y11,y12,y13,y16,rels,sR2a,x8,x08,x9,x11,IntY,UniY,
U3,NUniA,sLK,lk,invLk2,UniLk,sY,sCxY,sf;
##
##############################################################################
##
## The input of this function are:
### CxY: the list of elementary partial conjugations of Conj(G_zeta) or Conjv
###      computed  in "GeneratorsOfSubgroupConj" or "GeneratorsOfSubgroupConjv",
### Y:   the list of the non-empty union of connected components of zeta\St(v)
###      computed  in "GeneratorsOfSubgroupConj" or "GeneratorsOfSubgroupConjv",
### Lk:  the list of links computed in "StarLinkDominateOfVertex"
### f:   the list of the names of the definitions of the generators CxY
###      [f(n) = CxY(n), n in N],
### R2a: the list of indices computed in "APCGRelationRConj1".
##
## It computes the list of indices [0,idx1,idx2,idx3,idx4] of relations of type (C3)
## of Conj(G_zeta) or (Re3) of Conjv and adds each of them  to the list R2a (we
## can replace R2a by [] if we need just the indices [0,idx1,idx2,idx3,idx4] of
## relations of type (C3) or (Re3)).
## In addition it calculates the size of the list R2a.
## It returns [R2a,sR2a].
###############################################################################
##
sY:=Size(Y);
sCxY:=Size(CxY);
sf:=Size(f);
if sY<>0 then
    y9:=[];
    for i in [1..sCxY-1]do # loop through the list CxY excluding the last entry in CxY
        x8:=CxY[i][2];
        diff2:=Difference(CxY[i][1],[x8]);
                 # diff2: represents the connected component Y(i) which is related to
                 # a specific partial conjugation "alpha_Y(i),v" (CxY in this code) of (C3)
        for t in [1..sY]do
            # Verify the index of a given list diff2 (Y(i)) in Y which related
            # to "alpha_Y(i),v"
            if diff2=Y[t] then
```

```
        y9:=[t];
    fi;
od;
x08:=AbsoluteValue(x8);
invLk2:=-Lk[x08];
UniLk:=Concatenation(Lk[x08],invLk2);
for j in [i+1..sCxY]do   # loop through the list CxY excluding the first entry in CxY
    x9:=CxY[j][2];
    diff3:=Difference(CxY[j][1],[x9]);
    # diff3: represents the connected component Y(j) which is related to
    # a specific partial conjugation "alpha_Y(j),v" (CxY in this code) of (C3)
    y10:=[];
    for m in [1..sY]do
        # Verify the index of a given list diff2 (Y(j)) in Y which related
        # to "alpha_Y(j),v"
        if diff3=Y[m] then
            y10:=[m];
        fi;
    od;
    ##################################################################################
    ##
    ## In this section we compute the list of indices of the generators which is
    ## [0,idx1,idx2,idx3,idx4] of the relators of type (C3) or (Re3) by satisfying
    ## the conditions of the relation (C3) or relation(Re3).
    ## 0: is just flag to let us know that all the generators here of power 1.
    ## idx1: represents the index of a specific generator f(i) of f.
    ## idx2: represents the index of another specific generator f(j) of f.
    ## idx3: represents the index of the inverse of the specific generator f(i).
    ## idx4: represents the index of the inverse of the specific generator f(j).
    ## For example if [0,idx1,idx2,idx3,idx4]= [ 0, 1, 2, 4, 3 ] then this means
    ## f1*f2*f4*f3=1.
    ##
    if not (x8 in diff3) and not (x9 in diff2) then
        if x8<>x9 and x8<>-x9 then
           IntY:=Intersection( [ diff2 , diff3 ] );
         if IntY=[] or x9 in UniLk then
               XX1:=Concatenation(["c",String(x8),",","Y",String(y9[1])]);
               # XX1: represents a specific partial conjugations
               # automorphism "alpha_Y(i),v" of the list CxY
               XX2:=Concatenation(["c",String(x9),",","Y",String(y10[1])]);
               # XX2: represents a specific partial conjugations
               # automorphism "alpha_Y(j),u" of the list CxY
               XX3:=Concatenation(["c",String(-x8),",","Y",String(y10[1])]);
               # XX3: represents a specific partial conjugations
               # automorphism "alpha_Y(i),v^-1" of the list CxY
               # which is the inverse of "alpha_Y,v"
               XX4:=Concatenation(["c",String(-x9),",","Y",String(y9[1])]);
               # XX4: represents a specific partial conjugations
               # automorphism "alpha_Y(j),u^-1" of the list CxY
               # which is the inverse of "alpha_Y(j),u"
               idx1:=0;
               idx2:=0;
               idx3:=0;
```

```
                        idx4:=0;
                        for t in [1..sf] do
                            if XX1=f[t] then
                                idx1:=t;
                            fi;
                            if XX2=f[t] then
                                idx2:=t;
                            fi;
                            if XX3=f[t] then
                                idx3:=t;
                            fi;
                            if XX4=f[t] then
                                idx4:=t;
                            fi;
                        od;
                        Add(R2a,[0,idx1,idx2,idx3,idx4]);
                    fi;
                fi;
            fi;
        od;
     od;
else
     return("sY must be greater than zero");
fi;
sR2a:=Size(R2a);
return([R2a,sR2a]);
end;
```

7. APCGRelationRConj4 Function

```
APCGRelationRConj4:=function(CxY,V,Lk,gens1,Y,f,R2a)
local k,m,n,j,i,q,l,diff2,diff3,diff4,R4,XX1,XX2,XX3,XX4,idx1,idx2,idx3,idx4,
t,y9,y10,y11,y12,y13,y16,sR2a,x8,x08,x9,x11,W,sW,IntY,UniY,U3,NUniA,sLK,lk,
invLk2,UniLk,KK,gens4,sgens4,gens3,sgens3,st1,st2,jx,Wj4,Wj,Wj3,Wj2,Wj1,Wznot,
sWznot,j1,y99,NCY,CxY1,sCxY1,x09,W1,y14,diff5,sY,sCxY,sf,sgens1,invV,L;
##
#############################################################################
##
## The input of this function are:
### CxY: the list of elementary partial conjugations of Conj(G_zeta) or Conjv
###      computed  in "GeneratorsOfSubgroupConj" or "GeneratorsOfSubgroupConjv",
### V:   the list of vertices
### Lk:  the list of links computed in "StarLinkDominateOfVertex"
### gens1: type(1) generators of Conj(G_zeta) or Conjv computed in
###        "GeneratorsOfSubgroupConj" or "GeneratorsOfSubgroupConjv",
### Y:   the list of the non-empty union of connected components of zeta\St(v)
###      computed  in "GeneratorsOfSubgroupConj" or "GeneratorsOfSubgroupConjv",
### f:   the list of the names of the definitions of the generators CxY
###      [f(n) = CxY(n), n in N],
### R2a: the list of indices computed in "APCGRelationRConj1".
##
```

```
## Firstly, it computes the list of elementary inner automorphisms W, then
## gens4 the list of the generators of Conj(G_zeta) or Conjv. This is the
## concatenation of the lists gens1 and W but; without repeating generators
## that appear in gens1.
## Secondly, it computes the list of indices [1,idx1,idx2,idx3,idx4] of relations
## of type (C4) or (Re4) and adds each of them  to the list R2a (we
## can replace R2a by [] if we need just the indices [1,idx1,idx2,idx3,idx4]
## of these relations.
## It returns [W,gens4,R2a,sW,sgens4,sR2a] where sW, sgens4 and sR2a are the
## sizes of W, gens4 and R2a respectively.
##############################################################################
##
sCxY:=Size(CxY);
sgens1:=Size(gens1);
sY:=Size(Y);
sf:=Size(f);
invV:=-V;                  # invV: is the inverses list of the vertex list V
L:=Concatenation(V,invV);  # L is the union of the lists V and invV
if sY<>0 then
       ##########################################################################
       ##
       ## In this section we compute the list of elementary inner automorphisms W
       ## of the subgroup Conj(G_zeta) or Conjv by satisfying the conditions of this
       ## type of partial conjugations automorphisms
       ##
       W:=[];
       for j in [1..sCxY]do    # loop through the list CxY defined above
             x9:=CxY[j][2];
             x09:=AbsoluteValue(x9);
             invLk2:=-Lk[x09]; # Compute invLk2 the inverse of of each link Lk(v); v in V
             UniLk:=Concatenation(Lk[x09],invLk2);
                                        # Compute UniLk the link Lk(v) with respect to L
             diff4:=Difference(L,UniLk); # For each vertex v of V we remove the list UniLk
                                        # from L, since UniLk consist of vertices with
                                        # thier inverses which cancel each other
             diff5:=Difference(diff4,-[x9]);
             # diff5 is a one list (connected component) Y(i) of the list
             # Y which forms the first part of the inner automorphism W1
             W1:=[diff5,x9];   # Forms the elementary inner automorphism W1
             Add(W,W1);
       od;
       ##
       ##############################################################################
       ##
       sW:=Size(W);
       Wznot:=[];
       gens3:=[];
       j1:=0;
       for j in [1..sCxY]do
           # In this loop we add each elementary inner automorphisms W(j) to
           # a new list Wznot if W(j) not belong to the list CxY and it is not
           # trivial automorphism then add its name W(j1) to the list gens3
           if not (W[j] in CxY) and Size(W[j][1])<>1 then
```

```
        Add(Wznot,W[j]);
        j1:=j1+1;
        Add(gens3,Concatenation(["W",String(j1)]));
    fi;
od;
sWznot:=Size(Wznot);
sgens3:=Size(gens3);
if Wznot<>[] then
    gens4:=Concatenation(gens1,gens3);
          # gens4: the list of the generators of Conj(G_zeta) or Conjv
else
    gens4:=gens1; # Means the subgroup Conj(G_zeta) or Conjv has just the
                  # type (1) generators (elementary partial conjugations)
fi;
sgens4:=Size(gens4);
y14:=[];
for i in [1..sCxY]do # loop through the list CxY excluding the first entry in CxY
     x8:=CxY[i][2];
     diff2:=Difference(CxY[i][1],[x8]);
           # diff2: represents the connected component Y(i) which is related
           # to a specific partial conjugation "alpha_Y(i),v" (CxY in this code)
     for t in [1..sY]do
           # Verify the index of a given list diff2 ( Y(i) ) in Y which related
           # to "alpha_Y(i),v"
           if diff2=Y[t] then
               y14:=[t];
           fi;
     od;
     ##############################################################################
     ##
     ## In this section we compute the list of indices of the generators which is
     ## [1,idx1,idx2,idx3,idx4] of the relators of type (C4) or (Re4) by
     ## satisfying the conditions of these relations.
     ## 1: is just flag to let us know that R corresponds to a word
     ## W_R = gamma_u * alpha_Y,v * gamma^-1_u * alpha_Y,v^-1 of length 4 as in
     ## relation (C4) and (Re4) of the subgroups Conj(G_zeta) and Conjv respectively.
     ## idx1: represents the index of a specific generator f(i) of f.
     ## idx2: represents the index of another specific generator f(t) of f.
     ## idx3: represents the index of the inverse of the specific generator f(i).
     ## idx4: represents the index of the inverse of the specific generator f(t).
     ## For example if [0,idx1,idx2,idx3,idx4]= [ 0, 1, 2, 4, 3 ] then this means
     ## f1*f2*f4*f3=1.
     ##
     for j in [1..sCxY]do
            x9:=W[j][2];
            diff3:=Difference(CxY[j][1],[x9]);
            if not (x9 in diff2) and x8<>x9 and x8<>-x9 and Size(W[j][1])<>1 then
                 diff4:=Difference(W[j][1],[x9]);
                 Add(diff4,-x9);
                 diff4:=SSortedList(diff4);
                 diff5:=[diff4,-x9];
                 idx3:=0;
                 for k in [1..sW]do
```

```
                    if diff5=W[k] then
                        idx3:=k+sgens1;
                    fi;
                od;
                idx1:=j+sgens1;
                Wj:=W[j];
                Wj1:=Difference(W[j][1],[W[j][2]]);
                Wj2:=Union([Wj1,[-W[j][2]]]);
                Wj3:=SSortedList(Wj2);
                Wj4:=[Wj3,-W[j][2]];
                for q in [1..sCxY]do
                    if Wj=CxY[q] then
                        j:=q;
                        idx1:=q;
                        st1:="f";
                     else
                        st1:="W";
                    fi;
                    if Wj4=CxY[q] then
                        jx:=q;
                        st2:="f";
                        idx3:=q;
                        else
                           st2:="W";
                    fi;
                od;
                XX2:=Concatenation(["c",String(x8),",","Y",String(y14[1])]);
                # XX2: represents a specific partial conjugations
                # automorphism "alpha_Y(j),v" of the list CxY
                XX4:=Concatenation(["c",String(-x8),",","Y",String(y14[1])]);
                # XX4: represents a specific partial conjugations
                # automorphism "alpha_Y(j),v^-1" of the list CxY
                # which is the inverse of "alpha_Y(j),v"
                idx2:=0;
                idx4:=0;
                for t in [1..sf] do # loop through the list f defined above
                    if XX2=f[t] then  # Verify the index of the specific partial
                                      # conjugations XX2 in the list Y
                        idx2:=t;
                    fi;
                    if XX4=f[t] then  # Verify the index of the specific partial
                                      # conjugations XX4 in the list Y
                        idx4:=t;
                    fi;
                od;
                Add(R2a,[1,idx1,idx2,idx3,idx4]);
            fi;
        od;
        ##
        ##########################################################################
        ##
    od;
else
```

```
     return("sgens4 must be greater than zero");
fi;

sR2a:=Size(R2a);
return([W,gens4,R2a,sW,sgens4,sR2a]);
end;
```

8. APCGConjLastReturn Function

```
APCGConjLastReturn:=function(gens4,R2a,sR2a)
local i,j,C,F,rels,srels,GHK,KK,GGG,gens,sgens,GHK1,KK1,ZZa;
##
##############################################################################
##
## The input of this function are:
### gens4: the list of generators (defined in APCGRelationRConj4) of the
###        subgroup Conj(G_zeta),
### R2a: the list of the indices of the relators (computed in the function
###      APCGRelationRConj, ..., APCGRelationRConj4), and
### sR2a: the size of the list R2a.
##
## It forms  the list of relations "rels" from the list R2a For each
## element R of R2a the relator W_R is added to  a new list rels
##
## In fact this function forms the output of the functions
## "FinitePresentationOfSubgroupConj" and "FinitePresentationOfSubgroupConjv"
## in the package AutParCommGrp.
###############################################################################
##
C:=gens4;
F:=FreeGroup(C);  # computes the free group on gens4. The generators
                  # are displayed as string.1, string.2, ..., string.n
gens:=GeneratorsOfGroup(F); # returns a list of generators gens of the free group F
sgens:=Size(gens);
##
########################################################################
##
## In this section we form  the list of relations rels from the list R2a
## For each element R of R2a the relator W_R is added to  a new list rels
##
rels:=[];
for i in [1..sR2a] do
      GHK:=Size(R2a[i]);
      KK:=AbsoluteValue(R2a[i][2]);
      rels[i]:=gens[KK];
      for j in [3..GHK] do
          KK:=AbsoluteValue(R2a[i][j]);
          rels[i]:=rels[i]*gens[KK];
      od;
od;
##
```

```
########################################################################
##
GGG:=F/rels;      # computes the finitely presented group on
                  # the generators gens of F defined above
srels:=Size(rels);
return[gens,rels,GGG];
end;
```

9. FinitePresentationOfSubgroupConj Function

```
FinitePresentationOfSubgroupConj:=function(V,E)
local R1,R2,R3,R4,R5,R6,R7,R8,St,Lk,sV,M,NV,NE,sNV,sNE,Bs,CxY,sCxY,gens1,
sgens1,gens,sgens,R2a,sR2a,Y,sY,f,sf,F,T,gens4,sgens4,GGG,L,sL,W,sW,rels,
srels,Q,i,j,tempedgex,tempedgey;
##
#############################################################################
##
## The input of this function is  a simple graph zeta=(V,E), where V and E
## represent the set of vertices and the set of edges respectively.
##
## It returns [gens,rels,GGG], where
### gens: is a list of free generators of the subgroup Conj(G_zeta) of the
###       group Aut(G_zeta).
### rels: is a list of relations in the generators of the free group F.
###       Note that relations are entered as relators, i.e., as words in
###       the generators of the free group
### GGG:=F/rels: is a finitely presented of the subgroup Conj(G_zeta)
###              with generators gens and relators rels.
##
## In fact, the main work of this function is to run all the functions
## we have read them below to give a finite presentation for the subgroup
## Conj(G_zeta) of Aut(G_zeta).
##############################################################################
##
if IsSimpleGraph(V,E)=true then      # Call the function IsSimpleGraph to test
                                     # whether the graph zeta is simple or not
  ##
  ############################################################################
  ##
  ## This section is to compute the star St(v) and the link Lk(v) for each v in V
  ##
  R1:=StarLinkOfVertex(V,E);         #F StarLinkOfVertex( <V>, <E> )
                                     ## return([St,Lk]);
  St:=R1[1];
  Lk:=R1[2];
  ##
  ############################################################################
  ##
  ## This section is to delete the star St(v) of a specific vertex v
  ## from the graph zeta
  ##
```

```
R2:=DeleteVerticesFromGraph(St,V,E);  #F DeleteverticesFromGraph( <St>, <V>, <E> )
                                      ## return([NV,NE,sNV,sNE]);
NV:=R2[1];
NE:=R2[2];
sNV:=R2[3];
sNE:=R2[4];
##
##############################################################################
##
## This section is to compute the first part of the generators (elementary
## partial conjugations) of the subgroup Conj(G_zeta)
##
R3:=GeneratorsOfSubgroupConj(NE,NV,V);
                                #F GeneratorsOfSubgroupConj( <NE>, <NV>, <V> )
                                ## return[CxY,sCxY,Y,sY,f,sf,gens1,sgens1];
CxY:=R3[1];
Y:=R3[3];
f:=R3[5];
gens1:=R3[7];
##
##############################################################################
##
## This section is to compute the relation C1 of the subgroup Conj(G_zeta)
##
R4:=APCGRelationRConj1(CxY,Y,f);   #F APCGRelationRConj1( <CxY>, <Y>, <f> )
                                   ## return([R2a,sR2a]);
R2a:=R4[1];
sR2a:=R4[2];
##
##############################################################################
##
## This section is to compute the relation C2 of the subgroup Conj(G_zeta)
##
R5:=APCGRelationRConj2(CxY,Y,Lk,f,R2a);
           #F APCGRelationRConj2( <CxY>, <Y>, <Lk>, <f>, <R2a> )
           ## return([R2a,sR2a]);
R2a:=R5[1];
sR2a:=R5[2];
##
##############################################################################
##
## This section is to compute the relation C3 of the subgroup Conj(G_zeta)
##
R6:=APCGRelationRConj3(CxY,Y,Lk,f,R2a);
                      #F APCGRelationRConj3( <CxY>, <Y>, <Lk>, <f>, <R2a> )
                      ## return([R2a,sR2a]);
R2a:=R6[1];
sR2a:=R6[2];
##
##############################################################################
##
## This section is to compute the relation C4 of the subgroup Conj(G_zeta)
##
```

```
  R7:=APCGRelationRConj4(CxY,V,Lk,gens1,Y,f,R2a);
              #F APCGRelationRConj4( <CxY>, <V>, <Lk>, <gens1>, <Y>, <f>, <R2a> )
              ## return([W,gens4,R2a,sW,sgens4,sR2a]);
  W:=R7[1];
  gens4:=R7[2];
  R2a:=R7[3];
  sW:=R7[4];
  sgens4:=R7[5];
  sR2a:=R7[6];
  ##
  ###############################################################################
  ##
  ## This section is to compute the final relations rels from the matrix R2a
  ## of indices of the generators and find the final return
  ##
  R8:=APCGConjLastReturn(gens4,R2a,sR2a);
                              #F APCGConjLastReturn( <gens4>, <R2a>, <sR2a> )
                              ## return[gens,rels,GGG];
  gens:=R8[1];
  rels:=R8[2];
  GGG:=R8[3];
  ##
  ###############################################################################
  ##
else
    return("The graph must be a simple graph");
fi;
return[gens,rels,GGG];
end;
```

A.3 Appendix to Chapter 4

In this appendix we will attached the codes for all the functions we have written in Chapter 4 as follows:

1. **EquivalenceClassOfVertex Function**

```
EquivalenceClassOfVertex:=function(St)
local i,j,sV,EqCl,EqCl1,diff1,diff2;
##
############################################################################
##
## The input of this function is the list of stars St.
##
## It computes the equivalence classes for each vertex v in V.
#############################################################################
##
EqCl:=[];
sV:=Size(St);          # Since the size of St is the same of the list of vertices V
for i in [1..sV] do    # Loop through the list of vertices V
EqCl1:=[];
    for j in [1..sV] do                  # Loop through the list of vertices V and
                                         # for all vertices i not equal j do that:
        diff1:=Difference(St[i],[i,j]);  # compute diff1(i,j)=St(i)\{i,j}
        diff2:=Difference(St[j],[i,j]);  # compute diff2(i,j)=St(j)\{i,j }
        if diff1 = diff2 then
            Add(EqCl1,j);                # add the vertex j to the list EqCl1 if
                                         # diff1 = diff2
        fi;
    od;
      Add(EqCl,EqCl1);
od;
return(EqCl);
end;
```

2. **ClassPreservingConnectedComponents Function**

```
ClassPreservingConnectedComponents:=function(EqCl, Comps)
local i, j, k ,cdash, remainingcdash, sizeComps, sizeEqClcurrent,sizeEqCl;
##
############################################################################
##
## The input of this function is:
### EqCl: the list of equivalence classes of vertices of the graph zeta, and
### Comps: the list of connected components of the graph zeta.
##
## It constructs a new list of connected components Comps from the connected
## components of the graph zeta by finding the connected components which
## satisfy the conditions of partial conjugation for W_V (see Chapter one of
```

```
## the manual for this package).
################################################################################
##
sizeEqCl:=Size(EqCl);
for i in [1 ..sizeEqCl] do          # loop through the list EqCl
    sizeComps:=Size(Comps);
    sizeEqClcurrent:=Size(EqCl[i]); # computes the size of each element of EqCl
    cdash:=[];
    remainingcdash:=[];
    for j in [1..sizeEqClcurrent] do      # loop through each element of EqCl
for k in [1..sizeComps] do              # loop through the list Comps
    if EqCl[i][j] in Comps[k] then      # if any element of EqCl(i)(j) belong to
                                          # any connected component Comps(k) then do:
        cdash:=Union(cdash, Comps[k]);  # Union between the lists cdash and Comps(k)
    fi;
od;
    od;
    for k in [1..sizeComps] do # For each element Comps(k) of Comps, the function IsSubset
                               # is called to find remainingcdash the remaining components
                               # from the list Comps that contain no element of EqCl(i)

if IsSubset(cdash,Comps[k])=false then
    Add(remainingcdash,Comps[k]);
        fi;
    od;
    Add(remainingcdash,cdash);
    Comps:=remainingcdash;   # Make a new list of connected components by
                             # making Comps equal to list remainingcdash
od;
return(Comps);
end;
```

3. GeneratorsOfSubgroupConjv Function

```
GeneratorsOfSubgroupConjv:=function(NE,NV,St,V)
local i,j,gens2,gens,genss,rels,Rels,Bs,h,G2,G1,R3,R4,Comps,sComps,sMV,sNE,
UniA,D,DD,sD,S,YYY,NYY,invNYY,DYY,sDYY,Ls,t,xn,union_element,NCxY,sgens,
gens4,sgens4,gens3,sgens3,invV,sL,Y6,xs2,Y3,Y4,sY4,xs1,diff2,Y5,sY5,sY6,
sz,Y7,sY7,sxs2,xs3,sxs3,xs,sxs,Uxs,sUxs,CxY,sCxY,y9,y8,Y,sY,sBs,Y8,sY8,
y19,x11,sxs1,k,f,sf,gens1,sgens1,CxY1,sCxY1,y10,y99,NCY,KK,HH,R10,R11,
R12,SuccComps,EqCl,sR12,PY4,sPY4,L,sV;
##
###############################################################################
##
## The input of this function are:
### the list NE of all lists of edges of the subgraph zeta\St(v),
### the list NV of all lists of vertices of the subgraph zeta\St(v),
### the list of stars St,
### the list of vertices V.
##
## It computes the list gens1 which form the type(1) generators of partial
```

```
## conjugation for W_V  the subgroup of Conj_V of the group Aut(G_zeta).
###############################################################################
##
gens:=[];
Bs:=[];
Y6:=[];
xs2:=[];
sNE:=Size(NE);
sV:=Size(V);
invV:=-V;                  # invV: is the inverses list of the vertex list V
L:=Concatenation(V,invV);  # L is the union of the lists V and invV
R11:=EquivalenceClassOfVertex(St); # Call this function to computes the equivalence
                                   # Classes of each vertex v of the graph zeta
EqCl:=R11;
for h in [1..sNE]do #loop through the lists NV and NE since they have the same size
      G2:=NE[h];
      G1:=NV[h];
      R3:=ConnectedComponentsOfGraph(G1,G2); # computes the list of all Connected components
                                             # for each subgraph (NV(h),NE(h))
      Comps:=R3[1];                          # Comps: list of components of (NV(h),NE(h))
      sComps:=R3[2];                         # sComps: size of Comps

      R12:=ClassPreservingConnectedComponents(EqCl,Comps);
                    # Call this function to construct a new list of connected components
                    # Comps from the connected components of the subgraph (NV(h),NE(h))
                    # by finding the connected components which satisfy the conditions
                    # of partial conjugation for W_V
      sR12:=Size(R12);
      ############################################################
      Y4:=[];
      for i in [1..sR12] do    # loop through the lists R12
            if R12[i]<>[] then # Chech that if R12(i) is not empty list
               Add(Y4,R12[i]); # If R12(i) is not empty add it to the list Y4
            fi;
      od;
      sY4:=Size(Y4);
      ###############################################################
       xs1:=[];
       for i in [1..sY4] do            # loop through the list Y4
           diff2:=Difference(L,Y4[i]); # computes the difference diff2 between the
                                       # list L and each elements (list) of the list
           Add(xs1,diff2);             # Y4 add each diff2 to the new list xs1
       od;
       sxs1:=Size(xs1);
      ###########################################################################
      ##
      ## In this section: loop through the list Y4 to construct a list called Y6.
      ## In order to do this first find the size sz of xs1(i). For each element l
      ## of xs1(i) concatenate elements of Y4(i)  with elements of l to give a list
      ## KK. Then form a listY5 of pairs HH; with entries (KK, l), for each element
      ## l of xs1(i). Then append Y5 to the list Y6.
      ##
      Y5:=[];
```

```
      for i in [1..sY4] do
           sz:=Size(xs1[i]);
           for j in [1..sz] do
               KK:=Concatenation(Y4[i],[xs1[i][j]]);
               HH:=[KK,xs1[i][j]];
               Add(Y5,HH);
           od;
      od;
      sY5:=Size(Y5);
      Add(Y6,Y5);
      sY6:=Size(Y5);
      ##
      ############################################################################
      ##
      Add(xs2,xs1); # Make new list xs2, by adding xs1 to xs2. This step and tht
                    # next one are needed because there are two inner loops
      Add(Bs,Y4);   # Make new lists Bs, by adding Y4 to Bs
od;                 # ending the loop through the lists NV and NE

if Y6<>[] then  # To check that the list Y6 is nonempty list i.e., Y6 have
                # connected components that satisfy the conditions of Conjv
  sY6:=Size(Y6);
  Y7:=Concatenation(Y6);
      # Compute the list Y7 by concatenating the dense list of lists Y6
  sY7:=Size(Y7);
  sxs2:=Size(xs2);
  xs3:=Concatenation(xs2);
       # Compute the list xs3 by concatenating the dense list of lists xs2
  sxs3:=Size(xs3);
  ##
  ###############################################################################
  ##
  ## In this section: loop through the list xs3 to construct a list called xs by
  ## adding each non-empty entry of xs3 to xs, and calculate the size of xs.
  ##
  xs:=[];
  for i in [1..sxs3] do
      if not (xs3[i] in xs) and xs3[i]<>[] then
          Add(xs,xs3[i]);
      fi;
  od;
  sxs:=Size(xs);
  ##
  ###############################################################################
  ##
  Uxs:=Union(xs);    # Call the function Union to construct a list called Uxs by
  sUxs:=Size(Uxs);   # computing the union of xs and calculates it size sUxs
  CxY1:=[];
  for i in [1..sY7] do   # Loop through the list Y7 to construct a list
                         # called CxY1 by adding each non-empty entry of
                         # Y7 to CxY1, and calculate its size sCxY1
     if not (Y7[i] in CxY1) and Y7[i]<>[] then
        Add(CxY1,Y7[i]);
```

```
   fi;
od;
sCxY1:=Size(CxY1);
CxY:=[];
for j in [1..sCxY1]do   # Loop through the list CxY1 to compute a list of
                        # the definitions CxY of the elementary partial
                        # conjugations, with its size sCxY
    y9:=CxY1[j][2];
    y10:=CxY1[j][1];
    y99:=SSortedList(y10);
    NCY:=[y99,y9];
    Add(CxY,NCY);
od;
sCxY:=Size(CxY);
Y8:=Concatenation(Bs); # Make a list Y8 by concatenating the dense
                       # list of lists Bs defined above
sBs:=Size(Bs);
sY8:=Size(Y8);
Y:=[];
for i in [1..sY8] do   # Loop through the list Y8 to construct a list Y
                       # of the non-empty unions of connected components
                       # of zeta\St(v)
   if not (Y8[i] in Y) and Y8[i]<>[] then
      Add(Y,Y8[i]);
   fi;
od;
sY:=Size(Y);
#################################################################################
##
## In this section: loop through the lists CxY and Y to construct a list f such
## that each element of f represents the element of CxY of the same index, i.e.,
## f(n)=CxY(n), n in N, and calculate its size sf
##
f:=[];
y19:=[];
for k in [1..sCxY]do
    x11:=CxY[k][2];
    diff2:=Difference(CxY[k][1],[x11]);
    for j in [1..sY]do
          if diff2=Y[j] then
             y19:=[j];
          fi;
    od;
    NCxY:=Concatenation(["c",String(x11),",","Y",String(y19[1])]);
    Add(f,NCxY);
od;
sf:=Size(f);
##
################################################################################
##
gens1:=[];
for j in [1..sf]do # Loop through the list f to create a list gens1 of type(1)
                   # generators of of the subgroup Conj(G_zeta), and calculate
```

```
                     # its size sgens1. Each element of gens1 represents the
                     # element of f of the same index, i.e., gens1(n)=f(n), n in N.
                     # (This make these generators compatible with GAP format.)

     Add(gens1,Concatenation(["f",String(j)]));
  od;
  sgens1:=Size(gens1);
  return[CxY,sCxY,Y,sY,f,sf,gens1,sgens1];
else
   Print("There is no component C satisfies the conditions of partial conjugations");
   Print("\n");
   return[];
fi;
end;
```

4. FinitePresentationOfSubgroupConjv Function

```
FinitePresentationOfSubgroupConjv:=function(V,E)
local R1,R2,R3,R4,R5,R6,R7,R8,St,Lk,Lk1,sV,M,NV,NE,sNV,sNE,Bs,CxY,sCxY,
gens1,sgens1,gens,sgens,R2a,sR2a,Y,sY,f,sf,F,T,gens4,sgens4,GGG,L,sL,W,
sW,rels,srels,Q,i,j,tempedgex,tempedgey;
##
###############################################################################
##
## The input of this function is  a simple graph zeta=(V,E), where V and E
## represent the set of vertices and the set of edges respectively.
##
## It returns [gens,rels,GGG], where
### gens: is a list of free generators of the subgroup Conj_V of the
###       group Aut(G_zeta).
### rels: is a list of relations in the generators of the free group F.
###       Note that relations are entered as relators, i.e., as words in
###       the generators of the free group.
### GGG:=F/rels: is a finitely presented of the subgroup Conj_V with
###              generators gens and relators rels.
##
## In fact, the main work of this function is to run all the functions
## we have read them below to give a finite presentation for the subgroup
## Conj_V of Aut(G_zeta).
###############################################################################
##
if IsSimpleGraph(V,E)=true then    # Call the function IsSimpleGraph to test
                                   # whether the graph zeta is simple or not
   ##
   ###############################################################################
   ##
   ## This section is to compute the star St(v) and the link Lk(v) for each v in V
   ##
   R1:=StarLinkOfVertex(V,E);         #F StarLinkOfVertex( <V>, <E> )
                                      ## return([St,Lk]);
   St:=R1[1];
```

```
Lk:=R1[2];
##
##############################################################################
##
## This section is to delete the star St(v) of a specific vertex v
## from the graph zeta
##
R2:=DeleteVerticesFromGraph(St,V,E); #F DeleteverticesFromGraph( <St>, <V>, <E>)
                                     ## return([NV,NE,sNV,sNE]);
NV:=R2[1];
NE:=R2[2];
sNV:=R2[3];
sNE:=R2[4];
##
##############################################################################
##
## This section is to compute the first part of the generators W_v
## of the subgroup Conj_V
##
R3:=GeneratorsOfSubgroupConjv(NE,NV,St,V);
                         #F GeneratorsOfSubgroupConjv( <NE>, <NV>, <St>, <V> )
                         ## return[CxY,sCxY,Y,sY,f,sf,gens1,sgens1];
if R3[1]<>[] then
   CxY:=R3[1];
   Y:=R3[3];
   f:=R3[5];
   gens1:=R3[7];
   ##
   ##############################################################################
   ##
   ## This section is to compute the relation Re1 of the subgroup Conj(G_zeta)
   ##
   R4:=APCGRelationRConj1(CxY,Y,f);    #F APCGRelationRConj1( <CxY>, <Y>, <f> )
                                       ## return([R2a,sR2a]);
   R2a:=R4[1];
   sR2a:=R4[2];
   ##
   ##############################################################################
   ##
   ## This section is to compute the relation R2 of the subgroup Conj(G_zeta)
   ##
   R5:=APCGRelationRConj2(CxY,Y,Lk,f,R2a);
                         #F APCGRelationRConj2( <CxY>, <Y>, <Lk>, <f>, <R2a> )
                         ## return([R2a,sR2a]);
   R2a:=R5[1];
   sR2a:=R5[2];
   ##
   ##############################################################################
   ##
   ## This section is to compute the relation R3 of the subgroup Conj(G_zeta)
   ##
   R6:=APCGRelationRConj3(CxY,Y,Lk,f,R2a);
                         #F APCGRelationRConj3( <CxY>, <Y>, <Lk>, <f>, <R2a> )
```

```
                              ## return([R2a,sR2a]);
      R2a:=R6[1];
      sR2a:=R6[2];
      ##
      ##############################################################################
      ##
      ## This section is to compute the relation R4 of the subgroup Conj(G_zeta)
      ##
      R7:=APCGRelationRConj4(CxY,V,Lk,gens1,Y,f,R2a);
                 #F APCGRelationRConj4( <CxY>, <L>, <Lk>, <gens1> , <Y>, <f>, <R2a> )
                 ## return([W,gens4,R2a,sW,sgens4,sR2a]);
      W:=R7[1];
      gens4:=R7[2];
      R2a:=R7[3];
      sW:=R7[4];
      sgens4:=R7[5];
      sR2a:=R7[6];
      ##
      ##############################################################################
      ##
      ## This section is to compute the final relations rels from the matrix R2a
      ## of indices of the generators and find the final return
      ##
      R8:=APCGConjLastReturn(gens4,R2a,sR2a);
                               #F APCGConjLastReturn( <gens4>, <R2a>, <sR2a> )
                               ## return[gens,rels,GGG];
      gens:=R8[1];
      rels:=R8[2];
      GGG:=R8[3];
      ##
      ##############################################################################
      ##
      return[gens,rels,GGG];
   else
      Print("The subgroup here is trivial subgroup");
      Print("\n");Print("\n");
      return[];
   fi;
else
    return("The graph must be a simple graph");
fi;
end;
```

A.4 Appendix to Chapter 8

In this appendix we will attached the codes for all the functions we have written in Chapter 8 as follows:

1. **SwapRowsColumns Function**

```
SwapRowsColumns:=function(degf,x,y)
local Temp5,Temp6;
##
#############################################################################
##
## The input of this function are:
### a matrix degf of size m x m and two different numbers x,y where
### x,y in {1, ..., m}.
##
## It exchanges row(x) and row(y), and at the same time exchange,
## column(x) and column(y).
## It returns the matrix degf after the replacement.
##############################################################################
##
##In this section we exchange the two rows  x and y
##
      Temp5:=[];
      Temp5 := StructuralCopy(degf); # Row replacement
      degf[x]:=Temp5[y];
      degf[y]:=Temp5[x];
##
##############################################################################
##
      degf:=TransposedMatDestructive(degf); # compute the transpose of degf
##
##############################################################################
##
##In this section we exchange the two columns x and y
##
      Temp6:=[];
      Temp6 := StructuralCopy(degf);
      degf[x]:=Temp6[y];
      degf[y]:=Temp6[x];
##
##############################################################################
##
      degf:=TransposedMatDestructive(degf); # compute the transpose of degf
##
##############################################################################
##
return (degf);
end;
```

2. Solveindic1WithProof Function

```
Solveindic1WithProof:=function(dimf,f)
local i,j,diffk,dimej,dimei,f1,Cj,M1,M2,Cjb,Ca,Cja,Ma,Mb,Mc,Xd,Xd1,Md,Me1,Me2,m;
##
##############################################################################
##
## This function is called only if the conditions of Propositions 1.4.1
## (as in the manual) holds.
##
## The input of this function are:
### dimf: the matrix of the dimensions of the polynomials which is of size m x m,
###    f: the identity matrix of size m x m.
### dimf and f are output by the main function IsSolvableModuleWithProof.
##
## The function outputs a proof that M is solvable.
##
###############################################################################
##
m:=Size(dimf);
##
###############################################################################
##
## In this section we compute new entries for matrix f, by going through the
## entries of the matrix dimf and set f[i][j]= dimf[i][j] if dimf[i][j] < 0
## and f[i][j]=0 if dimf[i][j] >= 0, for i=1, ..., m, depending on the facts
## that in R, if dim (f) = j, i.e., f in R_j then degree of f = - j in the
## negative grading.
##
for j in [1..m] do
    for i in [1..m] do
        if i>j then
             if dimf[i][j]>=0 then
                  f[i][j]:=0;
             else
                   f[i][j]:=dimf[i][j];
             fi;
        else
             f[i][j]:=dimf[i][j];
        fi;
    od;
od;
Print("\ f=",f);
Print(" ","\n");Print(" ","\n");
##
###############################################################################
##
## In this section if f is an upper triangular matrix then we Compute Newf
## from f, using the fact that (partial)^2 =0 and R is an integral domain.
## Also we compute the matrix d of the differential "partial" with respect
## to the basis S = e_i where i=1, ..., m.
##
if IsUpperTriangularMat(f)=true then
```

```
    for i in [1..m] do
        f[i][i]:=0;
    od;
    Print("\ Newf=",f);
    Print(" ","\n");Print(" ","\n");
    for i in [1..m] do
        for j in [1..m] do
            if f[i][j]<>0 then
               f[i][j]:=Concatenation("f",String(i),String(j));
            fi;
        od;
    od;
    Print("\ d=",f);
    Print(" ","\n");Print(" ","\n");
else
    return("f is not upper triangular matrix");
fi;
##
###############################################################################
##
## In this section we construct a proof that M is solvable if f is an
## upper triangular matrix.
##
Print(" , ( Since d^2=0 and R is an integral domain ). ");
Print(" ","\n");Print(" ","\n");
Cjb:=" ";
Ca:="Let C0=0 and ";
Print(Ca);
for j in [1..m] do
    Cja:=Concatenation(["C",String(j),"=<"]);
    for i in [1..j] do
        if i=j then
             M1:=Concatenation(["e",String(i)]);
        else
             M1:=Concatenation(["e",String(i),","]);
        fi;
             Cja:=Concatenation([Cja,M1]);
    od;
    if j=m then
             Cja:=Concatenation([Cja,">  "]);
    else
             Cja:=Concatenation([Cja,"> , "]);
    fi;
    Print(Cja);
    if j=m then
             Cjb:=Concatenation([Cjb,"C",String(j),"/","C",String(j-1)," is free "]);
    else
             Cjb:=Concatenation([Cjb,"C",String(j),"/","C",String(j-1)," is free, "]);
    fi;
od;
Print(" ","\n");
Print(Cjb);
Print(" ","\n");Print(" ","\n");
```

```
M2:=[];
Ma:="x=";
Mb:="d(x)=";
Mc:="d(x)=a1(0)";
Xd:="If x in C";
Me2:="Hence, 0=C0 subset of ";
for j in [1..m] do
    Xd1:=Concatenation([Xd,String(j),", then x can be written uniquely as: "]);
    Print(Xd1);
    Ma:=Concatenation([Ma,"a",String(j),"*","e",String(j)]);
    Print(" ","\n");
    Print(Ma);
    Ma:=Concatenation([Ma,"+"]);
    Mb:=Concatenation([Mb,"a",String(j),"*","d(e",String(j),")"]);
    Print(" ","\n");
    Print(Mb);
    Mb:=Concatenation([Mb,"+"]);
    if j>1 then
           Mc:=Concatenation([Mc,"a",String(j),"("]);
           for i in [1..j-1] do
               if i<j-1 then
                    Mc:=Concatenation([Mc,"f",String(i),String(j),"*","e",String(i),"+"]);
               else
                    Mc:=Concatenation([Mc,"f",String(i),String(j),"*","e",String(i),")"]);
               fi;
           od;
    fi;
    Print(" ","\n");
    Print(Mc);
    Mc:=Concatenation([Mc,"+"]);
    Md:=Concatenation(["  in  ","C",String(j-1)]);
    Print(Md);
    Print(" ","\n");Print(" ","\n");
    Me1:=Concatenation(["Hence ","d(C",String(j),") subset of C",String(j-1)," and
    then d(C",String(j),"/C",String(j-1),")=0."]);
    Print(Me1);
    Print(" ","\n"); Print(" ","\n");
    if j<m then
          Me2:=Concatenation([Me2,"C",String(j)," subset of " ]);
    else
          Me2:=Concatenation([Me2,"C",String(j),"= M is a composition series for M. " ]);
    fi;
od;
Print(Me2);
Print(" ","\n"); Print(" ","\n");
##
##############################################################################
##

return ("M is solvable");
end;
```

3. Solveindic2WithProof Function

```
Solveindic2WithProof:=function(dimf,m)
local i,j,f,d;
##
###############################################################################
##
## This function is called only if the conditions of Propositions 1.4.3
##(as in the manual) holds.
##
## This function is called if the modules M is outside the classification.
##
## The inputs of this function are the matrix dimf of dimensions and the
## dimension m of the vector of dimensions which are output by the main
## function  IsSolvableModuleWithProof.
### dimf and f are output by the main function IsSolvableModuleWithProof.
##
## The function outputs a proof that M is solvable.
################################################################################
##
f:=dimf;
##
################################################################################
##
## In this section we compute new entries for matrix f, by going through the
## entries of the matrix dimf and set f[i][j]= dimf[i][j] if dimf[i][j] < 0
## and f[i][j]=0 if dimf[i][j] >= 0, for i=1, ..., m, depending on the facts
## that in R, if dim (f) = j, i.e., f in R_j then degree of f = - j in the
## negative grading.
##
for j in [1..m-2] do
    for i in [1..m] do
        if i<j+2 then
             if dimf[i][j]<0 then
                 f[i][j]:=dimf[i][j];
             else
                 f[i][j]:=0;
             fi;
        else
             if dimf[i][j]<0 then
                  f[i][j]:=dimf[i][j];
             else
                  f[i][j]:=0;
             fi;
        fi;
    od;
od;
Print("\ f=",f);
Print(" ","\n");
##
################################################################################
##
## We compute the matrix d of the differential "partial" with respect to
```

```
## the basis S = e_i where i=1, ..., m.
##
for i in [1..m] do
    for j in [1..m] do
        if f[i][j]<>0 then
            f[i][j]:=Concatenation("f",String(i),String(j));
        fi;
    od;
od;
Print("\ d=",f);
Print(" ","\n");Print(" ","\n");
##
################################################################################
##
return("The module M is outside the classification");
end;
```

4. Solveindic3WithProof Function

```
Solveindic3WithProof:=function(m,dimf,f)
local i,j,diffk,dimej,dimei,f1,Cj,M1,M2,Cjb,Ca,Cja,Ma,Mb,Mc,Xd,Xd1,Md,Me1,Me2,Tranf;
##
################################################################################
##
## This function is called only if the conditions of Propositions 1.4.4
## (as in the manual) holds.
##
## The input of this function are:
###    m: the dimension of the vector of dimensions
### dimf: the matrix of the dimensions of the polynomials which is of size m x m,
###    f: the identity matrix of size m x m.
### m, dimf and f are output by the main function IsSolvableModuleWithProof.
##
## The function outputs a proof that M is solvable.
################################################################################
##
################################################################################
##
## In this section we compute new entries for matrix f, by going through the
## entries of the matrix dimf and set f[i][j]= dimf[i][j] if dimf[i][j] < 0
## and f[i][j]=0 if dimf[i][j] >= 0, for i=1, ..., m, depending on the facts
## that in R, if dim (f) = j, i.e., f in R_j then degree of f = - j in the
## negative grading.
##
for j in [1..m] do
    for i in [1..m] do
        if i>j then
             if dimf[i][j]>=0 then
                 f[i][j]:=0;
```

```
            else
                f[i][j]:=dimf[i][j];
            fi;
        else
            f[i][j]:=dimf[i][j];
        fi;
   od;
od;
##
##############################################################################
##
## In this section if f is an lower triangular matrix then we set f[i][i]
## to zero, using the fact that (partial)^2 =0 and R is an integral domain.
##
if IsLowerTriangularMat(f)=true then
    for i in [1..m] do
        f[i][i]:=0;
    od;
    Print("\ f=",f);
    Print(" ","\n");
else
    return("f is not upper triangular matrix");
fi;
##
##############################################################################
##
Tranf:=TransposedMatDestructive(f); # We have used TransposedMatDestructive(f) function,
                                    # because it will give us,the same result when we
                                    # use the rows and columns replacement.
Print("\ Tranf=",Tranf);
Print(" ","\n");
##
##############################################################################
##
## In this section we construct a proof that M is solvable if f is an
## upper triangular matrix.
##
if IsUpperTriangularMat(Tranf)=true then
    for i in [1..m] do
        for j in [1..m] do
            if Tranf[i][j]<>0 then
                Tranf[i][j]:=Concatenation("f",String(i),String(j));
            fi;
        od;
    od;
    Print("\ d=",Tranf);
else
    return("Maybe d is not upper triangular matrix or maybe it is");
fi;
Print(" , ( Since d^2=0 and R is an integral domain ). ");
Print(" ","\n");Print(" ","\n");
Cjb:=" ";
Ca:="Let C0=0 and ";
```

```
Print(Ca);
for j in [1..m] do
      Cja:=Concatenation(["C",String(j),"=<"]);
      for i in [1..j] do
          if i=j then
               M1:=Concatenation(["e",String(i)]);
          else
               M1:=Concatenation(["e",String(i),","]);
          fi;
               Cja:=Concatenation([Cja,M1]);
      od;
      if j=m then
               Cja:=Concatenation([Cja,">  "]);
      else
             Cja:=Concatenation([Cja,"> , "]);
      fi;
      Print(Cja);
      if j=m then
           Cjb:=Concatenation([Cjb,"C",String(j),"/","C",String(j-1)," is free "]);
      else
           Cjb:=Concatenation([Cjb,"C",String(j),"/","C",String(j-1)," is free, "]);
      fi;
od;
Print(" ","\n");
Print(Cjb);
Print(" ","\n");Print(" ","\n");
M2:=[];
Ma:="x=";
Mb:="d(x)=";
Mc:="d(x)=a1(0)";
Xd:="If x in C";
Me2:="Hence, 0=C0 subset of ";
for j in [1..m] do
    Xd1:=Concatenation([Xd,String(j),", then x can be written uniquely as: "]);
    Print(Xd1);
    Ma:=Concatenation([Ma,"a",String(j),"*","e",String(j)]);
    Print(" ","\n");
    Print(Ma);
    Ma:=Concatenation([Ma,"+"]);
    Mb:=Concatenation([Mb,"a",String(j),"*","d(e",String(j),")"]);
    Print(" ","\n");
    Print(Mb);
    Mb:=Concatenation([Mb,"+"]);
    if j>1 then
        Mc:=Concatenation([Mc,"a",String(j),"("]);
        for i in [1..j-1] do
            if i<j-1 then
                 Mc:=Concatenation([Mc,"f",String(i),String(j),"*","e",String(i),"+"]);
            else
                 Mc:=Concatenation([Mc,"f",String(i),String(j),"*","e",String(i),")"]);
            fi;
         od;
    fi;
```

```
    Print(" ","\n");
    Print(Mc);
    Mc:=Concatenation([Mc,"+"]);
    Md:=Concatenation(["  in  ","C",String(j-1)]);
    Print(Md);
    Print(" ","\n");Print(" ","\n");
    Me1:=Concatenation(["Hence ","d(C",String(j),") subset of C",String(j-1)," and
    then d(C",String(j),"/C",String(j-1),")=0."]);
    Print(Me1);
    Print(" ","\n"); Print(" ","\n");
    if j<m then
         Me2:=Concatenation([Me2,"C",String(j)," subset of " ]);
    else
         Me2:=Concatenation([Me2,"C",String(j),"= M is a composition series for M. " ]);
    fi;
od;
    Print(Me2);
    Print(" ","\n"); Print(" ","\n");
##
##############################################################################
##
return ("M is solvable.");
#return (f);
end;
```

5. Solveindic4Size2by2 Function

```
##
#############################################################################
##
## This function to convert the matrix degf to an upper triangular matrix.
##
## The input of the function Solveindic4Size2by2 is a matrix degf of
## size 2x2 which is output by the main function IsSolvableModuleWithProof.
##
## It returns the matrix degf after replacement and tests whether it is
## an upper triangular matrix or not.
##############################################################################
##
Solveindic4Size2by2:=function(degf)
degf[1][1]:=0; # Using the hypothesis of Proposition 1.4.2.
degf[2][2]:=0; # Using the hypothesis of Proposition 1.4.2.
degf[1][2]:=0; # Using (partial)^2 =0 (d^2=0 in this code) and R is an integral domain
degf:= StructuralCopy(degf);
degf:=SwapRowsColumns(degf,1,2);
##
##############################################################################
##
## This section to check whether degf is an upper triangular matrix or not
##
if IsUpperTriangularMat(degf)=false then
```

```
     Print("\ degf=",degf);
     Print(" ","\n");Print(" ","\n");
     Print("\ Thus, degf is not a strictly upper triangular matrix");
     Print(" ","\n");Print(" ","\n");
     else
        Print("\ degf=",degf);
        Print(" ","\n");Print(" ","\n");
        Print("\ Thus, degf is a strictly upper triangular matrix, so M is solvable.");
        Print(" ","\n");Print(" ","\n");
fi;
##
##############################################################################
##
return (degf);
end;
```

6. Solveindic4Size3by3 Function

```
Solveindic4Size3by3:=function(degf)
#local SwapRowsColumns;
##
#############################################################################
##
## This function to convert the matrix degf to an upper triangular matrix.
##
## The input of the function Solveindic4Size3by3 is a matrix degf of size
## 3x3 as in Remark 2.1(i) (it is case(1) of 3x3 matrix when f32=0).
## This function is called only if f11=f22=f33=0 and Sum(b)=0.
##
## It returns the matrix degf after replacement and tests whether it is
## a strictly upper triangular matrix or not.
##############################################################################
##
degf[3][2]:=0; # Using (partial)^2 =0 (d^2=0 in this code) and R is an integral domain
degf[1][2]:=0; # Using (partial)^2 =0 (d^2=0 in this code) and R is an integral domain
degf:= StructuralCopy(degf);   # creating duplicate of degf
degf:=SwapRowsColumns(degf,1,2);

if IsUpperTriangularMat(degf)=false then
     Print("\ degf=",degf);
     Print(" ","\n");Print(" ","\n");
     Print("\ Thus for the first case, degf is not a strictly upper triangular matrix");
     Print(" ","\n");Print(" ","\n");
else
     Print("\ degf=",degf);
     Print(" ","\n");Print(" ","\n");
     Print("\ Thus for the first case, degf is a strictly upper
     triangular matrix, so M is solvable.");
     Print(" ","\n");Print(" ","\n");
fi;
return (degf);
```

```
end;
```

7. Solveindic4Size4by4A Function

```
Solveindic4Size4by4A:=function(degf)
##
############################################################################
##
## This function to convert the matrix degf to an upper triangular matrix.
##
## The input of the function Solveindic4Size4by4A is a matrix degf of size
## mxm where m>=4 and f[i][i]=0, i=1,...,m with f32=0, f12=0, f32=0 and
## Sum(b)=0 as in Remark 2.1(ii).
##
## It returns the matrix degf after replacement and tests whether it is
## a strictly upper triangular matrix or not.
#############################################################################
##
degf[3][2]:=0; # Using (partial)^2 =0 (d^2=0 in this code) and R is an integral domain
degf[1][2]:=0; # Using (partial)^2 =0 and R is an integral domain
degf:= StructuralCopy(degf); # creating duplicate of degf
degf:=SwapRowsColumns(degf,1,2);
if IsUpperTriangularMat(degf)=true then
     Print("\ degf=",degf);
     Print(" ","\n");Print(" ","\n");
     Print("\ Thus for the First case,  degf is  a strictly upper Triangular matrix,
     so M is solvable.");
     Print(" ","\n");Print(" ","\n");
     else
        degf:= StructuralCopy(degf); # creating duplicate of degf
        degf:=SwapRowsColumns(degf,3,4);
        Print("\ degf=",degf);
        Print(" ","\n");Print(" ","\n");
        Print("\ Thus for the First case,  degf is  a strictly upper Triangular matrix,
        so M is solvable.");
        Print(" ","\n");Print(" ","\n");
fi;
return (degf);
end;
```

8. Solveindic4Size4by4B Function

```
Solveindic4Size4by4B:=function(degf)
local i,m;
##
############################################################################
##
## This function to convert the matrix degf to an upper triangular matrix.
##
## The input of the function Solveindic4Size4by4B is a matrix degf of size
```

```
## mxm where m>=4 such that f32<>0 and f21=0 with zeros on the diagonal and Sum(b)=0.
## The matrix degf of Remark 2.1(ii) is one example of the input of this function.
##
##
## It returns the matrix degf after replacement and tests whether it is
## a strictly upper triangular matrix or not.
##############################################################################
##
m:=Size(degf);
degf[2][1]:=0; # Using (partial)^2 =0 (d^2=0 in this code) and R is an integral domain
degf[2][3]:=0; # Using (partial)^2 =0 and R is an integral domain

degf:=SwapRowsColumns(degf,2,3);
if IsUpperTriangularMat(degf)=true then
     Print("\ degf=",degf);
     Print(" ","\n");Print(" ","\n");
     Print("\ Thus for the second case, degf is a strictly upper triangular matrix,
     so M is solvable.");
     Print(" ","\n");Print(" ","\n");
else
     degf:=SwapRowsColumns(degf,3,4);
     degf[1][3]:=0;      # Using (partial)^2 =0 and R is an integral domain
     for i in [4..m] do
         degf[1][i]:=0;  # Using (partial)^2 =0 and R is an integral domain
         degf[2][i]:=0;  # Using (partial)^2 =0 and R is an integral domain
     od;
     degf:=SwapRowsColumns(degf,3,4);
     degf:=SwapRowsColumns(degf,2,3);
     degf:=SwapRowsColumns(degf,3,4);
     Print("\ degf=",degf);
     Print(" ","\n");Print(" ","\n");
     Print("\ Thus for the second case,  degf is  a strictly upper triangular matrix,
     so M is solvable.");
     Print(" ","\n");Print(" ","\n");
fi;
return (degf);
end;
```

9. Solveindic4Size5by5 Function

```
Solveindic4Size5by5:=function(degf)
local i,j,m;
##
#############################################################################
##
## This function to convert the matrix degf to an upper triangular matrix.
##
## The input of the function Solveindic4Size5by5 is a matrix degf of size
## 5x5 with f32=0 and Sum(b)=2 as in Remark 2.1(v).
##
## It returns the matrix degf after replacement and tests whether it is
```

```
## a strictly upper triangular matrix or not.
##
##############################################################################
##
m:=Size(degf);
degf[3][2]:=0; # Using (partial)^2 =0 (d^2=0 in this code) and R is an integral domain
degf[1][2]:=0; # Using (partial)^2 =0 and R is an integral domain
##
###############################################################
##
## We will do the following steps, because we have that
## (partial)^2 =0 (d^2=0 in this code).
## These steps will help us to convert the matrix degf
## to an upper triangular matrix
##
for i in [1..m] do
     for j in [1..m] do
         if j>= i+2 then
             degf[i][j]:=0;
         fi;
    od;
od;
##
##############################################################
##
degf:= StructuralCopy(degf); # creating duplicate of degf
degf:=SwapRowsColumns(degf,1,2);
if IsUpperTriangularMat(degf)=false then
     degf:=SwapRowsColumns(degf,3,4);
fi;
if IsUpperTriangularMat(degf)=false then
     degf:=SwapRowsColumns(degf,4,5);
fi;
if IsUpperTriangularMat(degf)=false then
     degf:=SwapRowsColumns(degf,3,4);
fi;
if IsUpperTriangularMat(degf)=false then
     Print("\ degf=",degf);
     Print(" ","\n");Print(" ","\n");
     Print("\ Thus for the First case, degf is not a strictly upper triangular matrix.");
     Print(" ","\n");Print(" ","\n");
else
     Print("\ degf=",degf);
     Print(" ","\n");Print(" ","\n");
     Print("\ Thus for the First case, degf is a strictly upper triangular matrix,
     so M is solvable.");
     Print(" ","\n");Print(" ","\n");
fi;
return (degf);
end;
```

10. Solveindic4Size6by6 Function

```
Solveindic4Size6by6:=function(degf)
#local SwapRowsColumns;
##
#############################################################################
##
## This function to convert the matrix degf to an upper triangular matrix.
##
## The input of the function Solveindic4Size6by6 is a matrix degf of size
## 6x6. It is the first case of size 6x6 where f32=0 and b= [1,1,1 ]
## i.e., Sum(b)=3 as in Remark 2.1(vi).
##
## It runs the function SwapRowsColumns five times swapping rows and columns
## until degf is upper triangular matrix.
## It returns the matrix degf.
##
##############################################################################
##
##############################################################################
##
## The the following steps will help us to convert the matrix
## degf to an upper triangular matrix
##
degf:=SwapRowsColumns(degf,1,2);
degf:=SwapRowsColumns(degf,2,6);
degf:=SwapRowsColumns(degf,3,4);
degf:=SwapRowsColumns(degf,4,5);
degf:=SwapRowsColumns(degf,3,4);
##
##############################################################################
##
return (degf);
end;
```

11. Solveindic4Size6by6Above Function

```
Solveindic4Size6by6Above:=function(degf)
local i,j,mysize,mycounter,mycounter1,mycounter2,mycounter3;
##
#############################################################################
##
## This function to convert the matrix degf to an upper triangular matrix.
##
## The input of the function Solveindic4Size6by6Above is a matrix degf of
## size m x m where m>=6. It is case (1) of size >= 6x6 where f32=0, as in
## Remark 2.1(vii)
##
## It returns the matrix degf after replacement and tests whether it is
## a strictly upper triangular matrix or not.
##
```

```
##############################################################################
##
mysize:=Size(degf);
degf[3][2]:=0; # Using (partial)^2 =0 (d^2=0 in this code) and R is an integral domain
degf[1][2]:=0; # Using (partial)^2 =0 and R is an integral domain
##
###############################################################
##
## We will do the following because we have that (partial)^2 =0
## These steps will help us to convert the matrix degf to an
## upper triangular matrix
##
for i in [1..mysize] do
     for j in [1..mysize] do
         if j>= i+2 then
             degf[i][j]:=0;
         fi;
    od;
od;
##
###############################################################
##
## The following steps will help us to convert the matrix
## degf to an upper triangular matrix
##
if mysize<6 then
    return("mysize must be >=6");
elif mysize=6 then
          degf:=Solveindic4Size6by6(degf);
elif mysize=7 or mysize=8 then
              mycounter:=mysize -6;
              degf:=Solveindic4Size6by6(degf);
              for i in [1..mycounter] do
                 if i=1 then
                   degf:=SwapRowsColumns(degf, 4+i,6+i);
                   degf:=SwapRowsColumns(degf, 3+i,4+i);
                   degf:=SwapRowsColumns(degf, 1  ,3+i);
                   fi;
                   if i>1 then
                        degf:=SwapRowsColumns(degf, 4+i,6+i);
                        degf:=SwapRowsColumns(degf, 3+i,4+i);
                        degf:=SwapRowsColumns(degf, 1+i,3+i);
                        degf:=SwapRowsColumns(degf, 1  ,1+i);
                        degf:=SwapRowsColumns(degf, 2  ,1+i);
                   fi;
              od;
fi;
if mysize>=9 then
              mycounter:=mysize -6;
              degf:=Solveindic4Size6by6(degf);
              for i in [1..mycounter] do
                 if i=1 then
                   degf:=SwapRowsColumns(degf, 4+i,6+i);
```

```
                    degf:=SwapRowsColumns(degf, 3+i,4+i);
                    degf:=SwapRowsColumns(degf, 1  ,3+i);
                    fi;
                    if i>1 then
                         degf:=SwapRowsColumns(degf, 4+i,6+i);
                         degf:=SwapRowsColumns(degf, 3+i,4+i);
                         degf:=SwapRowsColumns(degf, 1+i,3+i);
                         degf:=SwapRowsColumns(degf, 1  ,1+i);
                         degf:=SwapRowsColumns(degf, 2  ,1+i);
                    fi;
               od;
               degf:= StructuralCopy(degf); # creating duplicate of degf
               mycounter1:=mysize -8;
               for mycounter2 in [1..mycounter1] do
                   for i in [1..mycounter2] do
                         mycounter3:=mycounter2-i+1;
                         degf:=SwapRowsColumns(degf, 2+mycounter3,3+mycounter3);
                   od;
               od;
fi;
if IsUpperTriangularMat(degf)=false then
     Print("\ degf=",degf);
     Print(" ","\n");Print(" ","\n");
     Print("\ Thus for the first case,  degf is not a strictly upper triangular matrix");
     Print(" ","\n");Print(" ","\n");
     else
        Print("\ degf=",degf);
        Print(" ","\n");Print(" ","\n");
        Print("\ Thus for the first case,  degf is a strictly upper triangular matrix,
        so M is solvable.");
        Print(" ","\n");Print(" ","\n");
        #return("Thus, M is solvable.");
fi;
return (degf);
end;
```

12. Solveindic4Sizembym Function

```
Solveindic4Sizembym:=function(degf)
local i,j,m;
##
############################################################################
##
## This function to convert the matrix degf to an upper triangular matrix.
##
## The input of the function Solveindic4Sizembym is a matrix degf of
## size m x m with m>=3, as in Remark 2.1(viii). It is case (1) of size >= 6x6 where f32=0,
## as in Remark 2.1(vii)
##
## The function outputs a proof that M is solvable for this case.
############################################################################
```

```
##
m:=Size(degf);
##
##############################################################
##
## We will do the following because we have that (partial)^2 =0
## (d^2=0 in this code) and R is an integral domain.
## These steps will help us to convert the matrix degf to an
## upper triangular matrix
##
degf[2][1]:=0;
degf[2][3]:=0;
for i in [1..m] do
     for j in [1..m] do
         if j>= i+2 then
             degf[i][j]:=0;
         fi;
    od;
od;
##
##############################################################
##
## After we set i=2 and j=m we run the function SwapRowsColumns
## while i<j with the input: SwapRowsColumns(degf,i,j) with
## setting i=i+1 and j=j-1. These steps will help us to convert
## the matrix degf to an upper triangular matrix
##
i:=2;
j:=m;
while i<j do
degf:=SwapRowsColumns(degf,i,j);
i:=i+1;
j:=j-1;
od;
##
##############################################################
##
## Tests whether the matrix degf is a strictly upper triangular matrix or not.
##
if IsUpperTriangularMat(degf)=true then
     Print("\ degf=",degf);
     Print(" ","\n");Print(" ","\n");
     Print("\ Thus for the second case, degf is a strictly upper triangular matrix,
     so M is solvable.");
     Print(" ","\n");Print(" ","\n");
     else
        Print("\ degf=",degf);
        Print(" ","\n");Print(" ","\n");
        Print("\ Thus for the second case, degf is not a strictly upper triangular matrix.");
        Print(" ","\n");Print(" ","\n");
fi;
##
##############################################################
```

```
##
return (degf);
end;
```

13. Solveindic4WithProof Function

```
Solveindic4WithProof:=function(degf)
local i,j,t,Temp3,Cas1,b,x,jt,S1,j1,Temp4,g,m;
##
##############################################################################
##
## This function is called only if the conditions of Propositions 1.4.2
## (as in the manual) holds.
##
## The input of this function is a matrix degf of size m x m which is output
## by the main function IsSolvableModuleWithProof.
##
## It calls the functions: Solveindic4Size3by3, Solveindic4Size4by4A,
## Solveindic4Size4by4B, Solveindic4Size5by5, Solveindic4Size6by6,
## Solveindic4Size6by6Above and Solveindic4Sizembym
##
## The function outputs a proof that M is solvable.
###############################################################################
##
m:=Size(degf);
Temp3:=[];
Temp3 := StructuralCopy(degf); # backup
i:=0;
Cas1:=2^(m-3); ## Cas1 is the number of the cases which are solvable
jt:=0;
for i in [1..Cas1] do #loop through the solvable cases
    degf:= StructuralCopy(Temp3);
    ##
    ###################################################################
    ##
    ## In this section we convert decimal to binary which it helps us
    ## to represents fij by 0 oR 1 for some specific i and j, such that
    ## fij are entries below the diagonal of degf
    b:=[];
    x:=jt;
    while x>0 do
          Add(b,x mod 2);
          x:=(x-(x mod 2))/2;
    od;
    jt:=jt+1;
    S1:=m-Size(b)-3;
    if S1<>0 then
        for t in [1..S1] do
            Add(b,0);
        od;
    fi;
```

```
##
####################################################################
##
## Set some entries of degf to zero, using the fact that
## (partial)^2 =0 and R is an integral domain
##
j1:=0;
degf := StructuralCopy(Temp3);
for j in [1..m-3] do
      j1:=j+3;
      if b[j]=0 then
           degf[j1][j1-1]:=0;
           degf[j1][j1]:=0;
      else  ## this case when b[j]=1
           degf[j1][j1]:=0;
           degf[j1-1][j1]:=0;
      fi;
od;

##
####################################################################
##
## If degf of size 3x3 we set f[i][i]=0 for i=1, ..., 3,
## using the hypothesis of Proposition 1.4.2
##
degf[3][3]:=0;
degf[2][2]:=0;
degf[1][1]:=0;
##
####################################################################
##
Temp4:=[];
Temp4:= StructuralCopy(degf); # backup 2

g:=Sum( b); ## g: Represents the sum of the entries of each vector b
degf:= StructuralCopy(Temp4);
if g=0 then ## This case represents the vector b when all the entries of b are zeros
   Print("\ b=",b);
   Print(" ","\n");Print(" ","\n");
   Print("\ i=",i);
   Print(" ","\n");Print(" ","\n");
   Print("\ degf Original Case_after setting some elements to Zero is ",degf);
   Print(" ","\n");Print(" ","\n");
 if m=3 then
    degf:=Solveindic4Size3by3(degf); ## It represents the first case when f32=0.
   fi;
 if m>=4 then
    degf:=Solveindic4Size4by4A(degf); ## It represents the first case when f32=0.
      degf:= StructuralCopy(Temp4);
      degf:=Solveindic4Size4by4B(degf); ## It represents the second case when f32<>0.
   fi;
fi;
if g=m-3 then # This case represents the vector b when all the entries of b are Ones.
```

```
        Print("\ b=",b);
        Print(" ","\n");Print(" ","\n");
        Print("\ i=",i);
        Print(" ","\n");Print(" ","\n");
        Print("\ degf Original Case_after setting some elements to Zero is ",degf);
        Print(" ","\n");Print(" ","\n");
        if m=3 then
           degf:= StructuralCopy(Temp4);
           degf:=Solveindic4Sizembym(degf); ## It represents the second case when f32<>0.
        fi;
        if m=4 then
           degf:=Solveindic4Size4by4A(degf); ## It represents the first case when f32=0.
           degf:= StructuralCopy(Temp4);
           degf:=Solveindic4Sizembym(degf);  ## It represents the second case when f32<>0.
        fi;
        if m=5 then
         degf:=Solveindic4Size5by5(degf); ## It represents the first case when f32=0.
             degf:= StructuralCopy(Temp4);
             degf:=Solveindic4Sizembym(degf); ## It represents the second case when f32<>0.
        fi;
        if m>=6 then
           degf:=Solveindic4Size6by6Above(degf); ## It represents the first case when f32=0.
           degf:= StructuralCopy(Temp4);
           degf:=Solveindic4Sizembym(degf);  ## It represents the second case when f32<>0.
        fi;
     fi;
od; ############## End of The Loop of The Solvable Cases.
return("M is solvable.");
end;
```

14. SolvableModuleByUsualGradedWithProof Function

```
SolvableModuleByUsualGradedWithProof:=function(D,P)
local i,j,m,k1,k2,t,dimf,degf,f,diffk,dimej,dimei,f1,Cj,M1,M2,Cjb,Ca,Cja,Ma,
Mb,Mc,Xd,Xd1,Md,Me1,Me2,indic,indic1,x1,x2,x3,td,Temp1,Temp2,degf2,f12,Temp3;
##
############################################################################
##
## The function SolvableModuleByUsualGraded is called only if the conditions
## of Proposition 1.4.5 (as in the manual) hold.
##
## The inputs of this function are the list of dimensions of the modules
## D=[k_1, ..., k_n] where dim(e_i) = k_i  and the degree P of the
## differential on the module M. (The same inputs as the main function
## IsSolvableModuleWithProof.)
##
## The function outputs a proof that M is solvable.
##
#############################################################################
##
m:=Size(D);
```

```
f1:=IdentityMat(m);
k1:=D[1];
j:=0;
t:=[];
dimf:=IdentityMat(m);
f:=IdentityMat(m);
##
##############################################################################
##
## In this section we generate the dimf-matrix following the hypothesis of
## Proposition 1.4.5
##
for j in [1..m] do
    dimej:=D[j];
    for i in [1..m] do
        dimei:=D[i];
        dimf[i][j]:=dimej-dimei-P;
        if dimf[i][j]<0 then
           dimf[i][j]:=0;
        fi;
        degf[i][j]:=-1*dimf[i][j];
    od;
od;
Print(" ","\n");Print(" ","\n");
Print("\ dimf=",dimf);
Print(" ","\n");
##
##############################################################################
##
## In this section we compute new entries for matrix f, by going through the
## entries of the matrix dimf and set f[i][j]= dimf[i][j] if dimf[i][j] >= 0
## and f[i][j]=0 if dimf[i][j] < 0, for i=1, ..., m, depending on the facts
## that in R, if dim (f) = j, i.e., f in R_j then degree of f = - j in the
## unusual grading and any f of degree less than 0 it will be 0.
##
for j in [1..m] do
    for i in [1..m] do
        if i>j then
             if dimf[i][j]<0 then
                  f[i][j]:=0;
             else
                  f[i][j]:=dimf[i][j];
             fi;
        else
                  f[i][j]:=dimf[i][j];
        fi;
    od;
od;
Print("\ f=",f);
Print(" ","\n");
##
###############################################################
##
```

```
## Tests whether the matrix f is an upper triangular matrix or not.
## If f is an upper triangular we set f[i][i] to 0 where i=1,..., m
## using the hypothesis of Proposition 1.4.5. Then compute the
## matrix d of the differential "partial" with respect to the
## basis S ={ e_i, ..., e_m}.
##
if IsUpperTriangularMat(f)=true then
    for i in [1..m] do
          f[i][i]:=0;
    od;
    for i in [1..m] do
          for j in [1..m] do
              if f[i][j]<>0 then
                 f[i][j]:=Concatenation("f",String(i),String(j));
              fi;
          od;
    od;
    Print("\ d=",f);
else
    return("f is not upper triangular matrix");
fi;
##
###############################################################
##
## In this section we construct a proof that M is solvable
## if f is an upper triangular matrix.
##
Print(" , ( Since d^2=0 and R is an integral domain ). ");
Print(" ","\n");Print(" ","\n");
Cjb:=" ";
Ca:="Let C0=0 and ";
Print(Ca);
for j in [1..m] do
    Cja:=Concatenation(["C",String(j),"=<"]);
    for i in [1..j] do
        if i=j then
             M1:=Concatenation(["e",String(i)]);
        else
             M1:=Concatenation(["e",String(i),","]);
        fi;
             Cja:=Concatenation([Cja,M1]);
    od;
    if j=m then
         Cja:=Concatenation([Cja,">  "]);
    else
         Cja:=Concatenation([Cja,"> , "]);
    fi;
    Print(Cja);
    if j=m then
         Cjb:=Concatenation([Cjb,"C",String(j),"/","C",String(j-1)," is free. "]);
    else
         Cjb:=Concatenation([Cjb,"C",String(j),"/","C",String(j-1)," is free, "]);
    fi;
```

```
od;
Print(" ","\n");
Print(Cjb);
Print(" ","\n");Print(" ","\n");
M2:=[];
Ma:="x=";
Mb:="d(x)=";
Mc:="d(x)=a1(0)";
Xd:="If x in C";
Me2:="Hence, 0=C0 subset of ";
for j in [1..m] do
    Xd1:=Concatenation([Xd,String(j),", then x can be written uniquely as: "]);
    Print(Xd1);
    Ma:=Concatenation([Ma,"a",String(j),"*","e",String(j)]);
    Print(" ","\n");
    Print(Ma);
    Ma:=Concatenation([Ma,"+"]);
    Mb:=Concatenation([Mb,"a",String(j),"*","d(e",String(j),")"]);
    Print(" ","\n");
    Print(Mb);
    Mb:=Concatenation([Mb,"+"]);
    if j>1 then
         Mc:=Concatenation([Mc,"a",String(j),"("]);
         for i in [1..j-1] do
             if i<j-1 then
                  Mc:=Concatenation([Mc,"f",String(i),String(j),"*","e",String(i),"+"]);
             else
                  Mc:=Concatenation([Mc,"f",String(i),String(j),"*","e",String(i),")"]);
             fi;
         od;
    fi;
    Print(" ","\n");
    Print(Mc);
    Mc:=Concatenation([Mc,"+"]);
    Md:=Concatenation(["  in  ","C",String(j-1)]);
    Print(Md);
    Print(" ","\n");Print(" ","\n");
    Me1:=Concatenation(["Hence ","d(C",String(j),") subset of C",String(j-1)," and
    then d(C",String(j),"/C",String(j-1),")=0."]);
    Print(Me1);
    Print(" ","\n"); Print(" ","\n");
    if j<m then
         Me2:=Concatenation([Me2,"C",String(j)," subset of " ]);
    else
         Me2:=Concatenation([Me2,"C",String(j),"= M is a composition series for M. " ]);
    fi;
od;
Print(Me2);
Print(" ","\n"); Print(" ","\n");

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

```
return("M is solvable.");
end;
```

15. IsSolvableModuleWithProof Function

```
IsSolvableModuleWithProof:=function(D,P)
local i,j,m,k1,k2,t,dimf,degf,f,diffk,dimej,dimei,f1,indic,indic1,
x1,x2,x3,td,Case1,Case2,Case3,Case4,Case5,Temp1,Temp2,degf2,f12,
Temp3,t1,t2,sumt,S,B;
##
############################################################################
##
## The function IsSolvableModuleWithProof is the main function of our algorithm.
## It checks which of the conditions of Propositions 1.4.1, 1.4.2, 1.4.4, 1.4.5
## or Remark 1.4.3 hold (see the manual). Then it calls one of the functions:
## Solveindic1WithProof, Solveindic2WithProof, Solveindic3WithProof,
## Solveindic4WithProof and SolvableModuleByUsualGradedWithProof according
## to the condition that matches the function.
##
## The inputs of this function are the list of dimensions of the modules
## D=[k_1, ..., k_n] where dim(e_i) = k_i  and the degree P of the
## differential on the module M.
##
## The function outputs the dimension m of the vector of dimensions,
## the matrix dimf of dimensions, the identity matrix f of size mxm,
## the matrix degf of degrees, the flags indic and x_i; i=1,2,3 to
## determine which of Solveindic(n)WithProof function to run; where n=1,..., 4.
#############################################################################
##
m:=Size(D);
if P=1 or P=-1 then  ## With the usual graded or negative graded
   Print(" ","\n");Print(" ","\n");
   return("Then, M is solvable (by Carlsson,1983).");
fi;
if P<=-2 then  ## Negative graded
   f1:=IdentityMat(m); ####
   k1:=D[1]; # k1 represents dim(e_1)
   j:=0;
   t:=[];
   dimf:=IdentityMat(m);
   degf:=IdentityMat(m);
   degf2:=IdentityMat(m);#####
   f:=IdentityMat(m);
   ##
   ############################################################################
   ##
   ## In this section we set the flags "indic" and x_i; i=1,2,3, by using the
   ## degree P. These flags are used to determine which of "Solveindic(n)WithProof";
   ## n=1,...,4 functions to run, after checking the conditions of Propositions
```

```
## 1.4.1, 1.4.2, 1.4.4  and Remark 1.4.3.
##
indic:=0;
x1:=0;
x2:=0;
x3:=0;
for i in [2..m] do
    j:=j+1;
    k2:=D[i];
    diffk:=k1-k2; ## This step finds that diffk=k(i)-k(i+1)
    Print("\ diffk=",diffk);
    Print(" ","\n");Print(" ","\n");
    if k1>k2 then
       t[j]:=diffk;
       if diffk>=-P then
            indic:=1; # It means Propositions 1.4.1 holds
            x1:=x1+1;
       elif diffk<-P then
            indic:=2; # It means Propositions 1.4.3 holds
            x2:=x2+1;
       fi;
       k1:=k2;
    else
       if diffk<P then
           indic:=3; # It means Propositions 1.4.4 holds
           x3:=x3+1;
       fi;
    fi;
    k1:=k2;
od;
if indic=1 then
    if x1<m-1 then
       return("Not True2 (the conditions of this Proposition 1.4.1 must be satisfied)");
    fi;
elif indic=2 then
    if x2<m-1 then
       return("Not True3 (the conditions of this Proposition 1.4.3 must be satisfied)");
    fi;
elif indic=3 then
    if x3<m-1 then
       return("Not True4 (the conditions of this Proposition 1.4.4 must be satisfied)");
    fi;
fi;
if indic=2 then  # Case two when t(i)+ t(i+1)<=-P
    x1:=0;
    x2:=0;
    j:=0;
    td:=[];
    t1:=t[1];
    for i in [2..m-1] do
          j:=j+1;
          t2:=t[i];
          sumt:=t1+t2;
```

```
          td[j]:=sumt;
          if sumt<=-P then
             x1:=x1+1;
             indic:=2; # It means Propositions 1.4.3 holds (when t(i)+ t(i+1)<=-P)
          else
             indic:=4; # It means Propositions 1.4.2 holds (when t(i)+ t(i+1)>-P)
             x2:=x2+1;
          fi;
          t1:=t2;
    od;
    if x1<m-2 and x2<m-2  then
         return("Not True6");
    fi;
fi;
Print("\ indic=",indic);
Print(" ","\n");Print(" ","\n");
##
##############################################################################
##
##############################################################################
##
## In this section we compute the matrix dimf of dimensions of the elements
## f_ij; i,j=1, ..., m of the matrix of the differential "partial" with
## respect to the basis S ={ e_i, ..., e_m}.
## Also we compute a matrix degf of degrees of f_ij, by seting
## degf[i][j]=-dimf[i][j] where i,j=1, ..., m.
##

for j in [1..m] do
  dimej:=D[j];
  for i in [1..m] do
      dimei:=D[i];
      dimf[i][j]:=dimej-dimei+P;
      if dimf[i][j]>0 then
          dimf[i][j]:=0;
      fi;
      degf[i][j]:=-1*dimf[i][j];
   od;
od;
Print("\ dimf=",dimf);
Print(" ","\n");Print(" ","\n");
Print("\ degf=",degf);
Print(" ","\n");
##
##############################################################################
##
################################### START-----Case one ######################
if indic=1 then
   Case1:=Solveindic1WithProof(dimf,f);
fi;
################################### END-----Case One ########################
##
################################### START-----Case Two ######################
```

```
   if indic=2 or (indic=4 and m=2) then
      # (Since there is a common condition between them which is when m=2 and f11=f22=0)
      if m=2 then
          Case4:=Solveindic4Size2by2(degf);
          Print("\ Hence, if f11=f22=0 then the module M is solvable. Otherwise M
          outside the classification.");
          Print(" ","\n");Print(" ","\n");
      else
          Case2:=Solveindic2WithProof(dimf,m);
      fi;
   fi;
   ################################### END-----Case Two #########################
   ##
   ################################### START-----Case Three #####################

   if indic=3 then
      Case3:=Solveindic3WithProof(m,dimf,f);
   fi;
   ################################### END-----Case Three #######################
   ##
   ################################### START-----Case Four ######################
   if indic=4 then
      Case4:=Solveindic4WithProof(degf);
   fi;
   ################################### END-----Case Four ########################
   ##
   ################################### START-----Rerurn Cases 1-4 ###############
   if indic=1 then
       return(true);
   fi;
   if indic=2 and m<>2 then
       return(fail);
   fi;
   if indic=3 then
       return(true);
   fi;
   if indic=4 then
       return(true);
   fi;
   ################################### END-----Rerurn Cases 1-4 #################
fi;
################################### START-----Case Five ########################
##
## In this section we satisfy the conditions of Proposition 1.4.5
##
S:=1;
if P>=2 then  ## With the usual graded
   for i in [1..m-1] do
       diffk:=D[i+1]-D[i];
       Print(" ","\n");
       Print("\ diffk=",diffk);
       Print(" ","\n");
```

```
        if D[i]< D[i+1] and diffk>P then
            B:=1;
        else
            B:=0;
        fi;
        S:= S*B;
    od;
    if S=1 then
         Case5:=SolvableModuleByUsualGradedWithProof(D,P);
    else
        Print(" ","\n");Print(" ","\n");
        return("The input must be P>=2 and D[1]<D[2]<...<D[m] and
        D[i+1]-D[i]>P for i in [1..m]");
    fi;
fi;
return(true);
################################### END-----Case Five ##############################
end;
```

Bibliography

[1] A.J. AL-Juburie and A.J. Duncan, *AutParCommGrp(Finite Presentations of Automorphism Groups of Partially Commutative Groups and Their Subgroups) package*, 2015, GAP System Library.

[2] M. Aldrich and J.R. Rozas, *Exact and semisimple differential graded algebras*, Comm. Algebra **30** (2002), 1053–1075.

[3] M. Amasaki, *Generators of graded modules associated with linear filter-regular sequences*, Journal of Pure and Applied Algebra **114** (1996), 1–23.

[4] M. Angel and R. Dlaz, *On n-differential graded algebras*, Journal of Pure and Applied Algebra **210(3)** (2007), 673–683.

[5] L.L. Avramov and R. Buchweitz, *Homological algebra modulo a regular sequence with special attention to codimension two*, Journal of Algebra **230.1** (2000), 24–67.

[6] L.L. Avramov, H. Foxby, and L. Halperin, 1999, manuscript.

[7] L.L. Avramov and D.R. Grayson, *Resolutions and cohomology over complete intersections*, Computations in algebraic geometry with Macaulay 2, Algorithms and Computations in Mathematics 8, Springer (2002), 131–178.

[8] A. Baudisch, *Subgroups of semifree groups*, Acta Math. Acad. Sci. Hungar **(1-4)** (1981), 19–28.

[9] K.A. Beck, *On the image of the totaling functor*, Communications in Algebra **43.4** (2015), 1640–1653.

[10] J. Bernstein and V. Lunts, *Equivariant sheaves and functors*, Springer, 1994.

[11] M. Bestvina and N. Brady, *Morse theory and finiteness properties of groups*, Invent. Math. **129** (1997), 445–470.

[12] M. Bestvina, B. Kleiner, and M. Sageev, *The asymptotic geometry of right-angled Artin groups, I*, Geometry and Topology **12** (2008), 1653–1700.

[13] J.A. Bondy and U.S.R. Murty, *Graph theory with aplications, first edition*, The Macmillan Press LTD, 1976.

[14] K. Bux, R. Charney, J. Crisp, and K. Vogtmann, *Automorphisms of two-dimensional RAAGs and partially symmetric automorphisms of free groups*, Groups Geom. Dyn. **3(4)** (2009), 541–554.

[15] G. Carlsson, *On homology of finite free* $(z/2)^k$*-complexes*, Invent. Math. **74** (1983), 139–147.

[16] R. Charney, *An introduction to right-angled Artin groups*, Geom. Dedicata **125** (2007), 141–158.

[17] R. Charney, J. Crisp, and K. Vogtmann, *Automorphisms of 2-dimensional right-angled Artin groups*, Geom. Topol. **11** (2007), 2227–2264.

[18] R. Charney and M. Farber, *Random groups arising as graph products*, Algebraic and Geometric Topology **12** (2012), 979–995.

[19] R. Charney and K. Vogtmann, *Finiteness properties of automorphism groups of right-angled Artin groups*, Bull. Lond. Math. Soc. **41(1)** (2009), 94–102.

[20] ______, *Subgroups and quotients of automorphism groups of RAAGs*, Low-dimensional and symplectic topology **82(9)** (2011), 1–19.

[21] M. Cohen and L.H. Rowen, *Group graded rings*, Comm. Algebra **11(11)** (1983), 1253–1270.

[22] M. F. A. Couette, *Études sur le frottement des liquides*, Annales de Chimie et de Physique **21** (1890), 433–510.

[23] E.C. Dade, *Group-graded rings and modules*, Math. Z. **174(3)** (1980), 241–262.

[24] M.B. Day, *Peak reduction and finite presentations for automorphism groups of right-angled Artin groups*, Geometry and Topology **13** (2009), 817–855.

[25] ______, *On solvable subgroups of automorphism groups of right-angled Artin groups*, IJAC: Proceedings of the 2009 International Conference on Geometric and Combinatorial Methods in Group Theory and Semigroup Theory **21(1-2)** (2011), 61–70.

[26] ______, *Finiteness of outer automorphism groups of random right-angled Artin groups*, Algebraic and Geometric Topology **12** (2012), 1553–1583.

[27] ______, *Full-featured peak reduction in right-angled Artin groups*, Algebraic and Geometric Topology **14** (2014), 1677–1743.

[28] C. Droms, *Graph groups, coherence, and three-manifolds*, J. Algebra **106(2)** (1987), 484–489.

[29] ______, *Isomorphisms of graph groups*, Proc. Amer. Math. Soc. **100(3)** (1987), 407–408.

[30] ______, *Subgroups of graph groups*, J. Algebra **110(2)** (1987), 519–522.

[31] J.A. Drozd, *Tame and wild matrix problems. In: Representation theory II. lecture notes in mathematics, vol.832, pp.242-258*, Springer, Berlin,, 1980.

[32] D. Dugger and B. Shipley, *Topological equivalences for differential graded algebras*, Advances in Mathematics **212** (2007), 37–61.

[33] D. Dummit and M. Foote, *Abstract algebra*, John Wiley And Sons, third edition, New York, 2004.

[34] A.J. Duncan, I.V. Kazachkov, and V.N. Remeslennikov, *Automorphisms of partially commutative groups I: Linear subgroups*, Groups, Geometry, and Dynamics **4(4)** (2010), 739–757.

[35] A.J. Duncan and V.N. Remeslennikov, *Automorphisms of partially commutative groups II: Combinatorial subgroups*, International Journal of Algebra and Computation **22(7)** (2012), 1250074.

[36] E.S. Esyp, I.V. Kazachkov, and V.N. Remeslennikov, *Divisibility theory and complexity of algorithms for free partially commutative groups*, Contemporary Mathematics, Groups, Languages, Algorithms **378** (2005), 319–348.

[37] M. Ferrero and E. Jespers, *prime ideals of graded rings and related matters*, Communications in Algebra **18(11)** (1991), 3819–3834.

[38] M. Gutierrez and S. Krstic, *Normal forms for basis-conjugating automorphisms of a free group*, Int. J. Algebra Comput. **8** (1998), 631–669.

[39] M. Gutierrez, A. Piggott, and K. Ruane, *On the automorphisms of a graph product of abelian groups*, Groups Geom. Dyn. **6** (2012), 125–153.

[40] ______, *On the automorphisms of a graph product of abelian groups*, Groups, Geometry and Dynamics **6(1)** (2012), 125153.

[41] S.P. Humphries, *On representations of Artin groups and the Tits conjecture*, J. Algebra **169** (1994), 847862.

[42] T. Hungerford, *Algebra*, Springer-Verlang, New York, 1974.

[43] J.F. Jardine, *A closed model category structure for differential graded algebras, Cyclic cohomology and noncommutative geometry (Waterloo, ON, 1995)*, Fields Inst. Commun., vol. 17, Amer. Math. Soc., Providence, RI (1997), 55–58.

[44] C. Jensen and J. Meier, *The cohomology of right-angled Artin groups with group ring coefficients,bull*, London Math. Soc. **37** (2005), 711–718.

[45] E. Jespers, *Radicals of graded rings*, Colloq. Math. Soc. J. Bolyai 61, North Holland, Amsterdam **61** (1993), 109–130.

[46] B. Keller, *On differential graded categories*, International Congress of Mathematicians, Eur. Math. Soc., Zurich **II** (2006), 151–190.

[47] G.M. Kelly, *Chain maps inducing zero homology maps*, Proc. Cambridge Philos. Soc. **61** (1965), 847–854.

[48] S. Kim and F.W. Roush, *Homology of certain algebras defined by graphs*, J. Pure Appl. Algebra **17** (1980), 179–186.

[49] H. Koberda, *Right-angled Artin groups and their subgroups*, 2013, An advanced mathematical course, Yale University, USA, http://users.math.yale.edu/users/koberda/raagcourse.pdf.

[50] T.Y. Lam, *A first course in non commutative rings*, Springer-Verlag, New York, 1991.

[51] M. Laurence, *A generating set for the automorphism group of a graph group*, J. London Math. Soc. **52(2)** (1995), 318–334.

[52] A. Legrand, *Differential graded modules over a nonconnected differential graded algebra*, Journal of Pure and Applied Algebra **72** (1991), 53–66.

[53] M. Lohrey and S. Schleimer, *Efficient computation in groups via compression*, In Volker Diekert, Mikhail Volkov, and Andrei Voronkov, editors, Computer Science Theory and Applications, volume 4649 of Lecture Notes in Computer Science, Springer Berlin /Heidelberg **4649** (2007), 249–258.

[54] S. Maclane, *Homology*, Springer-Verlag, New York, 1995.

[55] X. Mao, *A criterion for a connected DG algebra to be homologically smooth*, arXiv:1301.4382 **4** (2013).

[56] J. McCool, *Some finitely presented subgroups of the automorphism group of a free group*, J. Algebra **35(6)** (1975), 205–213.

[57] ______, *On basis-conjugating automorphisms of free groups*, Canadian J. Math. **38(6)** (1986), 1525–1529.

[58] A. Minasyan, *Hereditary conjugacy separability of right angled Artin groups and its applications*, Groups Geometry and Dynamics **6** (2012), 335–388.

[59] C. Nastasescu and F. Van Oystaeyen, *Graded ring theory*, Mathematical Library, (28), North Holland, Amsterdam, 1982.

[60] G.A. Noskov, *The image of the automorphism group of a graph group under abelianization map*, Vestnik NGU, Mat., Mekh. **12(2)** (2012), 83–102.

[61] L.A. Orlandi-Korner, *The Bieri-Neumann-Strebel invariant for basis-conjugating automorphisms of free groups*, Proceedings of the American Mathematical Society **128(5)** (2000), 1257–1262.

[62] S. Papadima and A.I. Suciu, *Algebraic invariants for right-angled Artin groups*, Math. Ann. **334** (2006), 533–555.

[63] D. Pauksztello, *Homological properties of differential graded algebras*, Ph.D. thesis, Department of Pure Mathematics, Leeds University, 2008.

[64] M. Refai, *Group actions on finite CW-complexes*, Ph.D. thesis, Department of Mathematics, Colorado State University, 1989.

[65] ———, *On noetherian modules graded by g-sets*, Acta Mathematica Hungarica **69(3)** (1995), 211–219.

[66] M. Refai and M. Obiedat, *On graduations of* $k[x_1, x_2, \cdots, x_n]$, J. of Institute of Math and Computer Sci. **6(3)** (1993), 241–252.

[67] A.V. Roiter, *Matrix problems, proceedings of the international congress of mathematicians (Helsinki,1978)*, Acad. Sci. Fennica (1980), 319–322.

[68] K.H. Rosen, *Discrete mathematics and its applications, sixth edition*, McGraw-Hill, New York, 2007.

[69] H. Servatius, *Automorphisms of graph groups*, J. Algebra **126** (1989), 34–60.

[70] E. Toinet, *A finitely presented subgroup of the automorphism group of a right-angled Artin group*, Journal of Group Theory **15(6)** (2012), 811–822.

[71] R.D. Wade, *The lower central series of a right-angled Artin group*, The Quarterly Journal of Mathematics; doi: 10.1093/qmath/hat002 (2013).

[72] J.H.C. Whitehead, *On equivalent sets of elements in a free group*, Ann. of Math. **(2)37** (1936), 782–800.