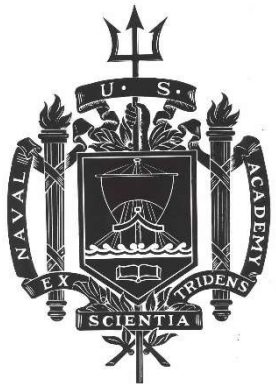
A TRIDENT SCHOLAR PROJECT REPORT

NO. 500

Decision Problems in Computational Group Theory

by

Midshipman 1/C Aidan J. Sabety-Mass, USN



UNITED STATES NAVAL ACADEMY
ANNAPOLIS, MARYLAND

USNA-1531-2

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From - To)*
7/6/20		

4. TITLE AND SUBTITLE
Decision Problems in Computational Group Theory

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Sabety-Mass, Aidan J.

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Naval Academy
Annapolis, MD 21402

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
Trident Scholar Report no. 500 (2020)

12. DISTRIBUTION / AVAILABILITY STATEMENT

This document has been approved for public release; its distribution is UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

In this report we discuss a proof following the ideas of Gabor Elek and Endre Szab that Kaplansky's conjecture is satisfied for group algebras over finite groups and arbitrary fields; as well as, add detail to a proof that Gottschalk's conjecture implies Kaplansky's conjecture over fields with positive characteristics. We further present two algorithms: one is a unique algorithm that, if given an infinite amount of time, can in principle iteratively check Kaplansky's conjecture for all finite groups. The other is a unique implementation of a paper by Dykema, Heister and Juschenko that can check infinite groups from the class of Universal Left Invertible Element (ULIE) groups.

15. SUBJECT TERMS
Kaplansky's conjecture, computational group theory

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**		53	**19b. TELEPHONE NUMBER** *(include area code)*

U.S.N.A – Trident Scholar Project Report; No. 500 (2020)

DECISION PROBLEMS IN COMPUTATIONAL GROUP THEORY

by

Midshipman 1/C Aidan J. Sabety-Mass
United States Naval Academy
Annapolis, Maryland

_____

(signature)

Certification of Advisers Approval

Dr. Kostya Medynets
United States Naval Academy
Annapolis, Maryland

_____

(signature)

_____

(date)

Acceptance for the Trident Scholar Committee

Professor Maria J. Schroeder
Associate Director of Midshipman Research

_____

(signature)

_____

(date)

USNA-1531-2

Acknowledgements

I would first like to thank Professor Kostya Medynets, for the multiple 9 pm nights he spent working with me. His guidance and feedback informs me to this day and will continue to help me mature as both an academic and an officer. I would like to thank my friends for always supporting me, despite me having to ditch them most weekends. I would also like to thank my girlfriend for letting this project take up a part of our relationship. Lastly, I would like to thank my family, both blood and non-blood, for always fighting and caring for me. They gave me the imagination to believe that someone with a GED could graduate college.

One certainly looks at their Trident Scholar Project as the culmination of their undergraduate career. I am no different. It is an honor to have been selected and an honor to be amongst such an incredible cohort of such intelligent and genuine people. I appreciate all of the readers and committee members who donate and donated hours and days of their time to help support such a rigorous, special, and unique program. Thank you.

Trident Final Report: "Decision Problems in Computational Group Theory"

MIDN 1/C Aidan J. Sabety-Mass
Adviser: Associate Professor Kostya Medynets

May 8, 2020

Abstract

In this report we discuss a proof following the ideas of Gabor Elek and Endre Szabo [3] that Kaplansky's conjecture is satisfied for group algebras over finite groups and arbitrary fields; as well as, add detail to a proof that Gottschalk's conjecture implies Kaplansky's conjecture over fields with positive characteristics. We further present two algorithms: one is a unique algorithm that, if given an infinite amount of time, can in principle iteratively check Kaplansky's conjecture for all finite groups. The other is a unique implementation of a paper by Dykema, Heister and Juschenko [4] that can check infinite groups from the class of Universal Left Invertible Element (ULIE) groups.

1 Introduction

Kaplansky's Direct Finiteness Conjecture states that for any group G and commutative field K, the group algebra $K(G)$ is directly finite, that is, $ab = 1$ in $K(G)$ implies $ba = 1$. This conjecture has been open for 50 years [9].

In 1969, Irving Kaplansky proposed and proved that for fields of characteristic zero this conjecture is satisfied [9]. Pere Ara, Kevin O'Meara and Francesc Perera in 2002 proved Kaplansky's conjecture holds for *residually amenable groups* [21]. Gabor Elek and Endre Szabo proved in 2004 that for a large class of groups called *sofic groups* this conjecture also holds. Sofic groups were first defined by Mikhail Gromov as a common generalization of amenable groups and residually finite groups. Interestingly, there are no known examples of non-sofic groups — thus it is an open question on whether they exist. Any group that fails Kaplansky's conjecture would then be the first example of a non-sofic group [3].

Kaplansky's conjecture is further related to a famous conjecture in dynamical systems called Gottschalk's conjecture [22]. This conjecture states that if a given countable group G, a finite set X, and the compact metrizable space X^G of X-valued functions on G are

equipped with the product topology such that $f : X^G \to X^G$ is a continuous map that commutes with the natural right G-action; then, if f is injective, it is surjective as well [3].

It has been proven that if a group satisfies Gottchalk's conjecture, then it satisfies Kaplansky's conjecture [3]. We note that the authors provided a sketch of the proof, and one of the main goals of this project has been to reconstruct this proof in detail, see Section 3. In [14] Mikhail Gromov, the researcher who defined sofic groups, proved that all sofic groups satisfy Gottschalk's conjecture. In [20], Benjamin Weiss presents an alternative proof of this.

Recently in the paper, *Finitely Presented Groups Related to Kaplansky's Direct Finiteness Conjecture* [4], Dykema, Heister, and Juschenko, proposed a mathematical algorithm that if implemented and given enough time, can eventually find a counterexample to Kaplansky's Conjecture — if one such exists.

Our interest in this subject is motivated by finding a counterexample to Kaplansky's conjecture which, as explained above, would be a counterexample to Gottschalk's conjecture; as well as, an example of a non-sofic group.

This paper is split into two parts. Part I focuses on our theoretical results, mainly a proof of Kaplansky's conjecture for finite groups; as well as, a proof that Gottschalk's conjecture implies Kaplansky's for group algebras of infinite groups with fields of positive characteristic. In Part II we present two algorithms, a uniquely designed algorithm and implementation for checking Kaplansky's conjecture for the class of finite groups and the other a unique implementation of an algorithm from [4] for the class of infinite Universal Left Invertible Element (ULIE) groups.

The structure of our paper follows:

In Section 2 we present the necessary background on groups, fields, Kaplansky's and Gottschalk's conjecture, and the word problem in rewriting systems.

Section 3 presents our work discussing a proof that Gottschalk's implies Kaplansky's conjecture for finite groups; gives a proof of Kaplansky's conjecture for finite groups, and a proof that Gottschalk's conjecture implies Kaplanksy's conjecture for infinite groups.

In Section 4 we present our algorithm and implementation for checking Kaplanksy's conjecture for finite groups. This was done by representing each symmetric group as a matrix group and the group algebra of each symmetric group over a finite field as a special ring of matrices, thus replacing group algebra computations with matrix operations. Justification for this switch is presented in this section.

In Section 5 we present the algorithm and our implementation for checking Kaplansky's conjecture for infinite groups. The idea of the algorithm belongs to Dykema, Heister, and

Juschenko [4], however, they did not specify how to code and implement the algorithm. We designed our own algorithm and code for creating matrices to iterate over a special class of matrices. In Appendix A we provide our the algorithm and code that creates, in the form of matrices, what are known as non-degenerate and minimally realizable partitions. At the time of this report, due to server hard-drive failure and the school closure related to COVID-19, we have been unable to successfully begin checking ULIE groups on the USNA cluster.

In Section 6 we present a summary of our contributions to the field and avenues of future work in continuing the project. This includes finishing the code to successfully operate on a single machine and then updating the code to successfully operate on a cluster network.

Appendix A.1, A.2 and A.3 provides screenshots of all three of our programs.

In Appendix B we provide additional background knowledge on Gottschalk's conjecture, groups, generators, and word problems, focusing on providing an intuitive and visual understanding of Gottschalk's conjecture.

Lastly, Appendix C provides a glossary of short definitions and examples of basic theorems in abstract algebra.

2 Background To Theoretical Results

Before discussing Kaplansky's and Gottschalk's conjectures, we will briefly review the definition of groups along with their basic properties. Intuitively, groups are mathematical objects that help represent the symmetries of nature. They provide a way to describe how specific objects, like triangles or icosahedrals, behave.

Below is the formal definition of a group: [15]

Definition 2.1. A set G with a binary operation is called a group if the following holds:

- $(ab)c = a(bc)$ for all a, b, c in G
- there exists an **identity** in G, which we will notate as e.
- for each $a \in G$ there exists an inverse of a called a^{-1} such that $aa^{-1} = a^{-1}a = e$.

Well known examples of groups are the integers, symmetries of a triangle or square, and groups of matrices.

A group can be described by its generators and relations. The generators are the elements that build the group and the relations dictate the combinations of the generators that are equal to the group identity, e. In the case of the symmetry group of an equilateral triangle, the generators are the two possible ways to permute the vertices: a flip or rotation. Each of these movements maintain the triangle's original shape. An example of a relation would be that three rotations returns the same permutation.

If the vertices of a square were labeled, then all possible configurations of labelling would be the elements of a group resulting in what we call a dihedral group. In the following theorem we give the generators and relations that define and build the dihedral group for a regular n-gon.

Theorem 2.2. *The group D_n (dihedral group of order $2n$), $n \geq 3$, consists of all products of the two elements, rotation (r) and reflection (s), satisfying these relations:* [15]

- $r^n = 1$
- $s^2 = 1$
- $srsr = 1$

Another group to be used will be the *free group*. We give the formal definition below.

Definition 2.3. Let X be a set, G a group, and $i : X \to G$ a function. The pair (G, i) is called *free on* X if for every group H and function $f : X \to H$ there is a unique homomorhpism $\phi : G \to H$ such that $f = i\phi$. [19] We note that *free groups* on X are unique up to isomorphism, and, thus, will be denoted by $F(X)$.

When X is a finite set, and $X^{-1} = \{x^{-1} : x \in X\}$, then the free group on X can be seen as the collection of all words over $X \cup X^{-1}$ with concatenation of words being the multiplication operation. If $card(X) = n$, the free group on X is customary denoted by F_n.

Each group can be constructed by their generators and relations, together giving the group its structure. Namely, if G is a group with generators X and relations R, then G is isomorphic to the group $F(X)/N(R)$, where $N(R)$ is the normal closure of R in $F(X)$.

We notice that a *free group* has no relations. This allows one to construct a blank group without structure and then to add that structure later in the form of a *quotient group*.

Corollary 2.4. *Every group is isomorphic to the quotient group of a free group.* [18]

One can also plot each element of a group onto a Cayley Graph, with the edges representing the relations of the group, to obtain a graphical representation of the group. An example is shown in figure 1.

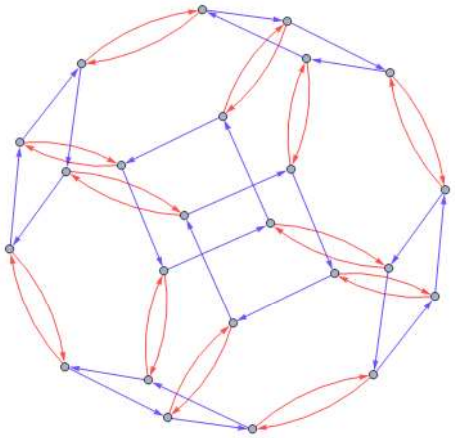


Figure 1: A graphical representation of a group. This is the Cayley graph for the symmetric group $\mathbb{S}_4$.

Definition 2.5. A group is commutative if for any elements $a, b \in G$, $ab = ba$.

Thus, for commutative groups the order of multiplication does not matter. For many other groups though this is not true. For instance, when multiplying matrices, the order of multiplication is critical.

Recall that a function $f : A \to B$ between sets A and B is called *injective* if $f(a_1) = f(a_2)$ $a_1, a_2 \in A$, implies that $a_1 = a_2$. A function $f : A \to B$ between sets A and B is called

surjective if for every $b \in B$ there is an $a \in A$ such that $f(a) = b$. A *bijection* is a function that is both injective and surjective.

An important group we will reference is the *symmetric group*.

Definition 2.6. For a finite set X, we denote by S_X the group of all bijections of X onto itself — in other words the permutations of X. This group, with the composition of functions as the operation, is called the **symmetric group** [16].

A theorem linking dihedral groups to symmetric groups follows:

Theorem 2.7. *The dihedral group, D_n, is a subgroup of S_n of order $2n$.* [15]

Definition 2.8. Two groups are isomorphic if there exists a one-to-one and onto map $\phi : G \to H$ such that the group operation is preserved; that is, [15]

$$\phi(a \cdot b) = \phi(a) \circ \phi(b)$$

Recall that a group is called *cyclic* if it is generated by a single element.

Theorem 2.9. *All cyclic groups of infinite order are isomorphic to $\mathbb{Z}$* [15]

Proof:
Let G be a cyclic group with infinite order and suppose that a is a generator of G. Define a map $\phi : \mathbb{Z} \to G$ by $\phi : n \to a^n$. Then,

$$\phi(m + n) = a^{m+n} = a^m a^n = \phi(m)\phi(n)$$

To show that ϕ is injective (thus one-to-one), suppose that m and n are two elements in $\mathbb{Z}$, where $m \neq n$. We can assume that $m > n$. We must show that $a^m \neq a^n$. Let us suppose to the contrary; that is, $a^m = a^n$. In this case $a^{m-n} = e$, where $m - n > 0$, which contradicts the fact that a has infinite order. Our map is onto since any element in G can be written as a^n for some integer n and $\phi(n) = a^n$. Thus the requirements for an isomorphism are satisfied. □

A property of groups is that any finite group is isomorphic to a subgroup of a symmetric group. Often times for a property we want to check on all groups, we can simply check the property against the symmetric group.

Theorem 2.10 (Cayley's Theorem)**.** *Every finite group is isomorphic to a subgroup of a symmetric group.*

Proof:
Let G be a group. We must find a group of permutations $\hat{G}$ that is isomorphic to G. For any $g \in G$, define a function $\lambda_g : G \to G$ by $\lambda_g(a) = ga$. We claim that λ_g is a permutation of G. To show that λ_g is one-to-one, suppose that $\lambda_g(a) = \lambda_g(b)$. Then

$$ga = \lambda_g(a) = \lambda_g(b) = gb$$

Hence, $a = b$. To show that λ_g is onto, we must prove that for each $a \in G$, there is a b such that $\lambda_g(b) = a$. Let $b = g^{-1}a$. Now we can define our group $\hat{G}$. Let

$$\hat{G} = \{\lambda_g : g \in G\}$$

We must show that $\hat{G}$ is a group under composition of functions and find an isomorphism between G and $\hat{G}$. We have closure under composition of functions since

$$(\lambda_g \circ \lambda_h)(a) = \lambda_g(ha) = gha = \lambda_{gh}(a)$$

also,

$$\lambda_e(a) = ea = a$$

and

$$(\lambda_g^{-1} \circ \lambda_g)(a) = \lambda_{g^{-1}}(ga) = g^{-1}ga = a = \lambda_e(a)$$

We can define an isomorphism from G to $\hat{G}$ by $\phi : g \to \lambda_g$. The group operation is preserved since

$$\phi(gh) = \lambda_{gh} = \lambda_g\lambda_h = \phi(g)\phi(h)$$

It is also one-to-one, because if $\phi(g)(a) = \phi(h)(a)$, then

$$ga = \lambda_g a = \lambda_h a = ha$$

Hence, $g = h$. That ϕ is onto follows from the fact that $\phi(g) = \lambda_g$ for any $\lambda_g \in \hat{G}$. □ [15]

Another mathematical object used in Kaplansky's conjecture is a field. A field is like a group but instead of one operation, it has two, usually multiplication and addiction. The set of real numbers $\mathbb{R}$ and the set of complex numbers $\mathbb{C}$ are examples of infinite fields. We could also have finite fields defined on a set such as $0, 1$ or $0, 1, 2$. The formal definition follows.

Definition 2.11. A field F is a set endowed with both addition and multiplication for which the following properties hold:

- for $a, b \in F$ $a + b = b + a$
- for $a, b, c \in F$ $(a + b) + c = a + (b + c)$
- the identity exists for F exists such that $a + 0 = a$
- the additive inverse of each element in F exists such that $a - a = 0$
- multiplication is commutative, $ab = ba$
- multiplication is associative, $(ab)c = a(bc)$
- the operations are distributive such that $(a + b)c = ac + bc$

- there exists an identity e such that $ae = ea = a$
- there exists a multiplicative inverse such that $aa^{-1} = e$

Definition 2.12. Let K be a commutative ring and let G be a finite group. The *group algebra (or ring)* of G over K is defined to be the ring of formal sums:

$$\sum_{g \in G} \{r_g g \quad | \quad r_g \in K\}$$

with the addition and multiplication inherited from G [24].

We now state Kaplansky's conjecture:

For any group G and commutative field K, the group algebra $K[G]$ is directly finite. That is, $ab = 1$ in $K[G]$ implies $ba = 1$ [3].

To illustrate again what a group algebra is, we consider our group $G = \mathbb{Z}$, where $\mathbb{Z}$ is the set of integers. We define our field K as the set of real numbers $\mathbb{R}$. Then $K(G)$ is the set of power series $\sum_{n=-\infty}^{\infty} c_n z^n$. This sum can be thought of as a set of polynomials. We want to be able to check if one polynomial, a, times another polynomial, b, satisfies Kaplansky's Conjecture such that $ab = 1$ implies $ba = 1$. In our previous example since Z and K are both commutative, then $ab = ba$, and thus trivially if $ab = 1$, then $ba = 1$. Kaplanksy in [9] already proved that for fields of characteristic 0. The focus of our work is on finite fields.

In order to check Kaplansky's conjecture we had to build a program to analyze groups and group algebras. This problem naturally leads to a class of problems in computational group theory called "decision problems." The most significant problem of this class is the "word problem." The word problem addresses one of the most important aspects of computational group theory: are two elements, or words, of a group equivalent? This along with the question of how to convert group elements into "words" will be discussed below. We will choose to primarily use free groups to build our groups.

The word problem for a group arises from the class of problems called *decision problems*. Decision problems are a subset of both decidable and undecidable problems in computational group theory that arise from analyzing large *words* or sequences of letters within a group, for instance *abghegheg*. The goal is to find an algorithm that can cancel out parts of the sequence based on certain properties or rules of the group. In this case, if our group had a special rule that *gh* always cancels out to 1, then our word *abghegheg* can be reduced to *abeeg*. One can apply the unique rules different groups have to words within that group in order to encode information or check certain group properties.

For a formal definition of a *word* problem, we begin with an *alphabet* :

$$A = \{a, b, c, .., a^{-1}, b^{-1}, c^{-1}...\}$$

Consider all *words* over this alphabet, for example $aabbgh$ or $aba^{-1}a^{-1}$. We will denote this collection by A^*. We use the convention that if a letter, say, $m \in A$ and its inverse $m^{-1} \in A$ appear next to each other, like mm^{-1} or $m^{-1}m$, in a word $w \in A^*$, then they are removed from the word. Suppose that we also have a collection of *cancellation* words $C = \{c_1, \ldots, c_n\}$. Due to these words being the equivalent to 1 or e, we are allowed to insert them into any arbitrary word w.

One of the fundamental questions in computational group theory is the algorithmic problem of deciding whether two words from A^* can be reduced to equal each other. Surprisingly, not every collection of cancellation words admits such an algorithm [1]. In this case, we say that the word problem is *unsolvable* or *undecidable*.

We began investigating Kaplansky's Conjecture by first designing a word solving program. Our rewriting system had to be both *confluent* and have a *canonical* form. To have a canonical form means words are reducible a unique "simplest" form. A system is confluent if when a word x reduces to two different words, w and y, then there is a third word, z, that each of them further reduce to. We describe below the various properties a canonical and confluent rewriting system must have.

Definition 2.13. A rewriting system on A^* is a set of ordered pairs (w_1, w_2), with w_1, $w_2 \in A^*$ [24].

Let G be a finitely represented group and X^* be the set of all possible words over a set X. The set X in this case represents the generators for our group G. If the group generators are a and b, then X^* would be the infinite combinations of infinite a's and b's. For instance, both $aabbbb$ and $aaaabbbbbaaa$ would be words contained in X^*.

In order for a rewriting system to solvable, it must be *terminating* and *confluent*, see Definition 2.15 and 2.16 [24].

One approach to the word problem is to attempt to choose for each element g of G one word w from among all the words in X^* which define g. Thus, one could reduce a family of words to a single word. Such a choice is called the *canonical form* of w.

Definition 2.14. We say that a word $w \in A^*$ is *irreducible* if there is no word $v \in A^*$ such that $w \to v$, where $w \to v$ symbolizes a reduction from w to v using A^*'s cancellation words [24].

A rewriting system is *terminating* if for every reduced form of the word w_i, there does not exist a further reduction, w_{i+1}. We give the formal definition below:

Definition 2.15. A system R is called terminating if there is no infinite words, w_i, with $i > 0$ such that $w_i \to w_{i+1}$ for all i. This implies that for any $w \in A^*$ there exists an irreducible $v \in A^*$ with $w \to v$

We now define confluence [24].

Definition 2.16. We say that a system R is said to be *confluent* if whenever $u, v_1, v_2 \in A^*$ with $u \to v_1$ and $u \to v_2$, there exists $w \in A^*$ with $v_1 \to w$ and $v_2 \to w$ [24].

Thus, if a rewriting system is both terminating and confluent, then by definition if we reduce a word w two different ways to s_i and s'_i, since s_{i+1} and s'_{i+1} do not exist (due to being complete), and since the system is still confluent, then $s_i = s'_i$. This means that every word in the group can be reduced to a unique "normal" form. To be complete, the process of reduction does not necessarily need to be unique, thus one word could be reduced two different ways, however, the end result must converge to the same reduced word [12]. Because the process of reduction does not need to be unique, the rules for rewriting can be applied in any order.

We outline how a word can be reduced into two different forms in a rewriting system.

Example 2.17. Let $X = a, b$ and let R consist of the rules $a^2 \to e$, $b^{10} \to e$, and $ba \to ab^4$, where e is the identity. This is a rewriting system on X^* with respect to the basic ordering of a being longer than b. Let us rewrite the word baa in two ways. The first way is very easy:

$$b\underline{aa} \to b. \tag{1}$$

The second way takes a little longer:

$$\underline{ba}a \to abbb\underline{ba} \to abb\underline{ba}bbb \to ab\underline{ba}bbbbbbb \to ababb\underline{bbbbbbbbb} \to a\underline{ba}bb \to \underline{aa}bbbbbb \to bbbbbb. \tag{2}$$

The result is b in the first case and b^6 in the second [12].

Although the relations in a group are used as the basis for the rewriting system for that group, they alone are not sufficient. The rewriting system must be made confluent. The most well-known process for this is the Knuth-Bendix algorithm [12]. The basic process is to first reduce each relation from the left, then reduce from the right. If the two words are equal, in that the relation reduces to the same word regardless of which rules were used, we say that relation is confluent and a part of our rewriting system. If the words do not equal, we set them equal to each other and add it to our system. We continue checking relations, and adding rules to our system until there are no further confluent rules to add. At this point the process terminates and the relations have been turned into a confluent rewriting system.

Example 2.18. Let $Y = \{a, b\}$ and let $\mathcal{T}$ consist of the rules

$$abab \to e, \quad baba \to e.$$

the overlap in our two relations gives rise to the following six words. To show that $\mathcal{T}$ is confluent, we must check the confluence of these overlap words:

$$\begin{aligned} ababa, &\quad babab, \\ ababab, &\quad bababa, \\ abababa, &\quad bababab. \end{aligned}$$

For example. we have

$$a\underline{baba} \to a, \quad \underline{abab}a \to a,$$

and

$$ba\underline{baba} \to ba, \quad \underline{baba}ba \to ba.$$

Confluence holds at the other four words as well, and thus $\mathcal{T}$ is confluent.

We now give an example of using the Knuth-Bendix process to build a rewriting system for a group using just its relations. We do this by combining overlapping relations, then solving them from the left and right to determine if they reduce to different words. If they do reduce to different words, we then add them as an equivalent relation and new rule [24]

Example 2.19. Let $G =< x, y | x^3 = 1, y^2 = 1, (xy)^2 = 1 >$ be the dihedral group of order 6. Then $A = \{x, x^{-1}, y, y^{-1}\}$. We can start our rewriting system as:

$$R_0 = \{(xx^{-1}, e), (x^{-1}x, e), (yy^{-1}, e), (y^{-1}y, e), (xx, x^{-1}), (y^{-1}, y), (yx^{-1}, xy)\}$$

We must now find any overlapping or possibly redundant pairs. One pair that has an overlap is $(x^{-1}x, e)$ and (xx, x^{-1}), with x overlapping as the first element in each pair. We then combine the first two pairs, overlapping the x, giving us the word $x^{-1}xx$. Reducing this word from the left and right gives us two results, x and $x^{-1}x^{-1}$, thus we can add this rule, $(x^{-1}x^{-1}, x)$ to R. Sampling and testing over all overlapping pairs by reducing them left and then right, and equating the result, gives us the complete set of rules:

$$\begin{aligned} R = \{&(xx^{-1}, e), (x^{-1}x, e), (yy^{-1}, e), (y^{-1}y, e), (xx, x^{-1}), (x^{-1}x^{-1}, x), \\ &(y^{-1}, y), (yy, e), (yx^{-1}, xy), (yx^{-1}, x^{-1}y)\} \end{aligned}$$

Word solving can be handled internally by *SAGE*, however, for any system built in *Python* we must manually build our system and define how to rewrite words into their canonical form.

3 Theoretical Results

In this section we will present a proof that Gottschalk's implies Kaplansky's conjecture for finite and infinite groups and give a proof of Kaplansky's conjecture for finite groups.

Definition 3.1. Let G be a group and X be a set. Then we define X^G as the space of functions $f : G \to X$.

Conjecture 3.2 (Gottschalk's conjecture)**.** *Let G be a countable group and X a finite set. Consider the compact metrizable space X^G of X-valued functions on G equipped with the product topology. Let $f : X^G \to X^G$ be a continuous map that commutes with the natural right G-action, see Definiton 3.6. Then if f is injective, it is surjective as well.* [3]

We note that if G is a finite group, then X^G is a finite set. Thus, any function $f : X^G \to X^G$ that is injective is automatically surjective, whereby establishing the conjecture for finite groups.

Definition 3.3. Let G be a finite group, F be a field, f_1, f_2 be elements of F^G and $h \in G$. Define a convolution between f_1, f_2 as

$$(f_1 * f_2)(h) = \sum_{g \in G} f_1(g) f_2(g^{-1}h).$$

Theorem 3.4. *Let F be a field and G a finite group. Then the convolution between $f_1, f_2 \in F^G$ is commutative if the group is commutative.*

Proof:

$$\begin{aligned}
(f_1 * f_2)(h) &= \sum_{g \in G} f_1(g) f_2(g^{-1}h) \\
(f_2 * f_1)(h) &= \sum_{g \in G} f_1(g^{-1}h) f_2(g) \quad \text{due to multiplication in finite fields being commutative} \\
&= \sum_{r \in G} f_1(r) f_2(r^{-1}h) \quad \text{only if } r^{-1}h = h^{-1}gh = g, \text{ and } r = g^{-1}h \\
&= (f_1 * f_2)(h). \qquad (3)
\end{aligned}$$

□

Definition 3.5. Let G be a group. We define the *Dirac function* on G as:

$$\delta_e(g) = \begin{cases} 1 & \text{if g} = e \\ 0 & \text{otherwise} \end{cases} \qquad (4)$$

Note that $f * \delta_e = f = \delta_e * f$, which can be seen as follows:

$$\begin{aligned}(f * \delta_e)(h) &= \sum_{g \in G} f(g)\delta_e(g^{-1}h) \\ &= f(h)\delta_e(h^{-1}h) \\ &= f(h)\delta_e(e) \\ &= f(h). \end{aligned} \tag{5}$$

and

$$\begin{aligned}(\delta_e * f)(h) &= \sum_{g \in G} \delta_e(g)f(g^{-1}h) \\ &= \delta_e(e)f(e^{-1}h) \\ &= \delta_e(e)f(h) \\ &= f(h). \end{aligned} \tag{6}$$

The Dirac function assumes the role of the group identity, within the space of functions F^G endowed with convolution as the group operation.

Definition 3.6. Let X be a set and G be a group. A function $\alpha : G \times X \to X$ is called a *left group action* on X if it satisfies two axioms:

$$1. \quad \alpha(e, x) = x, \text{ for all } x \in X \tag{7}$$

$$2. \quad \text{for all } g_1, g_2 \in G, x \in X, \quad \alpha(g_1, \alpha(g_2, x)) = \alpha(g_1 g_2, x) \tag{8}$$

The action of g on x — $\alpha(g, x)$ — will be denoted by $g \cdot x$.

Definition 3.7. For g ∈ G, we define $L_g : F^G \to F^G$ and $R_g : F^G \to F^G$ as follows: $L_g(f)(h) = f(g^{-1}h)$ and $R_g(f)(h) = f(hg)$, where $h \in G$.

Theorem 3.8. *Let G be a finite group and F be a field. Then,*
(1) $\alpha : G \times F^G \to F^G$, defined as $\alpha(g, f) = L_g(f)$, is a left group action on F^G.
(2) Similarly, $R_g : F^G \to F^G$ defines a right group action of G on F^G.

Proof: (1)

$$\begin{aligned} &1. \quad L_e(f)(h) = f(e^{-1}h) = f(h) \\ &2. \quad L_{g_1,g_2}(h) = (L_{g_1}(f) \circ L_{g_2}(f))(h) \\ &\qquad = f(g_2^{-1}(g_1^{-1}h)). \end{aligned} \tag{9}$$

Let $f(g_2^{-1}\bar{h}) = L_{g_2}(f)(\bar{h})$, $\bar{h} = g_1^{-1}h$ and $w = L_{g_2}(f)$, then

$$\begin{aligned} f(g_2^{-1}\bar{h}) &= L_{g_2}(f)(\bar{h}) \\ &= w(\bar{h}) \\ &= w(g_1^{-1}h) \\ &= L_{g_1}w(h) \\ &= L_{g_1}L_{g_2}(f)(h). \end{aligned}$$

The proof for (2) is similar to (1). □

We are now ready to prove Kaplansky's Conjecture for finite groups.

Theorem 3.9 (Kaplansky's Conjecture, [3])**.** *Let G be a finite group and F be a field. For $a, b \in F(G)$, if $ab = 1$ then $ba = 1$.*

Proof:

In the proof below we are going to view $a, b \in f(g)$, with convolution being the multiplication operation. We begin by noticing that any element f of the group algebra $F(G)$ induces a linear map $A_f : F^G \to F^G$ defined as $A_f(q) = f * q$, where $q \in F^G$.

Notice that $(f_1 * f_2) * q$ is associative. Indeed,

$$\begin{aligned} (f_1 * f_2) * q &= \sum_{g\in G}(f_1 * f_2)(g)q(g^{-1}h) \\ &= \sum_{g\in G}\sum_{r\in G} f_1(r)\cdot f_2(r^{-1}g)\cdot q(g^{-1}h) \\ f_1 * (f_2 * q) &= \sum_{g\in G} f_1(g)\cdot(f_2 * q)(g^{-1}h) \\ &= \sum_{g\in G}\sum_{r\in G} f_1(g)\cdot f_2(r)\cdot q(r^{-1}g^{-1}h) \\ &= \sum_{g\in G}\sum_{w\in G} f_1(g)\cdot f_2(g^{-1}w)\cdot q(w^{-1}h) \end{aligned}$$

with $w = r^{-1}g^{-1}$ and $r = g^{-1}w$. Now fix $f \in F^G$ and $q \in F^G$ and $A_f q(h) = (f * q)(h)$. Thus,

$$A_{f_1 * f_2}(q) = A_{f_1}\cdot A_{f_2}\cdot q. \tag{10}$$

Let $a, b \in F(G)$, or functions in F^G, and assume $a * b = \delta_e$ — implying $A_a A_b = I$. Then $det(A_a A_b) = det(I)$ and $det(A_a A_b) = det(A_a) * det(A_b)$. Thus, $det(A_a) * det(A_b) = 1$. Indeed, if this holds then both $det(A_a)$ and $det(A_b)$ are nonzero. Thus, A_a is the inverse of A_b and thus A_a and A_b commute. □

In order to prove Gottshalk's conjecture implies Kaplansky's conjecture for infinite groups we first need some results about topological spaces and the Cantor space.

Definition 3.10. A topological space is a set X together with a family of subsets T, which we call *open*, that satisfies the three conditions:

- Both X and the empty set $\emptyset$ are in T
- The union of an arbitrary number of sets in T is also in T
- The intersection of a finite number of sets in T is also in T

Definition 3.11. A subset A of a topological space X is called *dense* (in X) if every point $x \in X$ either belongs to A or is a limit point of A.

Let F be a finite field endowed with the discrete topology. Consider the space of functions F^G viewed as the product space with the product topology. We notice that the collection of functions with finite supports is dense in F^G. In particular, the group algebra $F(G)$, viewed as a subset of F^G, is dense in F^G. In the proof below, given two continuous functions f_a and f_b from F^G to F, in order to establish that $f_a \equiv f_b$ it suffices to check this identity on the dense subset $F(G)$.

Theorem 3.12 (Gottschalk's Conjecture implies Kaplansky's Conjecture for Infinite Groups, [3])**.** *Let G be a group and F be a finite field. If Gottschalk's conjecture holds true for G, then Kaplanksy's conjecture holds true for G. Namely, if every continuous injective function $A : F^G \to F^G$ commuting with the shift on F^G is automatically surjective, then for $a, b \in F(G)$, if $ab = 1$ then $ba = 1$.*

Proof:

Our proof will function very similar to the proof of Theorem 3.9. For each $f \in F(G)$, we define $A_f : F^G \to F^G$ by $A_f(q) = f * q$, where $q \in F^G$. Notice that functions from $F(G)$ have finite supports, so the linear operator A_f is well-defined. We observe that A_f is a continuous function on F^G when equipped with the product topology.

We claim that the linear operator A_f on F^G commutes with the right G-action, creating the following commutative diagram,

$$\begin{array}{ccc} F^G & \xrightarrow{A_f} & F^G \\ \downarrow{\scriptstyle R_g} & & \downarrow{\scriptstyle R_g} \\ F^G & \xrightarrow{A_f} & F^G \end{array} \tag{11}$$

We want to show $\left(R_g A_f(q)\right)(h) = \left(A_f R_g(q)\right)(h)$, with $g \in G$. Then,

$$\begin{aligned} R_g A_f(q)(h) = A_f(q)(hg) &= (f * q)(hg) \\ &= \sum_{r \in G} f(r) q(r^{-1}hg). \end{aligned}$$

For the right side,

$$\begin{aligned} A_f(R_g(q))(h) &= (f * R_g(q))(h) \\ &= \sum_{r \in G} f(r) R_g q(r^{-1}h) \\ &= \sum_{r \in G} f(r) q(r^{-1}hg). \end{aligned}$$

It follows from the proof of Theorem 3.9 that for all $f_1, f_2 \in F(G)$,

$$A_{f_1 * f_2}(q) = A_{f_1} \cdot A_{f_2}.$$

Let $a, b \in F(G)$, or functions in F^G, and assume $a * b = \delta_e$, implying $A_a A_b = I$. Then A_b must be injective. Indeed, if there was a $q \neq 0$ such that $A_b q = 0$, then $A_a A_b q = Iq$, which is $0 = q$, which is a contradiction. By Gottschalk's conjecture A_b is automatically surjective. Thus, A_b is an invertible operator, thus A_a is the inverse of A_b, and thus A_a and A_b commute. $\square$

4 Computational Approach: Finite Groups

In the remaining part of this paper we present several algorithms that if given enough time could in principle check the Kaplansky's conjecture for both finite groups and infinite groups. Since every finite group is a subgroup of a symmetric group, in order to check the conjecture for the class of finite groups, it suffices to check it for the symmetric group. This result comes from an important theorem in Group Theory that is reviewed in Section 2 by Theorem 2.10.

We first began to check Kaplansky's conjecture in *SAGE* using built in functions to define symmetric groups and to multiply and check group algebra elements. This code however ran too slow to effectively iterate over larger symmetric groups, thus we continued our work by shifting our mathematical approach whereby instead of constructing group algebra elements as abstract objects we converted them into large matrices to be easily checked by a computer. Namely, we switched from an abstract definition of symmetric groups to the matrix representation of symmetric groups using permutation matrices. Then we represented our group algebra elements as matrices, explained below. This allowed us to convert abstract multiplication in the group algebra to regular matrix multiplication. Thus, checking Kaplansky's conjecture reduced to simple matrix multiplication. Reference Appendix A.1 for our first program in *SAGE* for finite groups and reference Appendix A.2 for our second program for checking finite groups.

For any symmetric group, we can define the generators of our group via pair of cycles:

- the first as $a = (12)$
- the second is $b = (1...n)$

From these generators, we can combine them together to create the entire group. Additionally, each of these cycles can then be represented as a matrix. In the following example, we explain how it works in the case of the symmetric group on three elements.

Example 4.1. The group S_3 is generated by two cycles (123) and $(1, 2)$. The permutation matrix representation for these cycles is

$$(1,2) = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \text{ and } (1,2,3) = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

We can then multiply these matrices like multiplying cycles together

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

Thus, we have created a new permutation

$$(2,3) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

We can continue this until obtaining all six cycles in S_3.

Our next goal was to find a matrix representation for the group algebra elements. Consider a finite group $G = \{g_1, \ldots, g_n\}$. Define the *group matrix* $M(G)$ [13] as follows

$$\begin{pmatrix} g_1^{-1}g_1 & g_1^{-1}g_2 & g_1^{-1}g_3 & \cdots & g_1^{-1}g_n \\ g_2^{-1}g_1 & g_2^{-1}g_2 & g_2^{-1}g_3 & \cdots & g_2^{-1}g_n \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ g_n^{-1}g_1 & g_n^{-1}g_2 & g_n^{-1}g_3 & \cdots & g_n^{-1}g_n \end{pmatrix}$$

Consider a field F. Then the group algebra elements can be viewed as formal sums $\sum_{k=1}^{n} \alpha_k g_k$, $\alpha_k \in F$. We represent each group algebra element $w = \sum_{k=1}^{n} \alpha_k g_k$ as the matrix:

$$M(w) = \begin{pmatrix} \alpha_{g_1^{-1}g_1} & \alpha_{g_1^{-1}g_2} & \alpha_{g_1^{-1}g_3} & \cdots & \alpha_{g_1^{-1}g_n} \\ \alpha_{g_2^{-1}g_1} & \alpha_{g_2^{-1}g_2} & \alpha_{g_2^{-1}g_3} & \cdots & \alpha_{g_2^{-1}g_n} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \alpha_{g_n^{-1}g_1} & \alpha_{g_n^{-1}g_2} & \alpha_{g_n^{-1}g_3} & \cdots & \alpha_{g_n^{-1}g_n} \end{pmatrix}$$

The following result allows us to replace the group algebra operations with matrix addition and multiplication.

Theorem 4.2. *$M : F(G) \to Mat_{n\times n}(F)$ is a group algebra isomorphism on its image* [13].

Proof:

By its construction, the map M is additive, surjective and injective on its image. Thus, we must only prove that M preserves group multiplication to prove isomorphism. Consider the basis in the space of functions F^G: $\{\delta_{g_1^{-1}}, \delta_{g_2^{-1}}, \cdots, \delta_{g_2^{-1}}\}$.

We write our group G as $\{g_1^{-1}, g_2^{-1}, \cdots, g_n^{-1}\}$. Given $f \in F(G)$, viewed as a function in F^G, we define a linear operator $A_f : F^G \to F^G$ as follows: $A_f q(h) = (f * q)(h)$ with $h \in G$ and $q \in F^G$.

We notice that:

$$A_f(\delta_q)(h) = (f * \delta_q)(h) = \sum_{g \in G} f(g)\delta_q(g^{-1}h) = f(hq^{-1}), \text{ for } g = hq^{-1}$$

So the inner product,

$$< A_f\delta_q, \delta_g >= \sum_{h \in G} A_f\delta_q(h)(\delta_g)(h) = \sum_{h \in G} f(hq^{-1})\delta_g(h) = f(gq^{-1})$$

Thus, the ith and jth entry of the matrix of A_f is $< A_f\delta_{g_i^{-1}}, \delta_{g_j^{-1}} >= f(g_j^{-1}g_i)$.

Thus $M(f)$ is precisely the matrix representation of the operator A_f in the basis $\{\delta_{g_1^{-1}}, \delta_{g_2^{-1}}, \cdots, \delta_{g_2^{-1}}\}$. Since $M(f)$ is the matrix of A_f in the basis $\{\delta_{g_1}, \cdots, \delta_{g_n}\}$ and $A_{f_1 * f_2} = A_{f_1}A_{f_2}$ (see the proof of 3.9), we obtain that $M(f_1 * f_2) = M(f_1) * M(f_2)$. □

We will briefly go over an example of what these matrices look like:

Example 4.3. Let $G = Sym(3)$, $F = \{0, 1\}$ and $w = a + ab \in F(G)$. The group matrix, $M(G)$, is constructed by multiplying $G = \{1, a, a^{-1}, b, ab, a^{-1}b\}$ with $G^{-1} = \{1, a^{-1}, a, b, ab, ba\}$. Thus,

$$M(G) = \begin{pmatrix} 1 & a & a^{-1} & b & ab & a^{-1}b \\ a^{-1} & 1 & a & a^{-1}b & b & ab \\ a & a^{-1} & 1 & ab & a^{-1}b & b \\ b & ba & ba^{-1} & 1 & bab & ba^{-1}b \\ ab & aba & aba^{-1} & a & 1 & aba^{-1}b \\ ba & ba^{-1} & b & bab & ba^{-1}b & 1 \end{pmatrix}$$

Next, to find a matrix representation of w we look at w alongside $G = \{1, a, a^{-1}, b, ab, a^{-1}b\}$. This expansion looks like: $w = 0 \cdot 1 + 1 \cdot a + 0 \cdot a^{-1} + 0 \cdot b + 1 \cdot ab + 0 \cdot a^{-1}b$. We next substitute each g_i element for w's corresponding coefficient (such as $a^{-1} = a^2$, $a = ba^{-1}b$, etc...)

$$\begin{pmatrix} 1 & 1 & a^{-1} & b & 1 & a^{-1}b \\ a^{-1} & 1 & 1 & a^{-1}b & b & 1 \\ 1 & a^{-1} & 1 & 1 & a^{-1}b & b \\ b & ba & 1 & 1 & bab & 1 \\ 1 & aba & aba^{-1} & 1 & 1 & aba^{-1}b \\ ba & 1 & b & bab & 1 & 1 \end{pmatrix}$$

With the final result of w represented as:

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

Our code was further optimized by checking only the upper triangular (a, b) pairs, as well as, by parallelizing the code to run on multiple CPUs.

The original code using *SAGE's* built-in functions and the code using the matrix representation are presented in Appendix A.1 and Appendix A.2, respectively.

5 Computational Approach: Infinite Groups

In this section we are going to present the main idea behind the algorithm for checking Kaplansky's conjecture for infinite groups proposed by [4].

Recall that Kaplansky's conjecture states that for any group G and commutative field K, the group algebra $K(G)$ is directly finite, that is, $ab = 1$ in $K(G)$ implies $ba = 1$.

In what follows, we will assume that our field F is $\{0, 1\}$. Fix a group G. Let $a, b \in \mathbb{F}_2(G)$ be elements such that $ab = 1$. Let,

$$a = a_0 + a_1 + \cdots + a_{m-1} \text{ and } b = b_0 + b_1 + \cdots + b_{n-1},$$

where $\{a_i\}$ are distinct elements of the group G, and $\{b_j\}$ are distinct elements of G. We can apply ab element by element to obtain:

$$\sum_i \sum_j a_i b_j = ab = 1.$$

Due to our field size of $\{0, 1\}$, when we multiply a and b, the result must have an odd number of a_i, b_j pairs in order to equal 1. We obtain that $a_i b_j = 1$ for at least one pair (i, j). Without loss of generality, we can assume $i = j = 0$, thus we get the property that $a_0 b_0 = 1$. Thus, $a_0 = b_0^{-1}$, which implies that a_0 and b_0 commute and $a_0^{-1} b_0^{-1} = 1$.

Set

$$\bar{a} = a_0^{-1} a = 1 + a_0^{-1} a_1 + a_0^{-1} a_2 + \cdots$$

and

$$\bar{b} = b b_0^{-1} = 1 + b_1 b_0^{-1} + b_2 b_0^{-1} + \cdots .$$

Then

$$\bar{a}\bar{b} = a_0^{-1} a b b_0^{-1} = a_0^{-1} b_0^{-1} = 1$$

Relabeling $\bar{a}$ and $\bar{b}$ element by element, for instance $a_0^{-1} a_1$ to a_1, gives us our original a and b:

$$a = a_0 + a_1 + \cdots + a_{m-1}$$
$$b = b_0 + b_1 + \cdots + b_{n-1}$$

such that

$$\sum_i \sum_j a_i b_j = 1 = 1 + \sum_{i \neq 0} \sum_{j \neq 0} a_i b_j$$

with the assumption $a_0 b_0 = 1$.

Thus, we can obtain a partition $\pi : \{0, 1, \cdots, m-1\} \times \{0, 1, \cdots, n-1\}$ with one singleton $\{0, 0\}$ with all other sets containing two elements such that if $(i, j) \sim (i', j')$ then $a_i b_j = a_{i'} b_{j'}$.

Consider the finitely presented group:

$$\Gamma_\pi = \langle a_0, a_1, \cdots, a_{m-1}, b_0, b_1, \cdots, b_{n-1} | a_0 = b_0 = 1, (a_i b_j = a_{i'} b_{j'}) \text{ if } (i, j) \sim (i', j') \rangle$$

Notice that there is a natural homomorphism from $\Gamma_\pi \to G$. If the product $ab = (1 + a_1 + \cdots + a_{m-1})(1 + b_1 + \cdots + b_{n-1})$ is equal to 1 in $\mathbb{F}_2(G)$, then by definition of Γ_π, we have that $\hat{a}\hat{b} = 1 \in \mathbb{F}_2(\Gamma_\pi)$. On the other hand note that if $\hat{b}\hat{a} = 1$ in the group ring $\mathbb{F}_2(\Gamma_\pi)$, then $ba = 1$ in $\mathbb{F}_2(G)$.

Thus, if we can prove Kaplansky's Conjecture for all Γ_π, then we have covered all of G [4].

In the paper [4], Dykema, Heister, and Juschenko, proposed a mathematical algorithm that if implemented and given enough time can eventually find a counterexample to Kaplansky's Conjecture if such a counterexample exists. The algorithm utilizes the fact that one does not need to check every possible group against the conjecture but rather on the groups called ULIE groups. To check these ULIE groups we must shift our program from checking finite groups to checking finite generators and relations for infinite groups. We do this by creating the previously mentioned matrices.

Below we go over an abstract version of the matrices M we built. We let n and m be positive odd integers and $S = g_1, ..., g_n, h_1, ..., h_m$. Let $M = \{e_{i,j}\}$ be an $n \times m$ matrix whose entries take on values in $1, ..., \frac{nm}{2}$. We assume that each number in $1, 2, ..., \frac{nm}{2}$ appears twice among the entries of M. In this case, the matrix M represents our partition π that we then use to define the relations of Γ_π. For example,

$$M = \begin{bmatrix} x & 1 & 2 & 3 & 4 \\ 1 & 5 & 3 & 6 & 7 \\ 4 & 7 & 5 & 2 & 6 \end{bmatrix}$$

We denote by Γ_M the group generated by S and the defining relations:

$$R = \{g_i h_j = g_{i'} h_{j'} \; if \; e_{i,j} = e_{i',j'} \;\; 1 \leq i \leq n \;\; 1 \leq j \leq m\}$$

We then create an algorithm to effectively iterate over the groups in this class and test Kaplansky's Conjecture. It has already been shown that the conjecture holds for groups over the field $K = \mathbb{F}_2$ of two elements, for ranks $(3, n), n \leq 11$ and (5,5) [4].

In [4] they were able to extrapolate the "matrix-building" from earlier to represent any group with any type of relations as a matrix. For instance the cyclic group of order seven, $\mathbb{Z}_7$, can be represented by:

$$M = \begin{bmatrix} x & 1 & 2 & 3 & 4 \\ 1 & 5 & 3 & 6 & 7 \\ 4 & 7 & 5 & 2 & 6 \end{bmatrix}$$

In this case, we look at the top row and label each element as b_1, b_2, b_3, b_4 and the leftmost column as a_1, a_2 (thus a_1 " = " 1 and a_2 " = " 4). In this case because at a_1 and b_1 the matrix has "1," then $a_1 = b_1$. The reader will notice that there are only "pairs" in the matrix. Each creating an equivalence between the points. We can also see that $b_4 = a_2$. Letting $b_1 = a_1 = x$, we find that some of the relations for our group are $a_1 = b_1$, $b_1a_1 = b_2a_2$, $b_3a_2 = b_2$, $b_4 = a_2$... By taking these relations and dividing them into a created free group with 6 generators (from a_1, a_2, b_1...), we can create a quotient group that is isomorphic to $\mathbb{Z}_7$.

To check all infinite groups, we must check every possible matrix. Although this is something a computer can do very well, it is unrealistic and inefficient to simply check "every" possible matrix.

However, in [4] they were able to isolate a class of groups that are non-sofic and which would include all counterexamples, should any exist. These are called the Universal Left Invertible Element groups (ULIE). There are two main properties that all ULIE groups share: they are *non-degenerate* and *minimally realizable*.

Independently we had to build an algorithm for iterating through all the possible matrices that satisfy these constraints. In Appendix A we provide our own algorithm and code that creates these non-degenerate and minimally realizable partitions in the form of matrices.

We give the definition of each below:

Definition 5.1. Let $m, n \in \{2, 3, ...\}$ and let π be a partition of the set $\{0, ..., m-1\} \times \{0, ..., n-1\}$. We say that π is *degenerate* if in the group

$$\Gamma_\pi =< a_0, a_1, ..., a_{m-1}, b_0, b_1, ..., b_{n-1} | a_0 = b_0 = 1, (a_i b_j = a'_i b'_j) \text{ given } (i, j)\ (i', j') >$$

the group elements $a_0, a_1, ..., a_{m-1}$ are not distinct or the group elements $b_0, b_1, ..., b_{n-1}$ are not distinct. [4]

This constraint of our partition π, in practicality our matrix, allows us to restrict our program to building matrices that are almost like a "sodoku" puzzle, in that each row and column cannot repeat a generator.

Definition 5.2. Let K be a field and let $r_0, ..., r_{m-1}, s_0, ..., s_{n-1} \in K\backslash\{0\}$. A partition π of the set $\{0, ..., m-1\} \times \{0, ..., n-1\}$ is *minimally realizable with* $r_0, ..., r_{m-1}, s_0, ..., s_{n-1}$ if it is the minimal amount of all the partitions that are realizable with $r_0, ..., r_{m-1}, s_0, ..., s_{n-1}$. We say π is *minimally realizable over* L if it is minimally realizable with some choice of $r_0, ..., r_{m-1}, s_0, ..., s_{n-1} \in K\backslash\{0\}$, and π is simply minimally realizable if it is minimally realizable over some field K. [4]

By requiring our partition π (or once again in our case our matrix) to be minimally realizable, it implies that any partition can have at most one singleton and in fact must be "paired" off — similar to the $\mathbb{Z}_7$ example. So when programming our matrix builder, we simply needed to constrain our program to only build matrices that repeats each generator only once in a given column or row, and also produces only pairs in the matrix except for the singleton, or "x" value, at the $(0, 0)$ location.

Both of these constraints severely limited the number of matrices we need to check.

To build our group we must define its generators and relations. We can define our generators by the rows and columns of the matrices outputted from our matrix producing code. After that, we must use the matrix to begin building our relations one by one. This is done by looking at the values at each point then finding out what that value equals in relation to our group, then linking that value to its pair and the group element associated.

We begin with a matrix created from our program. This matrix takes into account the group being both non-degenerate and minimally realizable. Thus, to review, each values has a pair and there are no repeated values in a row or column.

For most of the groups we used *SAGE* to convert our relations into groups, however, we will provide examples of the algorithm by hand.
Example 5.3. Let

$$M = \begin{bmatrix} x & 1 & 2 \\ 1 & 3 & 4 \\ 4 & 2 & 3 \end{bmatrix}$$

Then the group Γ_M generated by $a_1, ..., b_2$ is subject to the following relations.

$$R =< x^4 = 1 >$$

We can find the group's relations by hand using the algorithm. From the matrix we start by labeling our top row, thus b_1 = "1" and b_2 = "2". Next, we label our leftmost column, thus $a_1 = 1$ and $a_2 = 4$. We can now begin defining our relations.

Because a_1 and b_1 both equal "1", our first relation is $a_1 = b_1$. We know that the "4" in the last column equals $a_1 b_2$, thus $a_2 = a_1 b_2$. We can see that $a_1 b_1 = a_2 b_2$ as they are both "3". Lastly, $b_2 = a_2 b_1$. Thus our relations have been defined as:

$$a_1 = b_1$$
$$a_2 = a_1 b_2$$
$$a_1 b_1 = a_2 b_2$$
$$b_2 = a_2 b_1$$

We can then take a free group on our four generators a_1, a_2, b_1, b_2 and divide by our relations which we will collectively label R. From this we get a quotient group F/R equivalent to $\mathbb{Z}_4$.

We check this ourselves by reworking our relations. For instance:

$$a_1 b_1 = a_1^2$$

thus,

$$a_1^2 = a_2 b_2$$

From this we can guess that a_1 is not our "base" generator. We see that $b_2 = a_2 b_1$, thus b_2 is most likely not our base generator. Guessing that $a_2 = x$ gives us that $b_2 = x b_1$ and $a_1^2 = x b_2$. Thus,

$$a_1^2 = x x b_1 \rightarrow a_1^2 = x x a_1 \rightarrow a_1 = x^2$$

From this result it follows that $b_2 = a_2 b_1 = x x^2 = x^3$ and lastly $a_1 = x^2 = b_1$. Thus, we can see that the group we built is $\mathbb{Z}_4$.

Example 5.4. Let

$$M = \begin{bmatrix} x & 1 & 2 & 3 & 4 \\ 1 & 5 & 3 & 6 & 7 \\ 4 & 7 & 5 & 2 & 6 \end{bmatrix}$$

Then the group Γ_M generated by $a_1, ..., b_4$ is subject to the following relations

$$R = < x^7 = 1 > .$$

We can find the group's relations by hand using the algorithm.

$$a_1 = b_1, \quad b_4 = a_2, \quad b_2 a_1 = b_3$$
$$b_1 a_1 = b_2 a_2, \quad b_1 a_2 = b_4 a_1, \quad b_3 a_1 = b_4 a_2$$
$$b_3 a_2 = b_2$$

To begin solving for the group we will solve each generator in terms of a_1:

$$b_1 = a_1, \quad b_2 = a_1^{-4}, \quad b_3 = a_1^{-1}$$
$$b_4 = a_1^{-1}, \quad a_2 = a_1^{-1}$$

From these relations we can see that $a_2 = b_4$. Setting $a_1 = x$, we obtain for our group:

$$a_1 = x, \quad a_2 = x^6, \quad b_1 = x$$
$$b_2 = x^3, \quad b_3 = x^4, \quad b_4 = x^6$$

Thus our group is the cyclic group $\mathbb{Z}_7$.

We will now cover a more complicated group.

Example 5.5. Let

$$M = \begin{bmatrix} x & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 2 & 3 & 9 & 5 & 10 & 11 & 12 & 13 \\ 6 & 11 & 7 & 12 & 8 & 13 & 4 & 10 & 9 \end{bmatrix}$$

Then the group Γ_M generated by $a_1, ..., b_8$ is subject to the following relations:

$$R =< x, y : y^4 = 1, xy = yx >$$

We can find the group's relations by hand using the algorithm.
From this matrix we can obtain the following relations:

$$b_1 = a_1, \quad b_1a_1 = b_2, \quad b_1a_2 = b_6a_1$$
$$b_2a_1 = b_3, \quad b_2a_2 = b_7, \quad b_3a_1 = b_8a_2$$
$$b_3a_2 = b_7a_1, \quad b_4a_1 = b_5, \quad b_4a_2 = b_8$$
$$b_5a_1 = b_7a_2, \quad b_5a_2 = b_8a_1, \quad b_6a_2 = b_4$$
$$b_6 = a_2$$

We can approach solving such a complex group by first getting every generator in terms of another. In this case, we will attempt to solve every generator in terms of a_1. After using our relations we reduce every generator into:

$$b_1 = a_1, \quad b_2 = a_1^2, \quad b_3 = a_1^3$$
$$b_4 = a_2^2, \quad b_5 = a_2^2a_1, \quad b_6a_2$$
$$b_7 = a_2^2a_1^2a_2^{-1}, \quad b_8 = a_2^3$$

The inability for us to reduce each generator to the single "base" generator, tells us that our group has more than one generator.

We relabel a_1 to x and a_2 to y and relabel our relations to:

$$(x, x^2, x^3, x^2y^2, x^3y^2, xy, x^3y, x^3y^3)$$

We get an equivalent group of $\mathbb{Z} \times \mathbb{Z}_4 \cong < x, y : y^4 = 1, xy = yx >$.

Example 5.6. Let

$$M = \begin{bmatrix} x & 1 & 2 & 3 & 4 \\ 1 & 2 & 5 & 6 & 7 \\ 6 & 4 & 7 & 5 & 3 \end{bmatrix}$$

Our group Γ_M generated by $a_1, ..., b_4$ is subject to the following relations:

$$R =< x^8 = 1 > .$$

We can find the group's relations by hand using the algorithm.
Thus, M gives:

$$\begin{gathered} a_1 = b_1, \quad b_2 a_2 = b_4 a_1, \quad b_1 a_2 = b_4 \\ b_3 a_1 = a_2, \quad b_1 a_1 = b^2, \quad a_1 b_2 = b_5 \\ b_4 a_1 = b_2 a_2, \quad a_2 b_4 = b_3, \quad b_3 a_2 = a_1 b_2 \end{gathered}$$

From these relations we get our group $\mathbb{Z}_8$.

Example 5.7. Let

$$M = \begin{bmatrix} x & 1 & 2 & 3 & 4 \\ 5 & 2 & 1 & 6 & 7 \\ 6 & 5 & 7 & 4 & 3 \end{bmatrix}$$

Then the group Γ_M generated by $a_1, ..., b_4$ is subject to the following relations.

$$R =< x, y : x^4 = y^2 = 1, yxy = x^{-1} >$$

We can find the group's relations by hand using the algorithm.
From M we obtain these relations:

$$\begin{gathered} b_1 a_1 = b_2, \quad b_1 a_2 = a_1, \quad b_2 a_1 = b_1 \\ b_2 a_2 = b_4 a_1, \quad b_3 a_1 = a_2, \quad b_3 a_2 = b_4 \\ b_4 a_2 = b_2 \end{gathered}$$

This gives the dihedral group D_4.

6 Conclusion

Our theoretical contributions to the field include a detailed expansion on a previous proof of Gottschalk's conjecture implying Kaplansky's conjecture for infinite groups and an alternate proof that Kaplansky's conjecture is satisfied for finite groups.

Our computational contributions to the field include a unique algorithm for checking Kaplansky's conjecture for finite groups and an outline of a unique implementation of a previous algorithm [4] for checking Kaplansky's conjecture for infinite groups.

Ultimately, we have furthered our theoretical understanding of Kaplansky's and Gottschalk's conjecture; as well as, advanced our understanding of rewriting systems.

Due to the incomplete nature of our project before COVID-19 and a hard-drive failure on our workstation server, we were unable to fully complete the code for checking Kaplansky's conjecture for infinite groups. Our algorithm uniquely creates the special matrices representing ULIE groups, and is able to check these groups if they satisfy Kaplansky's conjecture. The code is one step from running on a single computer, and a further step away from being able to run a cluster network. These aspects of the project are expected to be completed over the summer.

References

[1] P. S. Novikov. 1956, Algorithmic unsolvability of the word problem in group theory. *Trudy Matématičéskogo Instituta iméni V. A. Stéklova, vol. 44. Izdatél'stvo Akadémii Nauk SSSR, Moscow1955, 143 pp. Journal of Symbolic Logic***23**:50-52.

[2] A. Seress. 1997, An introduction to computational group theory. *vol. 44. number 6, Notices of the American Mathematical Society.*

[3] G. Elek, E. Szabo 2004, Sofic groups and direct finiteness. *eprint arXiv:math/0305440.*

[4] K. Dykema, T. Heister, K. Juschenko 2012, Finitely presented groups related to Kaplansky's direct finiteness conjecture. *eprint arXiv:1112.1790.*

[5] J. Crisp, E. Godelle, B. Wiest 2008, The conjugacy problem in right-angled Artin groups and their subgroups. *eprint arXiv:0802.1771.*

[6] J. Von Neumann, A. Burks 1966, Theory of Self-Replicating Automata. *University of Illinois Press, Champaign, IL, USA 1966*

[7] J. Kari 2013, Lecture notes on Cellular Automata, 2013. *http://users.utu.fi/jkari/automata/.*

[8] C. Kolon 2017, Proposed Method of Investigation, Autonomous Pattern Formation via Dynamical Systems. *United States Naval Academy.*

[9] I. Kaplansky, 1969, Fields and Rings, *Chicago Lectures in Mathematics, Univ. of Chicago Press, Chicago, (1969)*

[10] V. Berthe, M. Rigo, 2016, Combinatorics, Words and Symbolic Dynamics, *Encyclopedia of Mathematics and Its Applications 159*

[11] T. Ceccherini-Silberstein, M. Coorbaert, 2010, Cellular Automata and Groups, *Springer Berlin Heidelberg*

[12] C. Sims, 1994, Computation With Finitely Presented Groups, *Cambridge University Press*

[13] T. Hurley, 2015, Representations of group rings and groups. *eprint arXiv:1506.05149.*

[14] M. Gromov, 1999, Endomorphisms of symbolic algebraic varieties, *Springer JEMS*

[15] T. Judson, 2019, Abstract Algebra: Theory and Applications, *Abstract Algebra: Theory and Applications*

[16] O. Bogopolski, 2008, Introduction to Group Theory *EMS Textbooks in Mathematics*

[17] E. Pegg, 2018, WireWorld, *http://mathworld.wolfram.com/WireWorld.html*

[18] B. Sury, 2010 Free Groups, *https://www.isibang.ac.in/ sury/aisiit.pdf*

[19] D. Cohen, 1989, Combinatiorial Group Theory: A Topological Approach *Cambridge University Press*

[20] B. Weiss, 2000, Sofic Groups and Dynamical Systems, *https://pdfs.semanticscholar.org/; sofic groups and dynamical system*

[21] P. Ara, K. O'Meara, and F. Perera, 2002, Stable finiteness of group rings in arbitrary characteristic, *Advances in Mathematics, Volume 170, Issue 2*

[22] W. Gottschalk, 1973, Some general dynamical notions, *Recent Advances in Topological Dynamics, Lecture 318*

[23] S. Capobianco, J. Kari, S. Taati, 2016, An Almost Dual To Gottschalk's Conjecture, *https://siamak.isoperimetric.info/*

[24] D. Holt, B. Eick, E. O'Brien, Handbook of Computational Group Theory, *Chapman and Hall/CRC*

Appendix A Computational Results

A.1 Finite Groups Code in *SAGE*

We began building our first program by constructing a working word problem solver in *SAGE*. This program allowed us to use *SAGE's* built in functions for building a group and its group algebra, and constructing a confluent rewriting system for the group. The code for this is shown in figures 2 and 3.

Our program was able to accurately check S_3 for finite fields of 2,3, and 4.

In [1]:
```
'''
Program to initialize a symmetric group and field, then using SAGE, to build t
heir group algebra.

'''

#Initialze a symmetric groups with SAGE and presented with generators and rela
tions
G=groups.presentation.Symmetric(3)

#Define a rewriting system and then make it confluent for our program to be ab
le to reduce words to a canonical form
kG=G.rewriting_system()
kG.make_confluent() # key step
G
```

Out[1]:
```
Finitely presented group < a, b | b^2, a^3, (a*b)^2 >
```

In [2]:
```
#define explicitly the generators for our group
[a,b]=G.gens()

#initialize our field, in this case, a field of size two
R=GF(2)

#initialize our group algebra
GA = GroupAlgebra(G,R)
GA
```

Out[2]:
```
Algebra of Finitely presented group < a, b | b^2, a^3, (a*b)^2 > over Finite
Field of size 2
```

In [3]:
```
# reduce the coefficients of a group ring polynomial using the rewriting syste
m
def reducePol(pol,rewriteSys):
    coeff = pol.monomial_coefficients()
    return sum([GA(v)*(rewriteSys.reduce(u)) for (u,v) in coeff.items()])
```

In [4]:
```
#builds the elements in our group

grRingList = []
grElems = list(G)
for v in Tuples(list(R),len(G)):
    ringTuple = map(GA,v) # convert the tuple of integers into the tuple of el
ements of the group ring R[G] (data type conversion)
    grRingList.append(sum(map(lambda x,y: x*y,ringTuple,grElems)))

grElems
```

Out[4]:
```
[1, a, a^-1, b, a*b, a^-1*b]
```

Figure 2: First initial *SAGE* code.

```
In [*]: from multiprocessing import Pool
        from contextlib import closing
        cardGroupRing = len(grRingList)
        def myFun(i):
            v= grRingList[i]
            for j in range(i,cardGroupRing):
                w = grRingList[j]
                if reducePol(v*w,kG)==1:
                    if reducePol(w*v,kG)==1:
                        print("Yes",v,w)
                        return[("Yes",v,w)]
                    else:
                        print("No",v,w)
                        return[("No",v,w)]

        with closing(Pool(processes=24)) as pool:
            res = pool.map(myFun,range(cardGroupRing))
            pool.terminate()
```

```
('Yes', 1, 1)
('Yes', 2, 2)
('Yes', a, a^-1)
('Yes', 2 + 2*a, 1 + 2*a + a^-1)
('Yes', 1 + b + a + a^-1, 2 + b + 2*a + 2*a^-1)
('Yes', 1 + a, 2 + a + 2*a^-1)
('Yes', 1 + 2*b + 2*a + 2*a^-1 + a*b, 1 + b + 2*a + 2*a^-1 + 2*a*b)
('Yes', 2*b + a^-1 + a*b, 2 + b + 2*a^-1 + 2*a*b)
('Yes', 1 + 2*b + a*b, b + 2*a + 2*a^-1 + 2*a*b)
('Yes', 2 + 2*b + 2*a + a^-1 + a*b, 2 + b + a + 2*a^-1 + 2*a*b)
('Yes', 2*b + 2*a^-1 + a*b, 1 + b + a^-1 + 2*a*b)
('Yes', 2 + 2*b + a*b, b + a + a^-1 + 2*a*b)
('Yes', 1 + 2*b + a^-1 + a*b, b + 2*a^-1 + 2*a*b)
('Yes', 2*b + a + a*b, 2 + b + 2*a + 2*a*b)
('Yes', 1 + 2*b + a + a*b, b + 2*a + 2*a*b)
('Yes', 1 + 2*a + a*b, 2 + 2*b + 2*a^-1*b + a^-1)
('Yes', 2*a + a^-1 + a*b, 2*b + 2*a + 2*a^-1*b + a^-1)
('Yes', 1 + a^-1, 2 + 2*a + a^-1)
('Yes', 2 + b + a, 1 + 2*a^-1*b + 2*a^-1 + 2*a*b)
('Yes', 2*a, 2*a^-1)
('Yes', a*b, a*b)
('Yes', 1 + 2*a + 2*a*b, 2 + b + a^-1*b + a^-1)
```

Figure 3: *SAGE* output.

A.2 Finite Groups Code in Python

We next built our second program in *Python* for speed improvements and to build scaffolding for how we would construct a program to check infinite groups in *Python*. Our second program is shown in figure 4 and figure 5, each are screenshots of part of our code.

```
def GA_total_test(n, field_size):
    #inputs n and field size to run through entire program

    global field
    #ensures all local variables created are global
    global G_perm
    global G_minus
    global R_G
    global N_G1
    global N_G2
    global GA_len
    global idn_test

    global checks
    global failures
    global count_gens

    global N_E1
    global N_E2


    #measures time for function to complete
    t0 = time.time()

    #initializes field list
    field =[]
    for i in range(field_size):
        field.append(i)

    #imports perumtation matrices from csv file
    G_perm = G_matrix_import()

    #initializes inverted permutation matrices to create overall Group Matrice
s
    G_minus = []

    #creates list of inverses of symmetric group to help build group matrix la
ter
    G_minus = np.linalg.inv(G_perm)

    #checks that program initialized properly
    print 'BANG'

    #cleans up data to be readable as numpy arrays
    R_G = np.zeros((len(G_perm),len(G_perm)),dtype=object)
```

Figure 4: *Python* code example.

```
#creates 'RG' group matrix (NOT group algebra matrix yet)
for i in range(len(G_perm)):
    for j in range(len(G_perm)):
        R_G[i][j] = np.matmul(G_minus[i],G_perm[j])

#cleans up data to be readable as numpy arrays
R_G = np.array(R_G)

#initializes empty group algebra matrices
N_G1 = np.zeros((len(G_perm),len(G_perm)))
N_G2 = np.zeros((len(G_perm),len(G_perm)))

#creates list of all our group algebra elements as tuples of 1s and 0s
GA_len = (len(field))**(len(G_perm))

#initializes identity matrix to be compared
idn_test = np.identity(len(G_perm))

#initializes seed tuples
seed0 = []
for i in range(len(G_perm)):
    seed0.append(0)

seed1 = seed0[:]
seed2 = seed0[:]

#verifies program to checking correct number of Group Algebra elements
print GA_len

#cycles through and tests each group algebra element
checks = []
failures = []
count_gens = 0
```

Figure 5: *Python* code example.

A.3 Infinite Groups Code

Lastly, after constructing code to check finite groups for Kaplansky's conjecture we began building our third program code to check for infinite groups. Figure 6 and figure 7 are screen shots of part of our third program for building the ULIE matrices. You will see that we began by building matrices that satisfied being non-degenerate and minimally realizable (and thus being a ULIE group), then generating groups out of the matrices, and finally constructing a confluent rewriting system to check Kaplanksy's conjecture for the created group.

```
1  import numpy as np
2
3  i_m = [[0,1,2,3,4],[1,2,5,6,7],[6,4,7,5,3]]
4
5  width = len(i_m[0])
6  height = len(i_m)
7
8  b_vals = []
9  a_vals = []
10
11 #print height
12 #print width
13
14 aGens = []
15 bGens = []
16
17 for i in range(height):
18     g = "a" + str(i)
19     aGens.append(g)
20
21 for i in range(width):
22     g = "b" + str(i_m[0][i])
23     bGens.append(g)
24
25 print aGens
26 print bGens
27
28 rGens = []
29
30
31 larg = 0
```

Figure 6: Part 1 of ULIE code.

```
larg = 0
for i in range(height):
    a = max(i_m[i])
    if a > larg:
        larg = a
print larg


for i in range(len(i_m)):
    for k in range(larg):
        if k+1 in i_m[i]:
            b = i_m[i].index(k+1)
            temp1 = [i,b]
            #print 'temp1',temp1
            for j in range(len(i_m)):
                if k+1 in i_m[j]:
                    a = i_m[j].index(k+1)
                    if a == b:
                        continue
                    else:
                        temp2 = [j,a]
                        #print 'temp2',temp2

                    #print 'j',j

                    count = 0
                    ggens = []

                    if b != 0:
                        g2 = bGens[b]
                        count = count + 1
                        ggens.append(g2)

                    if i != 0:
```

Figure 7: Part 2 of ULIE code.

```
                    if i != 0:
                        g1 = aGens[i]
                        count = count + 1
                        if b == 0:
                            ggens.append(g1)
                        else:
                            ggens.append('*' + g1)

                    if j != 0:
                        g3 = aGens[j]
                        count = count + 1
                        if i == 0 and b == 0:
                            ggens.append(g3)
                        else:
                            ggens.append('*' + g3 + '^-1')

                    if a !=0:
                        g4 = bGens[a]
                        count = count + 1
                        if i == 0 and b == 0 and a == 0:
                            ggens.append(g4)
                        else:
                            ggens.append('*' + g4 + '^-1')

                    #print ggens

                    #print count

                    ggens2 = ggens[0]

                    for i in range(len(ggens)-1):
                        ggens2 = ggens2 + ggens[i+1]

                    rGens.append(ggens2)
```

Figure 8: Part 3 of ULIE code.

```
                    rGens.append(ggens2)


tGens = []

for i in range(len(aGens)-1):
    tGens.append(var(aGens[i+1]))

for i in range(len(bGens)-1):
    tGens.append(var(bGens[i+1]))


print aGens
print type(aGens[1])
print tGens
print type(tGens[1])

print rGens


F.<a1, a2, b1, b2, b3, b4> = FreeGroup()


G = F/tGens

print rGens
print rGens[1]
print type(rGens[1])

rGens2 = []

for i in range(len(rGens)):
    rGens2.append(F(rGens[i]))

```

Figure 9: Part 4 of ULIE code.

```
                    rGens.append(ggens2)


tGens = []

for i in range(len(aGens)-1):
    tGens.append(var(aGens[i+1]))

for i in range(len(bGens)-1):
    tGens.append(var(bGens[i+1]))


print aGens
print type(aGens[1])
print tGens
print type(tGens[1])
print rGens

F.<a1, a2, b1, b2, b3, b4> = FreeGroup()

G = F/tGens

rGens2 = []

for i in range(len(rGens)):
    rGens2.append(F(rGens[i]))


G = F / [b1*a1^-1, b1*a1*b2^-1, b1*a2*a1^-1*b6^-1, b2*a1*b3^-1, b2*a2*b7^-1, b3*a1*a2^-1*b8^-1, b3*a2*a1^-1*b7^-1, b4*a1*b5^-1, b4*a2*b8^-1, b5
k = G.rewriting_system()
k.make_confluent


```

Figure 10: Part 5 of ULIE code.

Appendix B Additional Background Knowledge

B.1 The Word Problem in Computational Group Theory

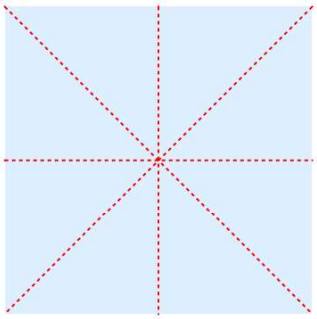


Figure 11: The rotational and reflective symmetries of a square.

We begin by briefly discussing mathematical objects called *groups*. In nature, shapes and objects have inherent symmetry. A group can be one defined as the system with which we represent this symmetry. In Figure 11 we see the possible reflections of a square. Further symmetries would be rotations of 90, 180, 270, and 0 degrees. Each of these actions, or generators, preserves the shape and space of the square. The group of symmetries of a square is called the dihedral, D_4, group. With only 1 reflection (s_1), and 1 rotation (r_1), one can build the entire group. For instance, if we define "r_1" as a rotation of 90 degrees. Then $r_1 \circ r_1 = r_2$ is a rotation of 180 degrees. These basic building blocks, r_1 and s_1, are called the *generators* of our group. The *relations* of the group define the interactions our generators have and build the algebraic structure of the group. For instance, one relation says that the same reflection repeated twice equals the original square. In terms of our square, if one rotates about the horizontal axis twice, then one obtains the original square back. Thus, $s_1 \circ s_1 = s_1^2 = e$. We can define the notation of the *inverse* of s_1 as s_1^{-1}. In this case, $s_1 = s_1^{-1}$. Figure 12 demonstrates generators building D_4. Each blue edge represents the same rotation, and each red edge represents the same reflection. Every possible representation of a square - 8 total - are represented by the vertices and generated by 1 rotation and 1 reflection.

Previously we defined a group as a collection of words (or objects), and a set of cancellation words (or relations). In this case, our rotations and reflections are our objects, and the relations of these reflections and rotations (like $s_1 \circ s_1 = s_1^2 = e$) are our cancellation words.

The word problem proposed by Max Dehn, a German-American mathematician in 1911, originates from the interplay between generators and relations [2]. Dehn proposed that if one has a long list of generators (which we call a *word*), then, for some groups there exists an algorithm such that one can check, using the group's relations, if that word is equivalent to the identity.

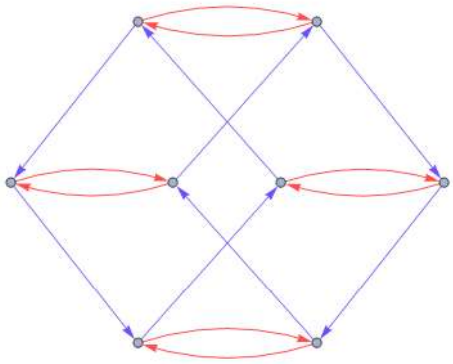


Figure 12: The Cayley graph for the D_4 group. Blue edges represent a rotation, r_1 of 90°, red edges represent a reflection, either s_1, s_2, s_3, or s_4.

Before moving forward we will describe what an *algorithm* is. An algorithm is a set of rules or steps. If the word problem is *decidable* then there exists a word problem solving algorithm. We say a word problem is *decidable* if we can identify if two words equal each other.

Finding an algorithm for word problems is group dependent [2]. Pytor Novikov in 1956 found that for some groups an algorithm exists but for others no such algorithm exists [1]. For each group that does have a solution to its word problem, the specific solution or algorithm is group specific. We will illustrate with an example in which we discuss the solution of the word problem for right-angled Artin groups.

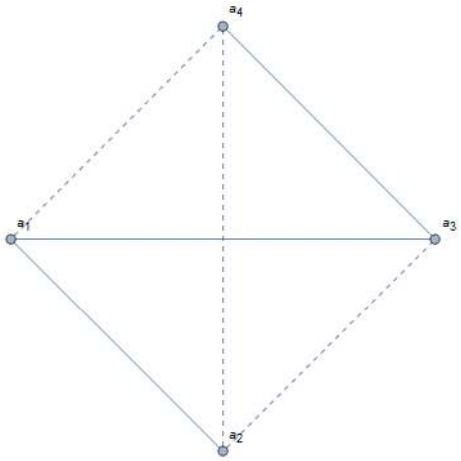


Figure 13: The graph for a right-angled Artin group. In this case if two generators, for example a_1 and a_4, commute such that $a_1a_4 = a_4a_1$, then that relationship is represented by a dashed edge. For elements that do not commute, that relationship is represented by a unidirectional solid edge.

Example: A right-angle Artin group can be defined as a group given by a finite presentation, where the relations defining the structure of the group are pairs of generators that commute [5]. Thus, if looking at the group as a *graph*, then graphically the relations are defined as where vertices on the graph are *not* connected by an edge. Figure 13 shows the graph of a possible right-angled Artin group with four generators. *Words* for this group would take the form of $a_1a_2^2a_4a_3^{-1}a_1^3$ or a similar element with combinations of a_1, a_2, a_3, and a_4.

For the right-angled Artin group we can build an algorithm that *reduces* a *word* in the group to its basic, or smallest, form. If this word reduces to e, then we say the word is equivalent to the identity. The process, or algorithm, is called the *piling method* for the right-angle Artin group. This algorithm was suggested by John Crisp, Eddy Godelle, and Bert Wiest in 2008 [5].

We begin discussing the *piling* problem by defining a function π on the infinite set of possible words from our previous example. This function associates an abstract piling to every word in the group by first starting with an empty piling and then reading left to right. When we encounter an $a_i^{\pm 1}$ from our word, we check what the last letter on that specific stack is. If the letter is the opposite sign, then we remove the letter. If the letter is something else, either a 0 or the same letter, then we append that letter onto its designated stack and a blank (white) ball onto the other stacks [5]. The red balls represent positive elements and the yellow balls represent negative elements. If when building this piling the word reduces to an empty *piling*, then we know that the original word reduces to the identity. To illustrate, we can begin with letting our right-Artin group, A, be represented by:

$$\langle\ a_1, a_2, a_3, a_4 \mid\ \ a_1a_4 = a_4a_1;\ \ a_2a_3 = a_3a_2;\ \ a_2a_4 = a_4a_2\ \rangle$$

Next we can build a word in A, $w = a_2^{-2}a_4^{-1}a_3a_2a_4a_1a_2a_1^{-1}a_2^2a_4^{-1}$ [5]. Applying our piling function π, we create our piling p, shown in Figure 14. The map, π is a well-defined function that preserves the image of the word. Thus, the output (the reduced word) is the same algebraically to the input. This piling algorithm exchanges elements based on the commutative relations above and eliminates $a_ia_i^{-1}$ and $a_i^{-1}a_i$ pairs. Thus, if the result is equal to e, then the original word is equal to e [5].

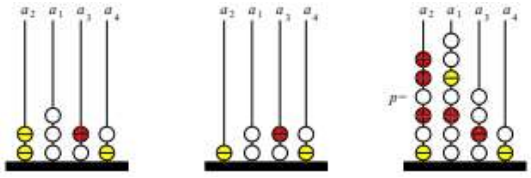


Figure 14: Example of a *piling* for $a_2^{-2}a_4^{-1}a_3$, $a_2^{-2}a_4^{-1}a_3a_2$, and the original word, $a_2^{-2}a_4^{-1}a_3a_2a_4a_1a_2a_1^{-1}a_2^2a_4^{-1}$, respectively. [5].

This algorithm can be implemented to run on a computer.

B.2 An Intuitive Interpretation of Gottschalk's Conjecture

When analyzing Gottschalk's conjecture, an intuitive understanding comes from cellular automata. We define a cellular automata as a grid with random squares turned "on", and then a rule-set that defines how other squares turn "on" or "off."

Gottschalk's conjecture states that if the dynamics on a Cayley graph are one-to-one, then they are onto. The dynamics, or rule-set on a grid, are one-to-one if distinct configurations have distinct successors. The dynamics are onto if every configuration has a specific predecessor.

Groups are determined by their Cayley graphs, see Figure 1. One way to study groups is to look at the *dynamics* of their Cayley graphs. We can study dynamical behavior on graphs with either *group actions* or cellular automata. I have subsequently studied the computational aspects of dynamics on many Cayley groups, see Figure 24. Furthermore, Gottschalk's conjecture and cellular automata are connected to Kaplanksy's conjecture.

Briefly, the background of cellular automata can be traced back in concept to John Von Neumann in the 1940s [6]. A cellular automaton is a discrete dynamical system that consists of a regular network of finite state automata (cells) that change states depending on the states of their neighbors [7]. We have investigated cellular automaton beginning with the most famous example, the *Game of Life*.

We began by designing a program to run a cellular automaton in 1-dimension with *Mathematica*. Different cellular automatons are referred to as a "rule" when in 1-dimension. Each rule corresponds to the binary number representation when the rule is laid out in 1-dimension. In this case we used "Rule 150." Figure **??** and Figure **??** show screen-shots of our program at different frames.

We continued with designing our own 2-dimension cellular automaton with the rules of *Game of Life* programmed independently into *Mathematica*. Figure 15 and Figure 16 show screenshots of our program at different frames.

We next designed a program in *Mathematica* to stack our 2D frames into a 3D model, to represent the "growth" occurring in our cellular automaton. Figure 17 and Figure 18 show screenshots of our program at different frames.

Afterwards we extrapolated the "rules" for Game of Life into a 3-dimension cube, creating a cellular automaton that functions in 3-dimension space. Figure 19 and Figure 20 show screenshots of our program at different frames.

Cellular automata, and specifically the Game of Life, have interesting properties of being Turing complete. Thus, one is able to build a computer, capable of any algorithmic calculation, within the cellular automaton.

We investigated this by first looking at a computer designed with the rules of *Wireworld*, a similar automaton to *Game of Life*. The first goal is to build different NAND, AND, NOR, and XOR gates that can act as logic gates. Taken from the *Wolfram Mathematica* database, we found the format for building these gates in *Wireworld*.

The gates in Figures 21 and 22 provide the foundation with which one could build a computer inside the *Wireworld* or *Game of Life* rule-set. This is shown in Figure **??**.

We next began applying our automata to other *homogeneous graphs*. A homogeneous

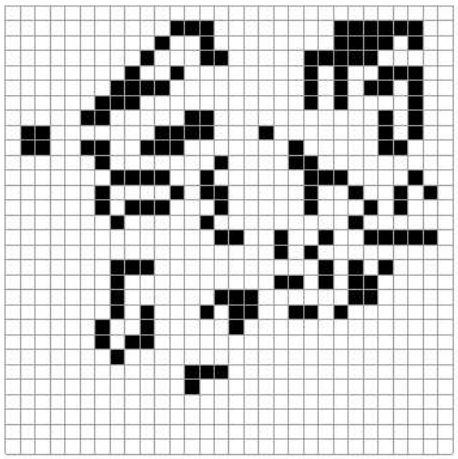


Figure 15: Game of Life 2D gif, with screenshot taken at frame 4.

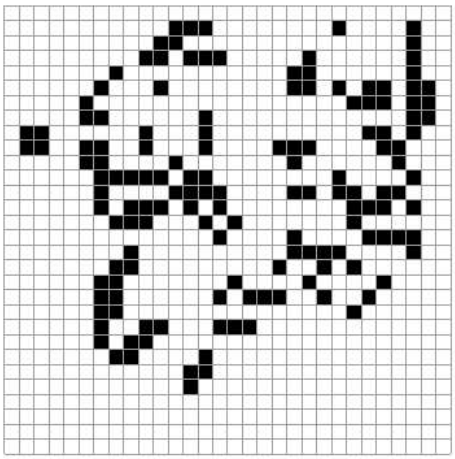


Figure 16: Game of Life 2D gif, with screenshot taken at frame 5.

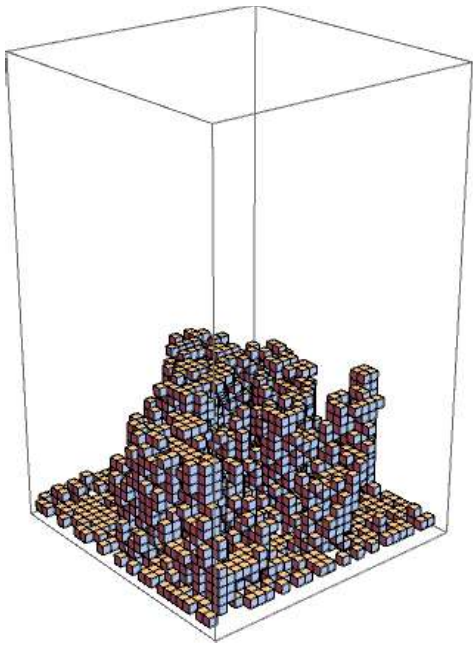


Figure 17: Game of Life 3D gif, with screenshot taken at frame 25.

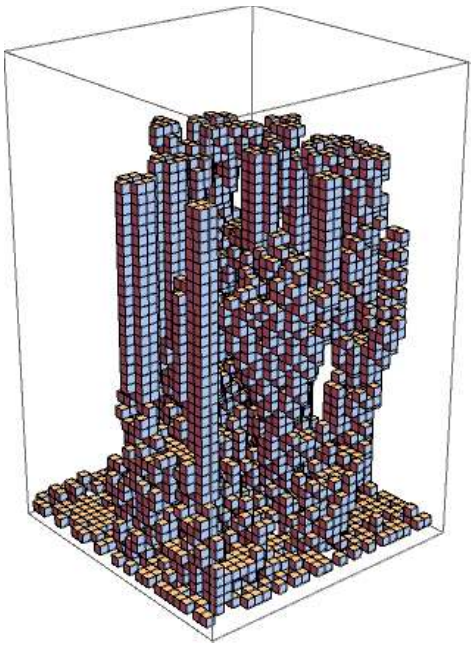


Figure 18: Game of Life 3D gif, with screenshot taken at frame 40.

graph is a graph with each point having the same topology. Thus, the graph at one vertex has an identical structure to another vertex. Within a 2-D grid structure, a point has 8 neighbors surrounding it. Moving anywhere else in the graph, a point still has only 8 neighbors. Thus we can say that a grid is a homogeneous graph. We can apply cellular automata to any homogeneous graph. Figure 12 demonstrates a homogeneous graph. At

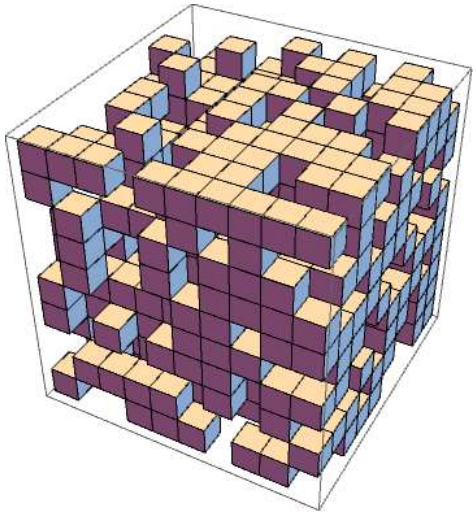


Figure 19: Game of Life 3D-space gif, with screenshot taken at frame 2.

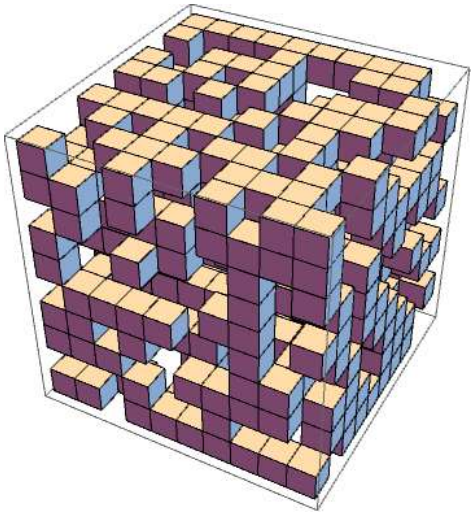


Figure 20: Game of Life 3D-space gif, with screenshot taken at frame 3.

Figure 21: NAND, AND, NOR, OR gates gif. Screenshot taken at frame 3. Referenced from [17]

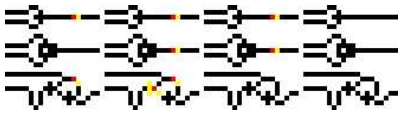


Figure 22: NAND, AND, NOR, OR gates gif. Screenshot taken at frame 15. Referenced from [17]

any vertex there are four edges: 2 leading towards the point and 2 leading away. We then took similar rules governing the cellular automaton in 1-D, 2-D, and 3-D, and applied them to a more complex Cayley graph of the permutation group generated by cycles $(1, 5, 4)$ and $(3, 4)$. This is shown in Figure 24 and figure 25.

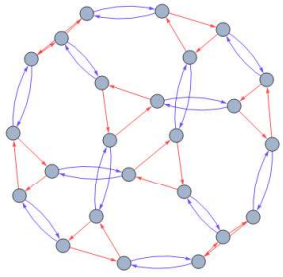


Figure 23: The Cayley graph for the permutation group generated by (1,5,4) and (3,4).

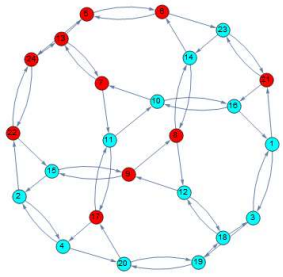


Figure 24: Cayley graph gif for permutation group (1,5,4) and (3,4), with cellular automata overlaid. Screenshot taken at frame 4.

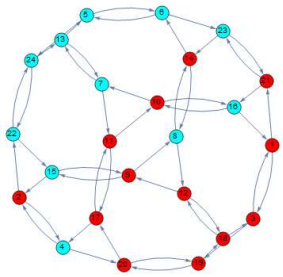


Figure 25: Cayley graph gif for permutation group (1,5,4) and (3,4), with cellular automata overlaid. Screenshot taken at frame 5.

Appendix C Glossary of Abstract Algebra Terms

We outline a brief overview of Abstract Algebra, including group and field properties. All definitions, theorems, and proofs were taken from [15] and [16].

Algebraic Structures Definition: A nonempty set, S, a nonempty collection of operations on S, and a collection (which may be empty) of relations on S.

Properties: All Algebraic structures have four common properties:

- Associativity: $(S, *)$ is associative if and only if $(x*y)*z = x*(y*z)$ for all $x, y, z \in S$.
- Existence of an Identity: $(S, *)$ has an identity element (or simply an identity) if and only if there exists an $e \in S$ such that $x * e = e * x = x$ for all $x \in S$.
- Existence of Inverses: $(S, *)$ has inverses if and only if S has an identity element e and for each element $x \in S$ there is an element $y \in S$ called the inverse of x, such that $x * y = y * x = e$.
- Commutativity: $(S, *)$ is commutative if and only if $x * y = y * x$ for all $x, y \in S$.

Basic Proofs

Theorem A3.1: Let $(S, *)$ be an algebraic structure. If $(S, *)$ has an identity it is unique.

Proof: Assume towards contradiction that $(S, *)$ has two distinct identity elements e and e'. By the definition of an identity, for every $a \in S$, (1) $a * e = e * a = a$, and (2) $a * e' = e' * a = a$. In equation (1) we let $a = e'$. The right-hand side of the resulting equation is $e * e' = e'$. Letting $a = e$ in equation (2), we obtain $e * e' = e$. It follows that $e' = e$, which is a contradiction. Thus, if an identity of an algebraic structure exists, it is unique.

Theorem A3.2: Let $(S, *)$ be an associative structure with identity e. If $a \in S$ has an inverse, then the inverse is unique.

Proof: Suppose that $(S, *)$ is associative with identity e and suppose that there exists an $a \in S$ and elements $x, y \in S$ such that x and y are inverses of a and $x \neq y$. Since x and y are inverses of a, (1) $a * x = e$ and (2) $y * a = e$. Multiplying (1) on the left by y yields $y * (a * x) = y * e = y$. By associativity of $*$ and equation (2), we have $y * (a * x) = (y * a) * x$. It follows that $y = x$, which is a contradiction. Thus the inverse element must be unique.

Group Theory We continue our discussion with the definition, properties, and basic proofs of Group Theory.

Properties Definition: A group is a nonempty set G with a binary operation $\circ$ that satisfies the following three properties:

- Associativity: $(G, \circ)$ is associative. That is, $(x \circ y) \circ z = x \circ (y \circ z)$ for all $x, y, z \in G$.
- Existence of an Identity: There exists an identity element $e \in G$. That is, there exists an $e \in G$ such that $x \circ e = e \circ x = x$ for all $x \in G$.
- Existence of Inverses: Every element $x \in G$ has an inverse $x^{-1} \in G$. That is, for all $x \in G$ there exists an $x^{-1} \in G$ such that $x \circ x^{-1} = x^{-1} \circ x = e$

Note: If the group also demonstrates commutativity, the group is called *Abelian.*

Although a group requires the property of being "closed," by specifying that the operation, $\circ$, is binary ensures the operation is closed.

Basic Definitions

Set: Well defined collection of objects.

Function: A function ϕ mapping X into Y is a relation between X and Y with the property that each $x \in X$ appears as the first member of exactly one ordered pair (x, y) in ϕ. Such a function is also called a map or mapping of X into Y. We write $\phi : X \to Y$ and express $(x, y) \in \phi$ by $\phi(x) = y$. The domain of ϕ is the set X, and the set Y is the codomain of ϕ. The range of ϕ is $\phi[x] = \{\phi(x) | x \in X\}$.

Injectivity: A function $\phi : X \to Y$ is injective if $\phi(x_1) = \phi(x_2)$ only when $x_1 = x_2$.

Surjectivity: The function ϕ is onto Y if the range of ϕ is Y.

Bijectivity: A function is Bijective if it is both Injective and Surjective.

Partition: A partition of a set, S, is a collection of nonempty subsets of S such that every element of S is in exactly one of the subsets. The subsets are the cells of the partition.

Subgroup: **(Normal)** When the left and right cosets of a group coincide. In order for a subgroup to be defined as a *Normal Subgroup*, the subgroup, H, must contain $g'^{-1}hg'$, for $gH \circ g'H = (gg')H$.

Subgroup: A subset of a group with the properties that it is closed under the binary operation, if x is in H, then x^{-1} is in H, and that it contains an identity element. Can be treated as a group in of itself.

Coset: Hg, gH. Right and left coset respectively. Defined as the sets:

$$Hg = hx : h \in H$$

$$gH = xh : h \in H$$

Can also be defined as the set H performed with every element of G. Within left and right cosets, if one element is within another, than they are equal. Thus cosets are equivalence classes and partition the group. If not equal, than disjoint.

Index: The index of H in G is the number of left cosets of H.

$$(G : H)$$

Can also be defined as the relative "size" of H in G, or, the number of cosets required to "fill up" G.

Basic Proofs

Theorem A3.3: If G is a group and $a, b \in G$, then $(a^{-1})^{-1} = a$ and $(ab)^{-1} = b^{-1}a^{-1}$.

Proof: Let G be a group and let $a \in G$. Since $a^{-1} \in G$ is the inverse of a, we have (1) $aa^{-1} = a^{-1}a = e$ where e is the identity of G. Because a satisfies equation (1), and the property of a unique inverse of a^{-1}, it follows that $(a^{-1})^{-1} = a$.

Let G be a group and let $a, b \in G$. Since $a, b \in G$, we have $ab \in G$ and $(ab)^{-1} \in G$. The element $(ab)^{-1}$ is the unique element $x \in G$ such that (2) $(ab)x = x(ab) = e$. We claim $x = b^{-1}a^{-1}$. Substituting $x = b^{-1}a^{-1}$ in the left hand side of equation (2), we find by associativity; definition of the inverse, and definition of the identity, that $(ab)x = (ab)(b^{-1}a^{-2}) = ((ab)b^{-1})a^{-1} = (a(bb^{-1}))a^{-1} = (ae)a^{-1} = aa^{-1} = e$.

Cycle Proofs

Theorem A3.4: For $n \geq 2$, S_n is generated by its transpositions.

Proof: This is clear for $n = 1$, and $n = 2$. For $n \geq 3$, we note $(1) = (12)^2$ and every cycle of *length* > 2 is a product of transpositions.

$$(\delta_1\delta_2 \ldots \delta_k) = (\delta_1\delta_2)(\delta_2\delta_3) \ldots (\delta_{k-1}\delta_k)$$

For example:

$$(13526) = (13)(35)(52)(26)$$

Since the cycles generate S_n, and products of transpositions give us all cycles, the transpositions generate S_n.

Theorem A3.5: For $n \geq 2$, S_n is generated by the $n - 1$ transpositions:

$$(12), (13), \ldots, (1n)$$

Proof 1: The theorem is obvious for $n = 2$, so we take $n \geq 3$. By theorem 2.4.1, it suffices to write any transposition in S_n as a product of transpositions involving a 1. For a transposition (ij), where i and j are not 1, check that:

$$(ij) = (1i)(1j)(1i)$$

Proof 2: We proceed by induction. The case where $n = 1$ or $n = 2$ is clear. Now suppose that S_n can be generated by transpositions of the for $(1i)$. Then each of the transpositions (ij) for $i, j \leq n$ are contained in the span of $(12), (13), \ldots, (1n + 1)$, so we need only show that we can generate $(i, n + 1)$ for each $1 < i < n + 1$. But we have

$$(1i)(1'n)(1i) = (i'n)$$

for any $i \in 2, \ldots, n$. This means we can generate all transpositions from the given set of transpositions, so S_{n+1} is generated by $(12), ..., (1'n)$. Having shown the inductive step, we are done.

Theorem A3.6: Every permutation σ of a finite set is a product of disjoint cycles.

Proof: Let $B_1, B_2, \ldots, B_r$ be the orbits of σ, and let μ_i be the cycle defined by:

$$\mu_i(x) = \begin{cases} \sigma(x), & \text{for } x \in B_i \\ x, & \text{otherwise} \end{cases}$$

Clearly $\sigma = \mu_1\mu_2 \ldots \mu_r$. Since the equivalence class orbits $B_1, B_2, \ldots, B_r$ (being distinct equivalence classes) are disjoint the cycles, $\mu_1\mu_2 \ldots \mu_r$ must be disjoint as well.

Coset Proofs

Theorem A3.7: If two cosets are not disjoint, they are identical.

$$aH = bH$$

Proof: Let $g \in aH \cap bH$, therefore $g = ah_1 = bh_2$ for $h_1, h_1 \in H$. Then $a = bh_2h_1^{-1}$. Now, let $ah \in aH$, then $ah = bh_2h_1^{-1}h$. The element on the right is in bH, since it is b times something in H. Therefore, $ah \in bH$, and $aH \subset bH$. By symmetry, $bH \subset aH$, so $aH = bH$.

Corollary: If g' is in gH, then gH is the same as $g'H$.

Theorem A3.8: **(Langrange's Theorem)** Let G be a finite group, and let H be a subgroup of G. Then

$$(G : H) = \frac{|G|}{|H|}$$

Therefore, the order of a subgroup divides the order of the group.

Proof: The costs of H partition G into $(G : H)$ pieces, and each piece contains $|H|$ elements. So the total number of elements in $(G : H)$ pieces is $(G : H) \cdot |H|$, but this is all of G:

$$(G : H) \cdot |H| = |G|$$

$$(G : H) = \frac{|G|}{|H|}$$

With $(G : H)$ being the collection of all cosets of H. Thus, the order of a subgroup divides the order of the group.

$(G : H)$ represented by $H * x_1, \ldots, H * x_k$, each piece has $|H|$ elements, so total number of elements in $|G| = (G : H) \cdot |H|$. Thus, total elements in $|G|$ must be divisible by $|H|$.

Theorem A3.9: Given $x, y \in G$, either $xH = yH$ or $(xH) \cap (yH) = \emptyset$. In otherwords, there is a perfect partitioning of G by cosets of H.

Proof: Suppose that $(xH) \cap (yH) \neq \emptyset$, let $z \in (xH) \cap (yH)$. Then $\exists\ h_1, h_2 \in H\ s.t.$ $z = xh_1 = yh_2$. An arbitrary element of xH has the form xh for some $h \in H$. If $\bar{h_1}$ is the inverse of h_1 in $(G, *)$ then we have

$$x = yh_2h_1^{-1} \quad with \quad xh \in xH$$

$$xh = yh_2h_1^{-1}h = yH$$

The element on the right is in yH, thus $xh \in yH$ and $xH \subset yH$. Symmetrically, $yH \subset xH$ can be proven. Thus, $xH = yH$.

Homomorphism Proofs

Theorem A3.10: **(Cayley's Theorem)** Let G be a group. G is isomorphic to a subgroup of S_G.

Proof: Define a one-to-one function $\phi : G \to S_G$, such that $\phi(xy) = \phi(x)\phi(y)$ for all $x, y \in G$. For $x \in G$, let $\lambda_x : G \to G$ be defined by $\lambda_x(g) = xg$ for all $g \in G$. The equation $\lambda_x(x^{-1}c) = x(x^{-1}c) = c$ for all $c \in G$ shows that λ_x maps G onto G. If $\lambda_x(a) = \lambda_x(b)$, then $xa = xb$ so $a = b$ by cancellation. Thus, λ_x is also one-to-one, and is a permutation of G. We now define $\phi : G \to S_G$ by defining $\phi(x) = \lambda_x$ for all $x \in G$.
To show that ϕ is one-to-one, suppose that $\phi(x) = \phi(y)$. Then, $\lambda x = \lambda y$ as functions mapping G into G. In particular $\lambda_x(e) = \lambda_y(e)$, so $x_e = y_e$ and $x = y$. Thus, ϕ is one-to-one. It only remains to show that $\phi(xy) = \phi(x)\phi(y)$, that is, that $\phi_{xy} = \lambda_x\lambda_y$. Now for any $g \in G$, we have $\lambda_{xy}(g) = (xy)g$. Permutation multiplication is function composition, so $(\lambda_x\lambda_y)(g) = \lambda_x(\lambda_y(g)) = \lambda_x(yg) = x(yg)$. Thus, by associativity, $\lambda_{xy} = \lambda_x\lambda_y$.

Theorem A3.11: If $ker(\phi) \leq G$, then $\phi : G \to K$ is a group homomorphism.

Proof: Want to show that $ker(\phi) \leq G$, $\{x \in G | \phi(x) = e_k\}$:

- $ker(\phi) \neq 0$
- closed under group operation
- closed under inverses

a) Note, $\phi(ea) = e_k$, so $e_G \in ker(\phi)$, thus $ker(\phi) \neq 0$.
b) Take any $x, y \in ker(\phi)$, thus $\phi(xy) = \phi(x)\phi(y) = e_ke_k = e_k$, so $xy \in ker(\phi)$.
c) Take any $x \in ker(\phi)$. Then $\phi(x^{-1}) \to (\phi(x))^{-1} = (e_k)^{-1} = e_k$, so $x^{-1} \in ker(\phi)$.
Thus, $ker(\phi)$ is a subgroup of G.

Theorem A3.12: If $\phi : G \to K$ is a group homomorphism $\to ker(\phi) \triangleleft G$.

Proof: $\forall g \in G, \forall h \in ker(\phi)$

$$\begin{aligned}\phi(ghg^{-1}) &= \phi(g)\phi(h)\phi(g)^{-1} \\ &= \phi(g)e_k(\phi(g))^{-1} \\ &= \phi(g)(\phi(g))^{-1} \\ &= e_k \qquad thus,\ \ ghg^{-1} \in ker(\phi)\end{aligned}$$

Theorem A3.13: Let $f : G \to H$ be a homomorphism from a group, $(G, *)$ to a group $(H, *)$ then,

$$\frac{G}{ker(f)} \cong Im(f)$$

Proof: We first define an isomorphism $\theta : \frac{G}{ker(f)} \to Im(f)$. One way to do this is:

$$\theta(g \cdot ker(f)) := f(g)$$

First, we check that θ is well-defined, that is, $\theta(g \cdot ker(f))$ does not depend on the choice, g, of a coset representative from the coset $g \cdot ker(f)$. Suppose that $g' \in g \cdot ker(f)$. Then, $g' = g \cdot x$ for some $x \in ker(f)$, so

$$f(g') = f(g \cdot x) = f(g)f(x) = f(g) \cdot e_H = f(g)$$

Hence, $\theta(g \cdot ker(f)) = f(g)$ is well-defined.
Second, we check that θ is a homomorphism:

$$\theta((g{*}ker(f))\dagger(h{*}ker(f)) = \theta((g{*}h)ker(f) = f(g{*}h) = f(g)f(h) = \theta(g{*}ker(f))\theta(h{*}ker(f))$$

Next, we check that θ is injective, suppose that $\theta(g * ker(f)) = \theta(h * ker(f))$. Thus, $f(g) = f(h)$. Then, $f(\bar{g} * h) = e_H$. Where $\bar{g}$ is the inverse of g in $(G, *)$ and e_H is the identity element in $(H, \dagger)$. Then, $x := \bar{g} * h \in ker(f)$, so $h = g * x \in g * ker(f)$. Hence, $h * ker(f) = g * ker(f)$, so θ is injective.
Finally, to see that θ is surjective, if $a \in Im(f)$, then there is an element $g \in G$ with $a = f(g) = \theta(g * ker(f))$.

Homomorphism Definitions:

Kernal: Measure of how "not" one-to-one a group homomorphism is. Kernal contains all elements mapped to identity such that kernal forms a subgroup of homomorphism. $x_1, x_2 \in G \to y$

$$\begin{aligned}f(x_1) = y \to f(x_1)f(x_1^{-1}) &= yy^{-1} = I_H \\ f(x_2) = y \to f(x_2)f(x_1^{-1}) &= yy^{-1} = I_H \\ \to f(x_1 * x_1^{-1}) &= I_H \\ \to f(x_2 * x_1^{-1}) &= I_H\end{aligned}$$

$$ker(f) = \{x \in G | f(x) = I_H\}$$

If f is not 1-1, then ker(f) has more than one element:

$$f(I_G) = I_H \rightarrow I_G \in ker(f)$$

If homomorphism, f, not an injection $\rightarrow$ more than one element maps to the identity. If $ker(f) = I_G \rightarrow$ f is 1-1.

Homomorphism: Mapping measuring structural similarities between groups. Has multiplicity property: $f(x) * f(y) = f(z) \rightarrow f(x)f(y) = f(x * y)$

Isomorphism: When two groups are identical. If homomorphism, f, is both injective and surjective.

Quotient Group: A Group divided by the kernal of the Group.

Relation: **(Equivalence)** R on a set S is one that satisfies three properties:

Reflexive: xRx

Symmetric: xRy

Transitive: If xRy, yRz, then xRz

An equivalence relation breaks a set into conjugacy classes where all elements within the same class are similar.