McMaster University
DigitalCommons@McMaster

Open Access Dissertations and Theses | Open Dissertations and Theses

10-1-2012

The (Nested) Word Problem: Formal Languages, Group Theory, and Languages of Nested Words

Christopher S. Henry
McMaster University, henrycs@math.mcmaster.ca

Follow this and additional works at: http://digitalcommons.mcmaster.ca/opendissertations

Part of the Algebra Commons, Geometry and Topology Commons, and the Theory and Algorithms Commons

Recommended Citation
Henry, Christopher S., "The (Nested) Word Problem: Formal Languages, Group Theory, and Languages of Nested Words" (2012). *Open Access Dissertations and Theses.* Paper 7298.

This Thesis is brought to you for free and open access by the Open Dissertations and Theses at DigitalCommons@McMaster. It has been accepted for inclusion in Open Access Dissertations and Theses by an authorized administrator of DigitalCommons@McMaster. For more information, please contact scom@mcmaster.ca.

THE (NESTED) WORD PROBLEM

THE (NESTED) WORD PROBLEM: FORMAL LANGUAGES, GROUP THEORY, AND LANGUAGES OF NESTED WORDS

By

Chris Henry, B.Sc.

A Thesis
Submitted to the School of Graduate Studies
in Partial Fulfilment of the Requirements
for the Degree of

MASTER OF SCIENCE

© Copyright by Chris Henry, 2012

DEGREE:	MASTER OF SCIENCE, 2012
UNIVERSITY:	McMaster University, Hamilton, Ontario
DEPARTMENT:	Mathematics and Statistics
TITLE:	The (Nested) Word Problem: Formal Languages, Group Theory, and Languages of Nested Words
AUTHOR:	Chris Henry, B.Sc.(McMaster University)
SUPERVISOR(S):	Dr. Hans U. Boden
PAGES:	viii, 72

MCMASTER UNIVERSITY
DEPARTMENT OF
MATHEMATICS AND STATISTICS

The undersigned hereby certify that they have read and recommend to the Faculty of Graduate Studies for acceptance of a thesis entitled "**The (Nested) Word Problem: Formal Languages, Group Theory, and Languages of Nested Words**" by **Chris Henry** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated: July 2012

Research Supervisor: _____
Dr. Hans U. Boden

Examing Committee: _____
Dr. Andrew J. Nicas

_____
Dr. Matthew Valeriote

To Michelle

Table of Contents

1 Introduction **3**

2 Formal Languages **7**

2.1 Regular Languages 8

2.2 Context-free Languages 12

3 Group Theory and Formal Languages **17**

3.1 Finitely Generated Groups and Presentations 18

3.2 Cayley Graphs and Group Geometry 20

4 Word Problem **23**

4.1 Regular Languages and the Word Problem 24

4.2 Context-free Languages and the Word Problem 26

5 Automatic Groups **33**

5.1 General Properties 33

5.2 Solving the Word Problem 40

5.3 Efficiency of the Solution 44

6 Generalizations of Automatic Structures **47**

6.1 Parallel-poly Pushdown Groups 48

6.2 Combings 51

7 Nested Words and Group Theory **55**

7.1 Theory of VPLs 56

7.2 VPLs and the Word Problem 60

7.3 Conclusion 68

Chapter 1

Introduction

This thesis concerns itself with drawing out some interesting connections between the fields of group theory and formal language theory. The motivating idea is a simple one. Given a group G with a finite set of generators $X \subset G$, it is natural to consider the set $A = X \cup X^{-1}$ of generators and their inverses as an alphabet. Within the set A^* of all finite length words over the alphabet, we can then consider particular collections of words $L \subset A^*$, such that every group element has at least one representative in L. Formal language theory calls these collections of words *languages*, and gives tools for recognizing different classes of languages based on their complexity. Our approach is to examine what the structure of L can tell us about group theoretic properties, both algebraic and geometric.

The second chapter provides the required background in formal language theory, covering two classes of languages that will be important for the topics discussed: regular and context-free languages. Chapter 3 makes mathematically precise what is meant by saying that formal language theory is connected with group theory. Chapters 4 to 6 then explore the implications of this framework. In chapter 4 we reformulate the classic word problem of Dehn as a language theoretic one; solving the word problem is equiva-

lent to determining whether a given word lies in the language of all words that represent the identity element. When this language is either regular or context-free, we are able to characterize the group algebraically . In the context-free case, this characterization is established by using the structure of words in the language to prove a geometric condition on the Cayley graph of the group.

Chapter 5 uses the framework to present some standard results from automatic group theory. The definition of an automatic structure can be presented in terms of a collection of regular languages. This translates to an equivalent geometric condition involving distances between paths in the Cayley graph. Regular languages are considered the most basic in terms of their complexity, and the implication of this for automatic groups is that we are able to efficiently perform computations such as solving the word problem. Finally, chapter 6 addresses the natural impulse to generalize automatic groups by using context-free languages in place of regular ones. The resulting class of groups encompasses the fundamental groups of all compact 3-manifolds, and the increased complexity associated with context-free languages gives rise to some important distinctions from automatic groups.

The final chapter attempts to incorporate some recent developments in formal language theory into parts of the research surveyed here. In [2], Alur and Madhusudan introduce the concept of a nested word. Nested words and their associated languages are similar to the traditional word languages considered in formal language theory, however with each word we attach additional information called a nested structure. The resulting languages (called visibly pushdown, or regular languages of nested words) share a subtle relationship with both regular and context-free languages; they have the same closure properties as regular languages, but in other ways are closer to context-free languages.

We focus on showing how regular languages of nested words provide a natural model for studying the word problem. To do so we first establish an appropriate definition for a group having a visibly pushdown word problem, and show that the free group F_n satisfies this property. Our main result, Theorem 7.23, extends this to the semi-direct product $F_n \rtimes_\psi S_n$ of a free group with a symmetric group. The proof relies on an operation that takes as inputs a visibly pushdown language and a regular language, and outputs another visibly pushdown language. This operation essentially captures the interaction in $F_n \rtimes_\psi S_n$ between words representing the identity in F_n and S_n respectively.

Chapter 2

Formal Languages

Let A be a finite set, which we will call an alphabet. For $n \in \mathbb{N}$, let $A^n = \{w \mid w : \{1, 2, \ldots, n\} \to A \text{ is a function }\}$. We refer to an element $w \in A^n$ as a word of length $|w| = n$ and we write $w = a_1 \ldots a_n$, where $w(i) = a_i$ and $a_i \in A$. For $1 \leq i < n$ we let $w[i] = a_1 \ldots a_i$ be the prefix of w of length i, taking $w[i] = w$ for $i \geq n$. We denote as ϵ the unique element $\epsilon : \emptyset \to A$ of A^0, which is called the empty word. Finally, we let $A^* = \bigcup_{n=0}^{\infty} A^n$, which is the set of all finite words over the alphabet A.

Definition 2.1. Given an alphabet A, a language of words over A is a subset $L \subset A^*$.

Given a language $L \subset A^*$, for L to be a meaningful collection of words we would like to possess an algorithm that would tell us which words $w \in A^*$ are contained in L. In fact, one way to define different classes of formal languages is precisely by the type of algorithm which allows us to recognize words in the language, and this is the approach we take here. We formalize the concept of algorithm by defining machines, which we think of as reading in words and deciding whether or not they belong to the language. Notation is borrowed mostly from [8].

2.1 Regular Languages

Definition 2.2. A deterministic finite state automaton (FSA) is a five-tuple $M = (A, S, s_0, Y, \delta)$ where:

- A is an alphabet
- S is a finite set of states
- $s_0 \in S$ is the start state
- $Y \subset S$ is the set of accept states
- $\delta : S \times A \to S$ is the transition function

We picture the FSA as a machine that reads in a word $w = a_1 \dots a_n \in A^*$ from left to right, one letter at a time. Each time a letter is read, the machine transitions from its present state s_i to a new state s_{i+1}, based on s_i and the letter read in a_{i+1}; the function δ describes this transition. If after reading the entire word w the machine is in state $s_n \in Y$, then we say that M accepts w. We let $L(M) = \{w \in A^* \mid w \text{ is accepted by } M\}$.

Definition 2.3. We say that a language of words $L \subset A^*$ is regular, if there exists a FSA M such that $L = L(M)$.

Example 2.4. Consider the alphabet $A = \{a, b\}$ and the language of words given by $L = \{a^m b^n \mid m, n = 0, 1, \dots\}$, where $a^n = \overbrace{a \cdots a}^{\text{n times}}$ and $a^0 = b^0 = \epsilon$. The figure below depicts a FSA recognizing this language which shows it to be regular. One thing to note is that the machine has no memory; it reads in a number of a's (possibly none) followed by a number of b's (possibly none), but cannot keep track of how many letters it has read in.

Now consider the closely related language that we will see is not regular, $L = \{a^n b^n \mid n = 0, 1, \dots\}$, where the number of a's and b's are required to be equal. The fact that this language is not regular is a consequence of the pumping lemma, which gives an important property of regular languages.

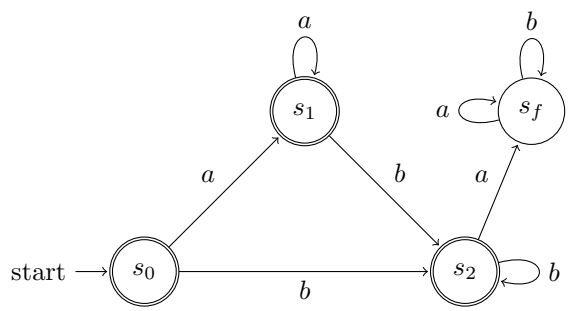


Figure 2.1: A FSA recognizing the language of Example 2.4. The set of states is $S = \{s_0, s_1, s_2, s_f\}$ depicted by nodes. The double lines indicate the subset of accept states, and transitions are depicted by labelled arrows.

Lemma 2.5 (Pumping Lemma)**.** *Let A be an alphabet and $L \subset A^*$ a regular language. There exists an $N \in \mathbb{N}$ such that for all $w \in L$ satisfying $|w| \geq N$, we can write $w = xyz$, with $x \neq \epsilon$ or $z \neq \epsilon$, and $xy^nz \in L$ for all $n \in \mathbb{N}$.*

Proof. Let $M = (A, S, s_0, Y, \delta)$ be a FSA such that $L = L(M)$, and let N be greater that the number of states, i.e. the number of elements of S. We consider a word $w = a_1 \cdots a_n \in L$ with $n \geq N$. Since $w \in L$, M begins in s_0 and undergoes transitions corresponding to the sequence of states $s_1, \ldots, s_n$ with $s_n \in Y$. Since $n \geq N$, M must have visited the same state twice. More precisely, $\exists(i, j)$ with $i \leq j$ such that $s_i = s_j$. We may repeat the subword $a_{i+1} \cdots a_j$ corresponding to these transitions an arbitrary number of times, and reading off the rest of the word will still put us in the accept state $s_n \in Y$. $\square$

Example 2.6. We use the pumping lemma to show that $L = \{a^nb^n \mid n = 0, 1, \ldots\}$ is not a regular language. For any non-trivial word $w \in L$, there are two cases:

- Case 1: The subword contains both a's and b's. In this case, repeating the subword any number of times results in a word where the a's do not appear only at the beginning of the word. For example, consider $w = aaabbb \in L$ and the subword ab. Repeating the subword one time, we have $w' = aaababbbb \notin L$.
- Case 2: The subword contains only a's or only b's. Without loss of generality, assume the subword contains only a's. Then repeating the subword any number of times results in a word where the number of a's and b's are not equal.

Remark 2.7. We could have defined a regular language in other ways, for instance via the type of logical expression that generates the language, or the grammar that generates the language. Most of the connections with group theory use machine related definitions, however the paper by Muller and Schupp [13] suggests that a further direction of research would be to consider more fully the implications of alternate ways of defining formal language classes.

Remark 2.8. To simplify proofs, we will also use non-deterministic FSA. The difference is that the transition function is of the form

$$\delta : S \times (A \cup \{\epsilon\}) \to \mathscr{P}(S)$$

where $\mathscr{P}(S)$ is the power set of S. The interpretation is that when the machine is currently in state s and reads in a symbol $a \in A$, it has the ability to transition to any one of the states $s' \in \delta(s, a)$. The symbol ϵ has the interpretation of the empty string, and this allows us to make transitions without reading a letter. A word $w = a_1 \dots a_n$ is accepted by a non-deterministic FSA M if it is possible to transition from the start state to an accept state by reading in the word from left to right, possibly allowing for ϵ-transitions. For regular languages it turns out that the distinction between deterministic FSA and non-deterministic FSA does not matter; for any language accepted by a non-deterministic FSA, there exists a deterministic FSA that accepts

it as well (Theorem 2.1 in [11]). Being able to work with non-deterministic FSA allows us to easily construct new machines from given ones, and we will employ this strategy in proving some closure properties of regular languages directly below.

Regular languages have many desirable closure properties. For any two languages L_1 and L_2, we consider the concatenation language $L_1L_2 = \{w_1w_2 \mid w_1 \in L_1 \text{ and } w_2 \in L_2\}$. Furthermore, given a language L we recursively define L^n by setting $L^0 = \{\epsilon\}$, and $L^n = L(L^{n-1})$. The Kleene star closure L^* of L is defined as $L^* = \bigcup_{n=0}^{\infty} L^n$. Finally, for a word $w = a_1 \dots a_n \in L$, we denote the reversal of the word as $w^R = a_n \dots a_1$, and we have $L^R = \{w^R \mid w \in L\}$. We have:

Theorem 2.9. *Let L, L_1, and L_2 be regular languages over the alphabet A. The following languages are also regular:*

1. $L_1 \cap L_2$
2. $L_1 \cup L_2$
3. L_1L_2
4. L^*
5. L^R
6. $L_{comp} = A^* \setminus L$
7. $L_{pre} = \{w \mid wu = v \in L\}$, *the prefix closure of* L

Proof. Assume $L = L(M)$ where $M = (A, S, s_0, Y, \delta)$, and $L_i = L(M_i)$ where $M_i = (A, S_i, s_0^i, Y_i, \delta_i)$, $i = 1, 2$.

1. To construct a machine accepting $L_1 \cap L_2$, we take the set of states to be $S_1 \times S_2$. We define a transition function by $\Delta((s_1, s_2), a) = (\delta_1(s_1, a), \delta_2(s_2, a))$. The initial state is (s_0^1, s_0^2), and accept states are of the form $\{(y_1, y_2) \mid y_i \in Y_i\}$.
2. We use the same machine as above, but the accept states are now $\{(y_1, s_2) \mid y_1 \in Y_1, s_2 \in S_2\} \cup \{(s_1, y_2) \mid s_1 \in S_1, y_2 \in Y_2\}$.

3. To construct a machine accepting L_1L_2 we think of connecting M_1 and M_2 together with ϵ-transitions, which will result in a non-deterministic FSA. We let s_0^1 be the only initial state, and create an ϵ-transition from each $y_1 \in Y_1$ to the intial state s_0^2 of M_2. This allows us to recognize words in $L(M_2)$ after first reading in a word in $L(M_1)$.
4. We construct a non-deterministic FSA M^* from M to accept L^*. The accept states are the same as M, except if the inital state satisfies $s_0 \notin Y$ then we include it as an accept state of M^*, which allows us to recognize the empty word. We create an ϵ-transition from each $y \in Y$ back to s_0, which allows us to reset the machine after reading in a word in L.
5. We construct M^R from M to accept L^R. The machine M^R has the same set of states as M, however we identify all of the states $Y \subset S$ as a single state which becomes the initial state. We reverse all of the transitions, so that if $\delta(s, a) = s'$ in M then the new transition function δ^R is defined by $\delta(s', a) = s$. By identifying all the states in Y as a single state, this makes M^R non-deterministic. Finally s_0 is now designated as the single accept state of M^R.
6. We construct M_{comp} to accept the language L_{comp}. M_{comp} is the same as M, except we take the set of accept states to be $Y_{comp} = S \setminus Y$.
7. Similarly, M_{pre} is the same as M, but we take the accept states to be $Y' = \{s \in S \mid \exists \text{a sequence } s = s_0, \ldots, s_n \text{ where } s_i = \delta(s_{i-1}, a_{i-1}) \text{ for some } a_{i-1} \in A \text{ and } s_n \in Y\}$.

$\square$

2.2 Context-free Languages

We now turn to context-free languages (CFLs). As mentioned above, the obvious deficiency with FSA is that they have no memory; or more precisely they have a finite amount of memory, with the number of states being the

effective constraint. Context free languages are recognized by a machine that has a finite number of states, but also a first-in-last-out memory stack. This will enable us to recognize languages that are not regular, as in Example 2.6. This language class includes regular languages, and is therefore often said to be more expressive. In some sense however, context free languages are not as nice of a class of languages as regular ones, in that they do not satisfy as many closure properties. Let us make these concepts more precise.

Definition 2.10. A deterministic pushdown automaton (PDA) M is a seven-tuple $(A, S, \Gamma, s_0, \gamma_0, Y, \delta)$ where:

- A is an alphabet
- S is a finite set of states
- Γ is a finite stack alphabet
- $s_0 \in S$ is the start state, and $\gamma_0 \in \Gamma$ is the bottom of stack symbol
- $Y \subset S$ is the set of accept states
- $\delta : S \times (A \cup \{\epsilon\}) \times \Gamma \to S \times \Gamma^*$ is the transition function, where a triple (s, ϵ, γ) being in the domain of δ implies (s, a, γ) is not defined for all $a \in A$

To make transitions, the machine needs to know the current state s_i, the input symbol being read in a, and the current top of stack symbol γ. It transitions to a new state s_{i+1}, and replaces γ on the top of the stack by a finite word $\chi = \gamma_1 \dots \gamma_n \in \Gamma^*$. We think of χ as being added to the stack one letter at a time, starting with γ_1 and ending with γ_n, which becomes the new top of stack symbol. We interpret $\delta(s, \epsilon, \gamma)$ as the machine performing a stack operation without having to read an input symbol.

Definition 2.11. Given a deterministic PDA $M = (A, S, \Gamma, s_0, \gamma_0, Y, \delta)$, we define an instantaneous description (ID) of M to be a triple (s, w, χ) where $s \in S$, $w \in A^*$ and $\chi \in \Gamma^*$. We think of w as the input word that we would

like the machine to process, and χ as the accumulated stack contents. For $a \in (A \cup \{\epsilon\})$, we write $(s, aw, \chi\gamma) \vdash_M (t, w, \chi\chi')$ if $\delta(s, a, \gamma) = (t, \chi')$, and denote $\vdash_M^*$ as the reflexive and transitive closure of $\vdash_M$. The language of words accepted by final state is defined as

$$L_{fs}(M) = \{w \in A^* \mid (s_0, w, \gamma_0) \vdash_M^* (y, \epsilon, \chi) \text{ for some } y \in Y \text{ and } \chi \in \Gamma^*\}$$

The language of words accepted by empty stack is defined as

$$L_{es}(M) = \{w \in A^* \mid (s_0, w, \gamma_0) \vdash_M^* (t, \epsilon, \epsilon) \text{ for some } t \in S\}$$

Theorem 2.12 (5.1 and 5.2 in [11])**.** *Let A be a finite alphabet and $L \subset A^*$ a language. There exists a PDA M_1 such that $L = L_{fs}(M_1)$ if and only if there exists a PDA M_2 such that $L = L_{es}(M_2)$.*

In this sense, acceptance by final state and empty stack are equivalent, and we may use whichever definition is more convenient. In the example below we construct a PDA which accepts words by empty stack, and this is more natural because the stack is acting as a counting mechanism. Similar to regular languages, we then define context-free languages as those accepted by PDA. For a PDA M we just write

$$L(M) = \{w \in A^* \mid w \text{ is accepted by } M\},$$

where we assume the acceptance is either by final state or empty stack, but suppress the notation.

Definition 2.13. A language $L \subset A^*$ is deterministic context-free if there is a PDA M such that $L = L(M)$.

Example 2.14. Consider the alphabet $A = \{a, b\}$ and the language of words given by $L = \{a^n b a^n \mid n = 0, 1, \dots\}$. We take the stack alphabet to be $\Gamma = \{0, 1\}$ with $\gamma_0 = 0$, and the set of states to be $S = \{s_0, s_1, s_2\}$. The transition function is given as:

$$\delta(s,x,\gamma) = \begin{cases} (s_0,a,0) \mapsto (s_1,01) & \\ (s_0,a,1) \mapsto (s_1,11) & \\ (s_0,b,\gamma) \mapsto (s_1,\gamma) & \gamma \in \{0,1\} \\ (s_1,a,1) \mapsto (s_1,\epsilon) & \\ (s_1,\epsilon,0) \mapsto (s_1,\epsilon) & \\ (s,x,\gamma) \mapsto (s_2,\gamma) & \text{in all other cases} \end{cases}$$

The interpretation of this machine is very intuitive. Starting in state s_0, we can read in any number of a's, and the stack keeps track of how many are read in by writing a 1 to the top of the stack each time. Once the letter b is read in, the machine transitions to state s_1. In this state, the machine will erase a 1 from the top of the stack every time an a is read in. Finally, if the number of a's read in before the b is equal to the number of a's after, the top of the stack will be 0 and the machine makes an ϵ transition to empty the stack. We eliminate unwanted transitions by including a third state s_2, in which the machine will never leave once entered. For example, if we read in more than one b, the machine transitions to s_2 and from then on neither adds nor subtracts from the stack. The machine accepts words by empty stack.

Remark 2.15. Here we present the theory for deterministic PDA, but it is important to note the distinction with non-deterministic PDA and their associated languages. The languages generated by non-deterministic PDA contain deterministic context-free languages as a strict subclass, and possess different closure properties. In a non-deterministic PDA the transition function δ is a mapping to finite subsets of $S \times \Gamma^*$. In addition, the last condition of Definition 2.10 says the machine is deterministic in the sense that there is no ambiguity with respect to ϵ-transitions; the machine will never read in an input symbol if it can make an ϵ-transition instead. Non-deterministic PDA do not have this restriction, and in these cases we think of the machine as guessing whether to read more input or make an ϵ-transition. There are languages recognized by non-deterministic PDA which are not recog-

nized by any deterministic PDA. An example is the palindrome language given by $L_{pal} = \{ww^R \mid w \in A^*\}$, where A is a fixed finite alphabet. A non-deterministic PDA can recognize this language by making guesses about when it is halfway through reading a word, which would allow it to match the second half with the reversal of the first. However no deterministic PDA is able to recognize this language [11].

We now review the closure properties of deterministic context free languages.

Theorem 2.16 ([11])**.** *Let L be a context-free language over the alphabet A. The following languages are also context-free:*

1. $L_{comp} = A^* \setminus L$
2. $L_{pre} = \{w \mid wu = v \in L\}$, *the prefix closure of L*

However, context-free language are not closed under union, intersection, concatenation, Kleene-star closure or reversal.

Example 2.17 (6.1 in [11]). Just as $L = \{a^n b^n \mid n = 0, 1, \dots\}$ is not a regular language because a FSA does not have the memory capacity to recognize it, it turns out that $L_1 = \{a^n b^n c^n \mid n = 0, 1, \dots\}$ is not a context-free language. This can be demonstrated by proving an analogous pumping lemma for context-free languages, and using an argument similar to the above example. However, Example 2.14 suggests that we can construct a PDA recognizing L, and this extends easily to showing that $L_2 = \{a^n b^n c^m \mid m, n = 0, 1, \dots\}$ and $L_3 = \{a^n b^m c^m \mid m, n = 0, 1, \dots\}$ are both context-free. We then have $L_1 = L_2 \cap L_3$, demonstrating that CFLs are not closed under intersection.

Chapter 3

Group Theory and Formal Languages

We now make explicit the sense in which formal language theory is connected with group theory. Given an alphabet A and a map $p : A \to G$, we extend p in a natural way to $\pi : A^* \to G$. The map π is a semi-group homomorphism where concatenation corresponds to group multiplication in G. If π is surjective we say that A is a set of semi-group generators for G. This means that every group element is represented by at least one word over the alphabet A. Fixing an alphabet A we let $\mathscr{L}$ be a class of languages over the alphabet, and $L \in \mathscr{L}$ be a specific language over A. For example, we denote by $\mathscr{L}_{reg}$ the set of all regular languages over A, where notation of the alphabet is supressed. From Example 2.4 we know that $L = \{a^m b^n \mid n = 0, 1, \dots\} \in \mathscr{L}_{reg}$ is a particular regular language.

Definition 3.1. Given $\mathscr{L}$, we say that a group G is $\mathscr{L}$-generated if there exists an $L \in \mathscr{L}$ such that for some $p : A \to G$, the natural map described above $\pi : A^* \to G$ is surjective when restricted to L. We will write $\pi : L \to G$ when referring to the restriction of the natural map to L.

Example 3.2. Consider the group $G = \mathbb{Z} \oplus \mathbb{Z}$. Each copy of $\mathbb{Z}$ is generated by the element 1, and $\{(1,0),(0,1)\}$ is therefore a set of generators. We take the alphabet to be $A = \{a, b, a^{-1}, b^{-1}\}$ and obtain the map π by extending $p : A \to G$ given by

$$\begin{cases} a \mapsto (1,0) \\ b \mapsto (0,1) \\ a^{-1} \mapsto (-1,0) \\ b^{-1} \mapsto (0,-1) \end{cases}$$

From Example 2.4, we know that $L_1 = \{a^m b^n \mid m, n = 0, 1, \dots\}$ is a regular language. Using the notation $a^{-n} = \overbrace{a^{-1} \cdots a^{-1}}^{\text{n times}}$, it is straightforward to generalize this example to show that $L = \{a^m b^n \mid m, n = 0, \pm 1, \pm 2, \dots\}$ is also regular. Since $\pi : L \to G$ is a surjection, G is $\mathscr{L}_{reg}$-generated. This example illustrates a particularly nice case in that π is actually a bijection. In this case, we think of each element of G as having a unique normal form representative in our language.

3.1 Finitely Generated Groups and Presentations

We use this framework to introduce the notion of a finitely generated group. Starting with the alphabet X, we construct a disjoint set X^{-1} of formal inverses of elements of X. This means we have an involution $\iota : X \to X^{-1}$, and write $x^{-1} = \iota(x)$ for the formal inverse of x. The free group on X is denoted by $F(X)$. The elements of $F(X)$ are reduced words over the extended alphabet $A = X \cup X^{-1}$ of generators and their inverses. A word is reduced if it contains no trivial relations, which are relations of the form xx^{-1} or $x^{-1}x$. Group multiplication is concatenation of words, where we reduce the resulting word

if necessary by deleting trivial relations. For a word $w = a_1 \dots a_n \in A^*$, the inverse is given by $w^{-1} = a_n^{-1} \dots a_1^{-1}$ and the empty string ϵ is the identity element.

We consider the situation where $A = X \cup X^{-1}$ is a set of semi-group generators for G, and the map p satisfies $p(x^{-1}) = (p(x))^{-1}$ as elements of G. The set X is then called a set of generators for G. By extending p to $\pi : F(X) \to G$ we get a group homomorphism that is surjective onto G. We use the letter e to denote the identity element of G, so we have $\ker \pi = \{w \in F(X) \mid \pi(w) = e\}$. A set of relations for G is any subset $R \subset F(X)$ such that $\ker \pi = \langle R \rangle$, where $\langle R \rangle$ denotes the normal subgroup of $F(X)$ generated by R. Together with the generators we write $G = \langle X \mid R \rangle$, and say that G is finitely generated if X can be chosen finite; if in addition R can be chosen to be finite then we say G is finitely presented.

Example 3.3. The group $\mathbb{Z} \oplus \mathbb{Z}$ can be generated by $X = \{a, b\}$ as shown above. A presentation is given by $\langle a, b \mid aba^{-1}b^{-1} \rangle$. Since in the group we have that $\pi(aba^{-1}b^{-1}) = e$, we could also write the presentation as $\langle a, b \mid ab = ba \rangle$. By definition words in R map to the identity, but we might choose to write them as an equation in the group if it is more intuitive. In this case, the equation $ab = ba$ says that the generators commute, so the second presentation seems more natural.

Remark 3.4. For words $w_1, w_2 \in A^*$ it is important to be aware of the distinction between equality as words (or equality as reduced words in $F(X)$), and equality in the group. We may have that w_1 and w_2 are not equal as words (reduced words), but that $\pi(w_1) = \pi(w_2)$ in G. To emphasize this we write $\overline{w}$ for the image of a word under π.

3.2 Cayley Graphs and Group Geometry

Finally, we introduce the Cayley graph of a group, which allows us to view groups as geometric objects, the motivating idea of geometric group theory. Given a finitely generated group $G = \langle X \mid R \rangle$, we construct the Cayley graph $\Gamma_A(G)$ as follows. The vertices of $\Gamma_A(G)$ are the elements of G, and the origin of the graph is the identity element $e \in G$. Directed edges are labelled by elements $a \in A = X \cup X^{-1}$, and we write (g_1, a, g_2) for the edge from g_1 to g_2 labelled by a; such an edge exits if $g_1\overline{a} = g_2$ in G. A path in $\Gamma_A(G)$ is a sequence of edges (g_{i-1}, a_i, g_i) for $i = 1, \ldots, n$. When $g_0 = e$ is the identity we generally suppress the g_i notation, and just write $a_1 \ldots a_n$. Since this is just a word w over $A = X \cup X^{-1}$, we write $\widehat{w} = a_1 \ldots a_n$ for the corresponding path in the Cayley graph. As with words, the prefix of length i is the path $\widehat{w}[i] = a_1 \ldots a_i$ for $1 \leq i < n$, and $\widehat{w}[i] = \widehat{w}$ for $i \geq n$. Finally for $i = 0$ the prefix is the empty word ϵ, with the corresponding path being the one that stays at the origin.

For any two elements $g, h \in G$ we can consider the word difference $g^{-1}h$. Since X generates G, there is some word $w = a_1 \ldots a_n \in A^*$ such that $\overline{w} = g^{-1}h$, and the corresponding path $\widehat{w}$ connects g to h in the Cayley graph. Letting $|w|$ be the length of a word over A, we then have a natural metric on G called the word metric, given by

$$d_{G,A}(g, h) = \min\{|w| \mid \widehat{w} \text{ is a path from } g \text{ to } h\}.$$

We may then consider the Cayley graph $\Gamma_A(G)$ as a metric space equipped with the word metric $d_{G,A}$, whose elements are the vertices (element of G). It is clear that both the Cayley graph and the induced word metric depend on the particular choice of generating set. To study groups from a geometric perspective, we look for properties that are invariant under a change of

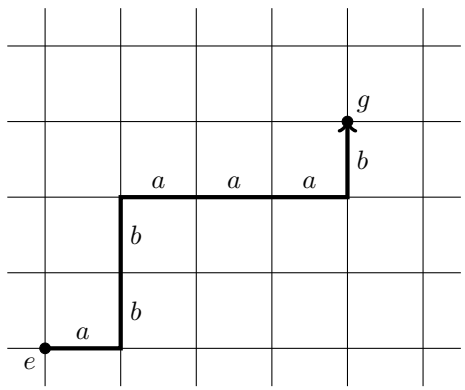


Figure 3.1: The path $\widehat{w} = abbaaab$ from the identity to the element $g = (4, 3)$, depicted in a portion of the Cayley graph for $\mathbb{Z} \oplus \mathbb{Z}$.

generators, which in turn has implications for the Cayley graphs as metric spaces.

Definition 3.5. Given two metric spaces (M_1, d_1) and (M_2, d_2), a map $f : M_1 \to M_2$ is a quasi-isometry if there exists constants $D \geq 1$ and $E, F \geq 0$ such that for all $x, y \in M_1$

$$\frac{1}{D} d_1(x, y) - E \leq d_2(f(x), f(y)) \leq D d_1(x, y) + E$$

and for all $z \in M_2$ there exists an $x \in M_1$ such that $d_2(z, f(x)) \leq F$

For two generating sets A_1 and A_2, the identity map on G induces a quasi-isometry between the Cayley graphs $\Gamma_{A_1}(G)$ and $\Gamma_{A_2}(G)$ as metric spaces. The appropriate bounds are $E = F = 0$, and $D = \max_{a \in A_2}\{d_{G,A_1}(a, e)\}$. This provides justification for calling a group property geometric if it is invariant under a change of generators. Geometric properties will arise frequently in the context of applications involving formal languages, as we will see below.

Chapter 4

Word Problem

It was Max Dehn who had the prescience to formulate the three defining problems in combinatorial group theory that still drive research to this day: the word, conjugacy, and isomorphism problems. We focus on the word problem. Given a finitely generated group $G = \langle X \mid R \rangle$ and two words $w_1, w_2 \in A^*$, the word problem asks for an algorithm that checks whether $\overline{w_1} = \overline{w_2}$ in G. By taking $w = w_1 w_2^{-1}$ this is equivalent to asking for an algorithm to tell whether a given word $w \in A^*$ is equal to the identity in G. This corresponds nicely to our machine-theoretic definitions of different formal language classes.

Definition 4.1. Let $G = \langle X \mid R \rangle$ be a presentation for a finitely generated group G, and take $A = X \cup X^{-1}$ as before. Given a formal language class $\mathscr{L}$, we say a group has an $\mathscr{L}$-word problem (with respect to A) if there exists an $L \in \mathscr{L}$ such that

$$L = W_A(G) = \{w \in A^* \mid \overline{w} = 1\}$$

4.1 Regular Languages and the Word Problem

Working through the language hierarchy, it is natural to first ask what can be said of groups that have an $\mathscr{L}_{reg}$-word problem, or regular word problem. Before addressing this, we establish that the property of having a regular word problem is geometric.

Theorem 4.2. *Let $G = \langle X_1 \mid R_1 \rangle$ and $G = \langle X_2 \mid R_2 \rangle$ be two presentations for G. Then G has a regular word problem with respect to A_1 if and only if G has a regular word problem with respect to A_2.*

Proof. Let M_1 be the FSA such that $L(M_1) = W(G_{A_1})$. Since A_1 is a set of semi-group generators for G, we have for all $a \in A_2$ that $\overline{a} = \overline{a_{1,1} \dots a_{1,k}}$, with $a_{1,i} \in A_1$. We construct a FSA M_2 from M_1 by taking the same initial state s_0, set of states S, and accept states $Y \subset S$. For all $a \in A_2$ and $s, t \in S$, M_2 has a transition $\delta(s, a) = t$ if and only if there is a sequence of transitions $\{s_j \mid j = 1, \dots, k+1\}$ in M_1 satisfying $\delta(s_j, a_{1,j}) = s_{j+1}$, $s_1 = s$ and $s_{k+1} = t$. So, on reading a generator or its inverse, M_2 makes the transition that M_1 would have made after reading the corresponding word over A_1. Since $w \in A_2^*$ represents the identity if and only if the corresponding word in A_1^* represents the identity, this implies that $L_{M_2} = W(G_{A_2})$. □

Example 4.3. Any finite group G has a regular word problem. To see this we consider the multiplication table presentation for G. We take $G = A$ as the set of semi-group generators, and relations are of the form $g_i g_j = g_k$, the list of all multiplications between any two given elements of G. To build a FSA recognizing the word problem, we take the set of states to be G, and the initial state and only accept state is e. There is a transition $\delta(g_i, a) = g_j$ for every $a \in A$ if $g_i a = g_j$. Since G is finite and because of our choice of presentation, this machine is a FSA allowing us to keep track of group multiplication,

and accepts words precisely when they are equal to the identity in G. The previous proposition assures us the result is true for any presentation of G.

While it is quite natural to construct a FSA recognizing the word problem of a finite group, it turns out that having a regular word problem is also a sufficient condition for a group to be finite. This is an old result due originally to Anisimov [3], but our proof is taken from Muller and Schupp [13].

Theorem 4.4. *A group G has a regular word problem if and only if it is finite.*

Proof. Example 4.3 gives the reverse direction. For the forward direction, assume that $G = \langle X \mid R \rangle$ is not finite and has a regular word problem given by the language $L = L(M)$. Let N be greater than the number of states in the FSA M accepting the word problem. There has to exist a word w_0 of length N such that no proper subword represents the identity. If this were not the case, every word of length greater than N could be reduced to a word of length less than N by successively deleting subwords that represent the identity. This in turn would imply that we could enumerate all the elements of G in a finite list, contradicting the fact the G is infinite. Similar to the pumping lemma decomposition we take $w_0 = xy$, such that y is a loop that can be repeated any number of times leaving M in the same state. In other words, after reading x the machine is in the same state as it would be after reading in $w_0 = xy$. By reading in x^{-1} we should then be in an accept state after reading xx^{-1} and xyx^{-1}, since we know that xx^{-1} represents the identity. But since y is a proper subword of w_0 it cannot represent the identity, hence it is a contradiction that xyx^{-1} is accepted by M.

□

4.2 Context-free Languages and the Word Problem

In [13], Muller and Schupp follow similar lines but characerize groups with a context-free word problem.

Example 4.5. The free group on n generators $F_n = \langle a_1, a_2, \ldots, a_n \rangle$ has a context-free word problem. Taking $X = \{a_1, a_2, \ldots, a_n\}$, any word over $A = X \cup X^{-1}$ that represents the identity can be reduced to the empty word through successive deletions of trivial relations. The PDA M accepting the word problem has a single state s_0 that is also the initial state. Each time a letter is read in, the machine checks whether the top of the stack is the inverse of that letter. If so, the top of the stack is erased. If not, the letter is added to the top of the stack. Defining acceptance by empty stack, this memory operation captures the procedure of successively cancelling trivial relations to arrive at the empty word, and $L(M) = W_A(F_n)$.

Free groups are the key example of groups with a context-free word problem, and we will show that this generalizes to groups having a free subgroup of finite index.

Definition 4.6. A group G is virtually free if it contains a free subgroup of finite index.

Example 4.7. Given two groups G and H together with a homomorphism $\psi : H \to Aut(G)$, we can form a new group denoted $G \rtimes_\psi H$. This is called the semi-direct product of G and H with respect to ψ, and we let $\psi_h = \psi(h)$ be the automorphism defined by the element $h \in H$. The underlying set for this group is the Cartesian product $G \times H$, and multiplication is given component wise by $(g_1, h_1)(g_2, h_2) = (g_1\psi_{h_1}(g_2), h_1h_2)$. Any group of the form $F_n \rtimes_\psi G_f$ for G_f a finite group is virtually free.

Example 4.8. The modular group Γ is virtually free. This is an important group that has a few interpretations. One way to think of Γ is as a subgroup

of Mobius transformations. A Mobius transformation is a map on $\hat{\mathbb{C}} = \mathbb{C} \cup \infty$ of the form

$$z \mapsto \frac{az+b}{cz+d}$$

where $a, b, c, d \in \mathbb{C}$. The Mobius group consists of all automorphisms of $\hat{\mathbb{C}}$. If we restrict our attention to coefficients $a, b, c, d \in \mathbb{R}$ such that $ad - bc = 1$, the resulting transformations are orientation preserving isometries of the hyperbolic plane H, modelled by the upper half plane

$$H = \{x + iy \mid x, y > 0 \in \mathbb{R}\}$$

The modular group Γ is the natural subgroup of these transformations where we take $a, b, c, d \in \mathbb{Z}$. To generate Γ we require two transformations $a_1 : z \mapsto \frac{-1}{z}$ inverts a point about the unit circle, followed by a reflection about the imaginary axis; and $a_2 : z \mapsto z + 1$ is a unit translation. From these generators we obtain the presentation $\langle a_1, a_2 \mid a_1^2, (a_1 a_2)^3 \rangle$, which is recognized as isomorphic to $\mathbb{Z}_2 * \mathbb{Z}_3$. We note that Γ is also commonly described as $PSL(2, \mathbb{Z})$, the quotient of $SL(2, \mathbb{Z})$ by its center, where we identify the Mobius transformation

$$z \mapsto \frac{az+b}{cz+d} \text{ with the matrix } \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

More generally, any free product $G_1 * G_2$ where G_1 and G_2 are finite is virtually free. This follows from considering the natural map from $G_1 * G_2$ to the direct product $G_1 \times G_2$ which is finite. Nielsen showed in [14] that for this map we have the following exact sequence

$$1 \to F_C \to G_1 * G_2 \to G_1 \times G_2 \to 1$$

where the kernel F_C is the free group generated by all the commutators $C = \{g_1 g_2 g_1^{-1} g_2^{-1} \mid g_1 \in G_1, g_2 \in G_2\}$, which is the required finite index free group.

As with groups having a regular word problem, we want to ensure that having a context-free word problem is a property that is invariant under a change of generators. This will be particularly helpful for proving things about such groups, as we can choose convenient presentations to work with.

Lemma 4.9 (Lemma 2 in [13])**.** *Let $G = \langle X_1 \mid R_1 \rangle$ and $G = \langle X_2 \mid R_2 \rangle$ be two presentations for G. Then G has a context-free word problem with respect to A_1 if and only if G has a context-free word problem with respect to A_2. Furthermore, if G has a context-free word problem, then any finitely generated subgroup $H \leq G$ also has a context-free word problem.*

Proof. The proof is very similar to that of Theorem 4.2. The only difference is that when we express $a \in A_2$ as a word $a = a_{1,1} \dots a_{1,k}$ over A_1, in order to simulate the action of M_1 we may have to introduce ϵ-transitions as well as extra states. We use these to keep track of the stack operations that would have taken place after reading the word $a_{1,1} \dots a_{1,k}$ through M_1. The result is that M_2 can simulate the action of M_1, so we end up with the same ID (see Defintion 2.11) after reading in a as we would have after reading $a_{1,1} \dots a_{1,k}$ in M_1. □

We can now state the main theorem of [13] which shows the deep connection between formal language theory and group theory.

Theorem 4.10 (Theorem III in [13])**.** *A finitely generated group $G = \langle X \mid R \rangle$ has a context-free word problem if and only if it is virtually free.*

Remark 4.11. The version in [13] is slightly different in the forward direction, where the group is required to have a context-free word problem as well as be accessible. The authors conjectured that the accessibility condition could be dropped, but lacked a proof. It was subsequently proven that finitely generated groups with a context-free word problem are in fact finitely presented. Combining this with Dunwoody's theorem [7] that all finitely presented groups are accessible, we have the result as stated above.

Remark 4.12. Our goal is to give an outline of the proof of this theorem in both directions. For the reverse implication we will describe directly below the idea of the argument in the general case, as it can be addressed using the concepts developed thus far. The rest of the chapter will be devoted to developing the tools required to show a special case of the forward implication, where the group is assumed to be torsion-free.

Assuming the group G is virtually free, we construct a PDA for the word problem that can be thought of as a combination of the PDA accepting $W(F_k)$, together with the FSA accepting $W(G_f)$ for G_f a finite group. This arises naturally from considering a presentation of G that we can always construct for virtually free groups. The presentation depends on a normal subgroup $N \trianglelefteq H$, where H is the free subgroup of finite index in G. Subgroups of a free group are free so $N \simeq F_k$, and N also has finite index in G. We take generators for G to be $C = \{c_1, \dots c_k\}$ which generate N, together with $D = \{d_1 \dots d_l\}$ such that the natural map $\eta : G \to G/N$ is bijective when restricted to D. For this presentation, any word over the generating set can be re-written as $w_c w_d$ with $w_c \in (C \cup C^{-1})^*$ and $w_d \in (D \cup D^{-1})^*$. This word will represent the identity precisely when $w_c \in W(F_k)$ and $w_d \in W(G/N)$. As the PDA reads in an arbitrary word over the entire generating set, it uses the relations to keep track of the word in this preferred form; after processing the word, w_c will be on the stack, and the machine will be in state $\overline{w_d}$ in G/N. Acceptance is then defined by having empty stack and being in state $d_1 = 1 \in D$.

To prove the special case of the forward implication, we combine some classical results of combinatorial group theory with important facts about the geometry of the Cayley graph. Given a word $w = a_1 \dots a_n \in W_A(G)$, the path $\widehat{w} \in \Gamma_A(G)$ is a closed loop based at the identity. We think about representing this loop as a simple, closed n-gon in the plane, where starting at the identity vertex e and proceeding clock-wise, we label each side with

the letter a_i for $i = 1, \ldots, n$ and each side has length $|a_i| = 1$. We can then triangulate the n-gon by drawing lines between vertices. In the Cayley graph we require that these lines correspond to paths $\widehat{u} = x_1 \ldots x_k$ that connect group elements represented by $\overline{w[i]}$ and $\overline{w[j]}$ for some (i, j) prefixes of w.

Definition 4.13. Using the construction above, we say that a closed path $\widehat{w} \in \Gamma_A(G)$ is K-triangulated if every side of the triangulation $u \in A^*$ satisfies $|u| \leq K$.

Theorem 4.14 (Theorem I in [13])**.** *A finitely generated group $G = \langle X \mid R \rangle$ has a context-free word problem if and only if there exists a constant K such that every closed path in the Cayley graph can be K-triangulated.*

The proof in [13] makes use of the properties of context-free grammars. Briefly, a grammar is a set of rules that tells us how to form words in a language. We start with a finite set of variables V, and use these rules to perform derivations which result in words. Context-free grammars give rise to the same class of languages as those accepted by PDA, however for any context-free grammar the advantage is that we can put it into a normal form, where the rules for generating words in the language are particularly simple. For each $v \in V$ the bound in Theorem 4.14 (as well as the triangulation) comes from picking a particular word w_v derivable from v that has shortest length; we then take $K = \max\{|w_v| \mid v \in V\}$. Being able to K-triangulate closed paths in the Cayley graph turns out to have implications for the number of ends of a group, which we address now.

Definition 4.15. Let $G = \langle X \mid R \rangle$ be a finitely generated group, and $\Gamma_A(G) = \Gamma$ the corresponding Cayley graph. The set

$$B_n(\Gamma) = \{g \in G \mid d_{G_A}(g, e) \leq n\}$$

is the ball of radius n about the origin. The number of ends of the group is the number of ends of the corresponding Cayley graph, given by

$$e(G_A) = e(\Gamma) = \lim_{n\to\infty} \ (\# \text{ components of } \Gamma \setminus B_n \ )$$

We have the following well known theorems for finitely presented groups. The first assures us that our definition of ends is truly a property held by the group and does not depend on the presentation, while the second is an interesting characterization of values that $e(G_A)$ can take on.

Theorem 4.16. *Let $\langle X_1 \mid R_1 \rangle$ and $\langle X_2 \mid R_2 \rangle$ be two presentations for a finitely generated group G. Then $e(\Gamma) = e(G_{A_1}) = e(G_{A_2})$.*

Theorem 4.17. *For a finitely generated infinite group $G = \langle X \mid R \rangle$ we have that $e(\Gamma) = 1, 2$ or ∞.*

Together with a technical combinatorial lemma, K-triangulation of closed paths allows Muller and Schupp to prove the following.

Lemma 4.18 (Lemma 6 in [13])**.** *If $G = \langle X \mid R \rangle$ is an infinite group with a context-free word problem, then G has more than one end.*

The final piece of the puzzle is Stallings' structure theorem for finitely generated groups with more than one end. We state the theorem in the case where G is torsion free.

Theorem 4.19 (Stallings)**.** *A finitely generated torsion free group G has more than one end if and only if $G \simeq \mathbb{Z}$ or $G = G_1 * G_2$ with G_1 and G_2 non-trivial.*

Putting this all together we have.

Theorem 4.20 (Theorem II in [13])**.** *A finitely generated torsion free group is free if and only if it has a context-free word problem.*

Proof. We essentially rephrase the proof given in [13]. Assuming G has a context-free word problem, we prove that G is free by induction on the rank. For the base cases, if $rank(G) = 0$ then G is trivial, and if $rank(G) = 1$ then G is torsion free with one generator therefore isomorphic to $\mathbb{Z}$. If $rank(G) = 2$ then we apply Lemma 4.18 to conclude that G has more than one end. Together with Stallings' theorem we get that $G = G_1 * G_2$. Grushko's theorem say that the rank of a free product is additive, that is $rank(G) = rank(G_1) + rank(G_2)$, and since G_1 and G_2 are non-trivial they have smaller rank than G. Each sits inside G as a finitely generated subgroup, and we apply Lemma 4.9 to conclude that they have a context-free word problem. By induction, G_1 and G_2 are therefore free, and the free product of two free groups is also free. $\square$

Chapter 5

Automatic Groups

Automatic groups are interesting for several reasons. They capture a wide variety of different groups such as braid groups and mapping class groups, as well as many 3-manifold groups. Yet there are important groups such as the discrete Heisenberg groups and some of the Baumslag-Solitar groups that are known to be not automatic. Automatic groups possess nice algorithmic properties, such as having a solvable word problem. Saying a group has a solvable word problem however, does not always mean the solution is a practical one; an algorithm may exist but be horrendously long and inefficient. It turns out that with automatic groups we are guaranteed a fast solution to the word problem. Finally, in terms of this thesis, automatic groups can be defined only using finite state automata. The idea is that we have a regular language of normal forms representing each group element, and we can also recognize right multiplication by a generator using a FSA.

5.1 General Properties

We introduce the concept of a shuffle, that is used to formally capture the idea of having a two-tape automata. A two tape automata is a FSA that

reads in two words at a time, letter by letter. To recognize multiplication by a generator, we want a machine that reads in two words at a time and decides whether or not they differ by a generator. Given an alphabet A, we construct the shuffle alphabet $A_\# = (A \cup \{\$\}) \times (A \cup \{\$\})$. For two words over A of equal length, $w = a_1 \dots a_n$ and $u = b_1 \dots b_n$, the shuffle is defined as $w\#u = (a_1, b_1) \dots (a_n, b_n) \in A_\#^*$. If $|w| = i < |u| = j$ then to take the shuffle we first pad w with $j - i$ end of string symbols to make them the same length, $w = a_1 \dots a_i \overbrace{\$ \dots \$}^{\text{j-i}}$. In a group theory context we force the end of string symbol \$ to map to the identity, and the padding is simply a technicality required to read in pairs of words of different lengths.

Definition 5.1. A two-tape automata M is a FSA over $A_\#^*$, and given a pair of words $w_1, w_2 \in A^*$, we say they are accepted by M precisely when $w_1\#w_2 \in L(M)$, where we pad w_i for $i \in \{1, 2\}$ as neccessary to make them the same length.

Definition 5.2. Let G be a group with a finite set of generators X. Then G has an automatic structure with respect to $A = X \cup X^{-1}$ if there exists a FSA M over A, and a collection of two-tape automata $\{M_x \mid x \in X\}$ over $A_\#$ such that:

1. the natural map $\pi : L(M) \to G$ is surjective
2. for all $x \in X$ and $w_1, w_2 \in A^*$, $w_1\#w_2 \in L(M_x)$ if and only if $\overline{w_1 x} = \overline{w_2}$ and $w_1, w_2 \in L(M)$

The first condition says that there is a regular language such that each element of G is represented by at least one word in that language. The second says that whenever we take two words in our language, feeding them into the two-tape automata M_x tells us whether or not they differ by right multiplication of that generator. Again we can exploit the concepts of formal languages to prove a geometric condition on the Cayley graph. That condition is called the K-fellow traveller property.

Definition 5.3. Let G be a finitely generated group with set of generators X, and $w_1, w_2 \in A^*$. If we consider the paths $\widehat{w_1}, \widehat{w_2} \in \Gamma_A(G)$, we say these paths K-fellow travel if there exists $K \in \mathbb{R}$ such that $d(\overline{w_1[i]}, \overline{w_2[i]})) \leq K$ for all $1 \leq i \leq max\{|w_1|, |w_2|\}$.

If we start out at the identity in $\Gamma_A(G)$ and traverse both paths at the same speed, the distance between the paths at any point in this traversal will be bounded by K. This is also sometimes called synchronous K-fellow travelling. The existence of an automatic structure guarantees that paths which end close together must K-fellow travel, or more precisely:

Lemma 5.4 (2.3.2 in [8])**.** *Let G be finitely generated by X, and assume that G has an automatic structure with FSA M and $\{M_x \mid x \in X\}$. There exists a constant K such that whenever $w_1, w_2 \in L(M)$ and $\overline{w_1 x} = \overline{w_2}$ for $x \in (X \cup \{\epsilon\})$, then $\widehat{w_1}, \widehat{w_2}$ K-fellow travel in $\Gamma_A(G)$.*

Proof. Let K' be greater that the number of states in M and all of the M_x's. Assuming that $w_1, w_2 \in L(M)$ and $\overline{w_1 x} = \overline{w_2}$, then by the definition of automatic structure we have that $w_1 \# w_2 \in L(M_x)$. For any prefix, we need to bound the distance between $\overline{w_1[i]}$ and $\overline{w_2[i]}$. The shuffle $w_1[i] \# w_2[i]$ corresponds to a sequence of state transitions within M_x from a start state s_0 to some state s. There must exist a path from s to an accept state $s_y \in Y$, since $w_1 \# w_2 \in L(M_x)$. If this path has more than K' labels it must visit a state twice, so we can shorten it by cutting out the loop based at this state. The labels on the resulting path correspond to a pair of words $(u_1, u_2) \in A^* \times A^*$. In the group we have $\overline{w_1[i] u_1 x} = \overline{w_2[i] u_2}$, so the path corresponding to $\widehat{v} = u_1 x u_2^{-1}$ connects $\overline{w_1[i]}$ to $\overline{w_2[i]}$ in the Cayley graph and is bounded by $K = 2K' + 1$. □

It turns out that this geometric condition can actually replace the second condition in the definition of automatic structure. This once again highlights the relationship between formal languages and geometric group theory.

Theorem 5.5 (2.3.5 in [8])**.** *Let G be a group finitely generated by X. Then G has an automatic structure with respect to A if and only if there exists a regular language $L = L(M)$ over A and a constant K such that:*

1. *the natural map $\pi : L(M) \to G$ is surjective*
2. *if $w_1, w_2 \in L$ satisfy $\overline{w_1 x} = \overline{w_2}$ for $x \in (X \cup \{\epsilon\})$, then $\widehat{w_1}, \widehat{w_2}$ K-fellow travel in $\Gamma_A(G)$*

Before proceeding to talk further about automatic groups, we should stop to record a theorem assuring us that having an automatic structure is again a geometric property of the group.

Theorem 5.6 (2.4.1 in [8])**.** *A finitely generated group G has an automatic structure with respect to the generating set A_1 if and only if G has an automatic structure with respect to any other generating set A_2.*

Definition 5.7. We say a group is automatic if it has an automatic structure for some set of semi-group generators $A = X \cup X^{-1}$.

Remark 5.8. Of course it is also true that not all automatic structures are created equal. By examining the proof of Theorem 2.4.1 in [8], it is not hard to imagine a case where a particular generating set gives an automatic structure where the K-fellow traveller constant is small, and changing generators would result in a different K' which is quite large.

We now collect some properties that tell us how to find automatic groups, and will help us discuss some particular examples.

Theorem 5.9.

1. *If G_1, G_2 are automatic then $G_1 \times G_2$ is automatic.*
2. *If G_1, G_2 are automatic then $G_1 * G_2$ is automatic.*
3. *G is automatic if and only if H is automatic, for any subgroup $H \leq G$ of finite index in G.*
4. *If G is a finitely generated group that is virtually free or virtually abelian, then G is automatic.*

Remark 5.10. We don't wish to revisit the full proofs that can be found in [8], however a brief discussion here is helpful. For part 1 (4.1.1 in [8]), if G_1 has an automatic structure with a regular language of normal forms given by L_1, and similar G_2 with L_2, the concatenation language L_1L_2 is regular over $A_1 \cup A_2$, and provides the desired language of normal forms for $G_1 \times G_2$ satisfying the K-fellow traveller propery. The language for $G_1 * G_2$ is similarly constructed from L_1 and L_2 and shown to be regular using the closure properties of regular languages (12.1.4 in [8]). Part 4 (4.1.4 in [8]) really follows from part 3 (4.1.6 in [8]), since finitely generated abelian groups as well as free groups have regular languages of normal forms that are easily described, and in fact consist of geodesic (shortest length) words with respect to the word metric.

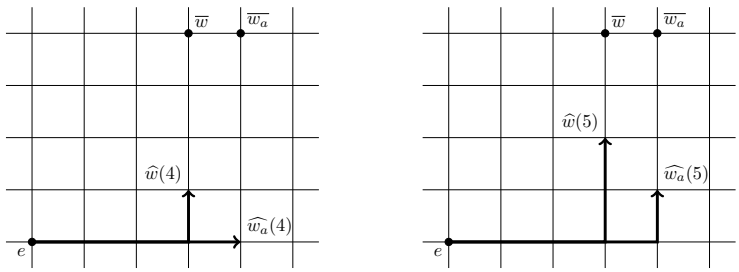
Example 5.11. We refer again to a previous example

$$G = \mathbb{Z} \oplus \mathbb{Z} \simeq \langle a, b \mid ab = ba \rangle$$

We already showed there was a regular language L satisfying $\pi : L \to G$ surjective. To show this regular language satisfies the K-fellow traveller property, we take an arbitrary word $w = a^m b^n$ for $m, n \in \mathbb{Z}$, and consider what happens when we try write $a^m b^n x$ as an element of L, for $x \in \{a, b\}$.

1. Case 1: $x = a$. Then the element in L which corresponds to right multiplication of an arbitrary word is $w_a = a^{m+1}b^n$. As paths in the Cayley graph, $\widehat{w(i)} = \widehat{w_a(i)}$ for all $1 \leq i \leq m$, so $d(\overline{w(i)}, \overline{w_a(i)}) = 0$. At the $(m+1)$-st prefix the paths diverge; $\hat{w}$ moves in the b direction, while $\hat{w_a}$ moves to the right one element in the a direction. But $d(\overline{w(m+1)}, \overline{w_a(m+1)}) = 2$, so the paths remain close. The remainder of the words from the $(m+2)$ prefix on are the same, except that w_a has one more b than w. But it is clear that $d(\overline{w(j)}, \overline{w_a(j)}) \leq 2$ for all $(m+2) \leq j \leq (m+n+1)$.
2. Case 2: $x = b$. Then the element in L corresponding to right multiplication is $w_b = a^m b^{n+1}$. The paths $\hat{w}$ and $\hat{w_b}$ are the same, except that $\hat{w_b}$ has one extra b. So $d(\overline{w(i)}, \overline{w_b(i)}) \leq 1$ for all $1 \leq i \leq (m+n+1)$.

The figure below depicts the above argument for the case $x = a$. We have a fellow traveller constant $K = 2$, and $\mathbb{Z} \oplus \mathbb{Z}$ is automatic.



(a) 4th prefix (b) 5th prefix

Figure 5.1: The normal form paths to group elements differing by right multiplication by a; the paths diverge at the 4th prefix, but remain distance 2 apart.

Example 5.12. A more involved example is the braid group on n strands, B_n. The standard presentation for B_n is given by

$$\langle \sigma_1, \ldots, \sigma_{n-1} \mid \sigma_i\sigma_j = \sigma_j\sigma_i \text{ for } |i-j| > 1, \sigma_i\sigma_{i+1}\sigma_i = \sigma_{i+1}\sigma_i\sigma_{i+1} \rangle$$

The regular language of normal forms for the braid group is determined by examining the lattice structure of the positive braid monoid P_n, which is the set of words over $\{\sigma_1, \ldots, \sigma_{n-1}\}$ modulo the relations of the braid group. For two positive braids p, r, we have a parital order $\prec$ defined by $p \prec r$ if there exists a postive braid q with $pq = r$ in P_n. To have a lattice structure means that for any two elements p, q we can find a greatest common divisor denoted $p \wedge q$ with respect to $\prec$, i.e. $p \wedge q$ is the maximal length word such that $p \wedge q \prec p$ and $p \wedge q \prec q$. With repect to this ordering there is a maximal element Ω such that $p \prec \Omega^k$ for any $p \in P_n$ and some $k \in \mathbb{N}$. A representative of the maximal element is given by $\Omega = \sigma_1(\sigma_2\sigma_1)\ldots(\sigma_{n-2}\ldots\sigma_1)(\sigma_{n-1}\ldots\sigma_1)$,

which is visualized as a half twist. Finally, the normal form for any $p \in P_n$ is obtained by successively taking the maximal divisor of Ω that also divides p. We define this inductively so that $p = q_1$, and if we have a decomposition $p = p_1 \dots p_{k-1} q_k$ then $p_k = \Omega \wedge q_k$ and $p = p_1 \dots p_k q_{k+1}$. Since p is a finite word, this process terminates when $\Omega \wedge q_{n+1} = e$ for some n, and the resulting decomposition $p = p_1 \dots p_n$ satisfies that p_i is the maximal divisor of both Ω and the rest of the word. To check that a positive braid is in this form, it turns out this is equivalent to checking $p_i \wedge p_{i-1} = e$ for successive chunks p_{i-1} and p_i. Since each p_i is a divisor of Ω its length is bounded by $|\Omega|$, and this condition can be checked by a FSA.

How does this all relate to the braid group? We have an embedding $P_n \hookrightarrow B_n$ that allows us to write any braid $b \in B_n$ as $b = \Omega^k p$ with $k \in \mathbb{Z}$ and $p \in P_n$. The regular language for the braid group is the set of words $\Omega^k p_1 \dots p_n$ where $p_1 \dots p_n$ is the normal form described above. To prove the K-fellow traveller property we then proceed with an analysis similar to Example 5.11 above, where we consider the rewriting of a normal form element after multiplication by a generator.

Example 5.13. We can use Theorem 5.9 above together with Example 5.12 to argue that the mapping class group of the $(n+1)$-punctured sphere is automatic. To see this, we consider the capping exact sequence

$$1 \to \langle T_\beta \rangle \to Mod(D_n) \to Mod(S_n, p) \to 1$$

The mapping class group $Mod(D_n)$ of the n-punctured disk turns out to be isomorphic to B_n. In the exact sequence we have $\langle T_\beta \rangle$ which is the subgroup generated by the Dehn twist about the boundary of D_n. This Dehn twist generates the centre of $Mod(D_n)$ and its corresonding element in the braid group is Ω^2. We have $B_n / \langle \Omega^2 \rangle \simeq Mod(D_n) / \langle T_\beta \rangle \simeq Mod(S_n, p)$. The group $Mod(S_n, p)$ is the mapping class group of the n-punctured sphere, where we are fixing a marked point p inside the disk used to cap D_n. This sits inside $Mod(S_{n+1})$ as a finite index subgroup. The automatic structure of $Mod(S_n, p)$ comes from a slight modification to the proof of that B_n is au-

tomatic, and Theorem 5.9 part 3 ensures that $Mod(S_{n+1})$ is also automatic. Mosher went further in showing that the mapping class of any compact surface S (with possibly a finite number of punctures) is also automatic, though more involved techniques were required in the general case [12].

5.2 Solving the Word Problem

As mentioned, automatic groups are important in part because they have a fast solution to the word problem. To outline why a solution exists we will proceed by describing an important function called the isoperimetric function, or Dehn function. In section 3.1 we discussed how to arrive at a presentation of G starting with a finite set of generators X. The map $\pi : F(X) \to G$ is surjective, and if $\ker \pi = \langle R \rangle$ then R is a set of relations for G, and $G = \langle X \mid R \rangle$ is a presentation. Due to this construction, if we take any word $w \in W_A(G)$, it is possible to rewrite it as a product

$$w = u_1 r_1^{\epsilon_1} u_1^{-1} \dots u_k r_k^{\epsilon_k} u_k^{-1}$$

where $u_i \in F(X)$, $r_i \in R$ and $\epsilon_i \in \{1, -1\}$. There are of course many ways to decompose an arbitrary word in $W_A(G)$ in such a way.

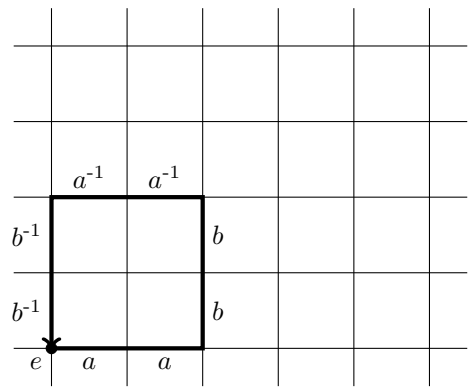


Figure 5.2: A closed loop in the Cayley graph bounding four smaller squares.

Remark 5.14. It is a somewhat subtle point to remember that although the smallest normal subgroup of $F(X)$ containing R consists of words of the form $w = u_1 r_1^{\epsilon_1} u_1^{-1} \dots u_k r_k^{\epsilon_k} u_k^{-1}$, an arbitrary word $w \in W_A(G)$ is not neccesssarily in this form. An example depicted above for $\mathbb{Z} \oplus \mathbb{Z}$ is given by the word $a^2b^2a^{-2}b^{-2}$, a closed loop in the Cayley graph that represents the identity. If we rewrite it as a product of conjugates of relators, we see that $w = (ara^{-1})(abrb^{-1}a^{-1})(r)(brb^{-1})$ where $r = aba^{-1}b^{-1}$. We have expressed the word as a product of conjugates of 4 relators. In the Cayley graph we notice that the outside loop $a^2b^2a^{-2}b^{-2}$ is a square which contains four smaller squares. This suggests the following definition.

Definition 5.15. Given $w \in W_A(G)$, we define the area of the word to be $area(w) = \min\{k \mid w = u_1 r_1^{\epsilon_1} u_1^{-1} \dots u_k r_k^{\epsilon_k} u_k^{-1}\}$.

The area gives us some insight into the complexity of the group with respect to the relations. If we take $w_1, w_2 \in W_A(G)$ with $|w_1| = |w_2| = n$, then we can think of w_1 being more complex than w_2 if $area(w_1) > area(w_2)$. Further, we would like to see how the $area(w)$ changes as we

increase the length of the word being considered. This is the motivation behind the following:

Definition 5.16. The isoperimetric function of a group $G = \langle X \mid R \rangle$ is defined with repect to a particular presention, and is given by:

$$\phi_A(n) = \max\{area(w) \mid w \in W_A(G) \text{ and } |w| = n\}$$

Remark 5.17. If G has presentations $\langle X_1 \mid R_1 \rangle$ and $\langle X_2 \mid R_2 \rangle$ there exists a constant c such that $\phi_{A_1} \leq c_1 \phi_{A_2}$. So isoperimetric functions are determined by the group up to a constant and we denote it simply by ϕ.

A solution to the word problem can be established by proving that ϕ is bounded by a class of functions known as recursive. A recursive function is a very theoretical concept, but the core idea is that the output of the function can be computed using a finite algorithm. This class of functions includes polynomials, exponential functions and trigonometric functions. It also includes much more general functions, for example the function which computes the greatest common divisor of any two integers. In our case we only need to work with functions that are maps on the natural numbers.

Definition 5.18. A function $f : \mathbb{N} \to \mathbb{N}$ is recursive if there exists a finite list of steps to compute $f(n)$ for all $n \in \mathbb{N}$.

Definition 5.19. We say a group $G = \langle X \mid R \rangle$ satisfies a recursive isoperimetric inequality if there is a recursive function $f(n)$ such that $\phi(n) \leq f(n)$ for all $n \in \mathbb{N}$.

The theorem showing equivalence of a solvable word problem with a recursive isoperimetric inequality becomes relatively straightforward after establishing the following lemma.

Lemma 5.20 (2.24 in [8]). *If $w = u_1 r_1^{\epsilon_1} u_1^{-1} \dots u_k r_k^{\epsilon_k} u_k^{-1}$ is a decompostion of a word representing the identity, there exists a constant $C = C(|w|, k, \max |r_i|)$, such that we can re-write the word as*

$$w = v_1 r_1^{\epsilon_1} v_1^{-1} \dots v_k r_k^{\epsilon_k} v_k^{-1}$$

with $|v_i| < C$ for all $1 \leq i \leq k$.

Theorem 5.21 (2.2.5 in [8]). *A group G has a solvable word problem if and only if it satisfies a recursive isoperimetric inequality.*

Proof. If G has a solvable word problem, then given any $n \in \mathbb{N}$ we first enumerate all words of length up to n, and then decide which words w_e belong to $W_A(G)$. Lemma 5.20 above implies that we can find the area of each w_e by exhausting all possible decompositions; since we have a bound on the $|u_i|$, we just increase the number of possible relators considered until we arrive at a decomposition. Conversely, if ϕ is bounded by some recursive function f we consider a word w of length $|w| = n$ and check to see if $w \in W_A(G)$. If it is, we can decompose it as $w = u_1 r_1^{\epsilon_1} u_1^{-1} \dots u_k r_k^{\epsilon_k} u_k^{-1}$, where $k \leq f(n)$. Again by Lemma 5.20 we can exhaust all possible decompositions of this form, because of the bound on $|u_i|$. □

Theorem 5.22 (2.3.12 in [8]). *If G is automatic, then it is finitely presented and satisfies a quadratic isoperimetric inequality.*

There are two things to note here. First, this fact is convenient because it provides an approach to showing that certain groups are not automatic. In [8] this is demonstrated for the 3-dimensional Heisenberg group

$$H_3 = \langle a, b, c \mid ac = ca, bc = cb, ab = cba \rangle$$

which has a cubic isoperimetric inequality. To provide some intuition for this result, consider the very natural language of normal forms given by $L_H = \{a^i b^j c^k \mid i, j, k \in \mathbb{Z}\}$. We can write any element in this way since

the c generator commutes with the other two generators, and we can move the a generator to the left of b simply by introducing another c generator. It is clear that $\pi : L_H \to H_3$ is actually bijective. If we try to perform a similar analysis to Example 5.11 however, it is clear where things break down. Taking a word $w = a^i b^j c^k \in L_H$ and multiplying by the a generator on the right, we must introduce exactly j more generators in order to move it to the left; the resulting word is $wa = a^i b^j c^{j+k}$ and there is no way to control the distance between the paths in the Cayley graph.

Interestingly enough we also have $(2n+1)$-dimensional Heisenberg groups which are nilpotent. Epstein and Holt show that a nilpotent automatic group must be virtually abelian, which implies that H_{2n+1} is not automatic [8]. However, these groups are shown in [1] to have a quadratic isoperimetric inequality for $n \geq 2$, showing that the converse to Theorem 5.22 does not hold.

5.3 Efficiency of the Solution

Finally, we would like to see that automatic groups have an efficient solution to the word problem. By efficiency we are refering to the time complexity of the algorithm that solves the word problem. This is a very standard and important concept in computer science, but we shall describe it informally and as suits the context of this thesis. What we are looking for is a time complexity function f_{tc}, that takes as input the length of a word $|w|$ and tells us approximately how many steps it will take to determine if $w \in W_A(G)$. The reason we say approximately is that we are not concerned about the precise number of steps as much as the leading behaviour of f_{tc}, which we denote using Landau's notation $O(f_{tc})$. For example if we have two complexity functions $f_{tc}(n) = 3n^2$ and $g_{tc}(n) = n^2 + 5$ then $O(f_{tc}) = O(g_{tc}) = n^2$ and

we say that the algorithm has quadratic time.

Theorem 5.23 (2.3.10 in [8])**.** *If G is an automatic group and $w \in A^*$, then the time complexity function to determine if $w \in W_A(G)$ is $O(|w|^2)$. In other words, the word problem is solvable in quadratic time.*

Chapter 6

Generalizations of Automatic Structures

Having surveyed some aspects of automatic groups, we are led to two natural and interconnected questions. Firstly, in what sense is the theory of automatic groups deficient? It should be possible to give examples of important non-automatic groups, as well as understand why they are not automatic. In addition, we should identify certain properties that do not hold for automatic groups, or at least are not known to be true. By way of example, if a direct product $G \times H$ is an automatic group, it has not yet been determined whether the factors G or H must then be automatic. Contrast this with Theorem 5.9 from the last chapter that a direct product of two automatic groups is automatic. The second question which relates more closely to the theme of this thesis is how can we generalize the idea of an automatic structure to include more complex formal language classes? The cornerstone of automatic groups are FSA, yet we have seen how generalizing this machine allows us to recognize more interesting kinds of languages.

With respect to the first question, one answer points in the direction of 3-manifold theory. It would be nice if the fundamental group of any compact

3-manifold were automatic, however this is not the case. Although six of the eight geometries give rise to fundamental groups that are automatic, any manifold with a JSJ decomposition containing a piece based on the Nil or Sol geometry has a fundamental group that is not automatic. To answer the second question we will highlight results from [6], where the authors generalize automatic structures by using PDA instead of FSA, and this turns out to have implications for 3-manifold theory. From there we will discuss combings of groups, which is another generalization of automatic structure. Here we keep the K-fellow traveller property, but consider other formal language classes besides regular ones.

6.1 Parallel-poly Pushdown Groups

The class of groups introduced in [6] is almost a straight generalization of automatic groups by replacing FSA with PDA. Recall however that context-free languages are not closed under intersection, and it is reasonable to assume we should avoid this complication.

Definition 6.1. A language L is parallel-poly pushdown if there are finitely many PDA $\{M_i \mid i = 1, \dots n\}$ such that $L = \bigcap_{i=1}^{n} L(M_i)$. We will denote the class of parallel-poly pushdown languages by $\mathscr{P}$.

The nomenclature, while undoubtedly bulky, is meant to be suggestive of how to interpret $\mathscr{P}$ languages. The type of machine that defines these languages can be thought of as a collection of PDA working simultaneously to process words. This allows for a finite number of stacks on which to store an unbounded amount of memory.

Definition 6.2. Let G be a group finitely generated by X. We say that G is $\mathscr{P}$ if there exists a $\mathscr{P}$ language L over A and a collection of $\mathscr{P}$ languages $\{L_x \mid x \in X\}$ over the shuffle alphabet $A_\#$ such that:

1. the natural map $\pi : L \to G$ is surjective
2. for all $x \in X$ and $w_1, w_2 \in A^*$, $w_1 \# w_2 \in L_x$ if and only if $\overline{w_1 x} = \overline{w_2}$ and $w_1, w_2 \in L$

The class of $\mathscr{P}$ groups contains automatic groups, as regular languages are a subclass of context free ones. As with automatic groups, we have the following:

Theorem 6.3 (4.2 in [6])**.** *If a group G is $\mathscr{P}$, then G has a solvable word problem.*

Furthermore we have some similar closure properties.

Theorem 6.4 (5.1 in [6])**.** *1. If G_1, G_2 are $\mathscr{P}$, then $G_1 \times G_2$ is $\mathscr{P}$*
2. If G_1, G_2 are $\mathscr{P}$ then $G_1 * G_2$ is $\mathscr{P}$

One of the key differences however has to do with closure under wreath product, which has important implications. For any two sets G and H, we denote by G^H the set of G valued functions on H, i.e.

$$G^H = \{\beta : H \to G \mid \beta \text{ a function}\}$$

To turn this set into a group we define the multiplication of two functions componentwise, so $(\beta\gamma)(h) = \beta(h)\gamma(h)$ for all $h \in H$. We have an action of H on G^H which gives rise to a map $\psi : H \to Aut(G^H)$, given by $h \mapsto \beta^h$, where $\beta^h(x) = \beta(xh^{-1})$. Now we are able to form the semi-direct product of H with G^H which we define as the wreath product,

$$G \wr H = H \rtimes_\psi G^H$$

This is important because it allows us to construct groups that are not finitely presentable; i.e. groups which have no presentation with both a finite set of generators and relations. An example of such a group is the Lamplighter

group, which is realized as the wreath product $\mathbb{Z}_2 \wr \mathbb{Z}$. Since $\mathbb{Z}_2$ is finite and $\mathbb{Z}$ is abelian, both of these groups are automatic. However their wreath product the Lamplighter group has the presentation $\langle a, t \mid (at^n at^{-n})^2, n \in \mathbb{Z} \rangle$ with an infinite number of relations. Furthermore, Baumslag has shown in [4] that $\mathbb{Z}_2 \wr \mathbb{Z}$ is not finitely presentable. In the general case he shows that when G and H are finitely presented, $G \wr H$ is not finitely presentable whenever $G \neq 1$ and H is infinite, as in the case of the Lamplighter group. This cannot be automatic since automatic groups must be finitely presented by Theorem 5.22. However, the group $\mathbb{Z}_2 \wr \mathbb{Z}$ is $\mathscr{P}$ due to the following.

Theorem 6.5 (5.4 in [6])**.** *If G and H are $\mathscr{P}$ then the wreath product $G \wr H$ is $\mathscr{P}$.*

Another important difference has to do with the isoperimetric inequality, which we have seen is quadratic for automatic groups.

Theorem 6.6 (5.8 in [6])**.** *For every $k \in \mathbb{N}$ there exists a $\mathscr{P}$ group G whose isoperimetric inequality is $O(n^k)$.*

Finally, we outline how $\mathscr{P}$ groups present a more complete picture of the fundamental groups of all compact 3-manifolds. As with many papers of the time, the authors worked with 3 manifolds satisfying Thurston's conjecture, which was subsequently proven by Perelman, as described in [15]. Six of the eight geometries result in compact 3-manifolds whose fundamental group is automatic; the most prevalent cases are hyperbolic. For 3-manifolds based on the Nil or Sol geometry, the fundamental group contains a finite index subgroup of the form $\mathbb{Z}^2 \rtimes_\psi \mathbb{Z}$. Unlike automatic groups, it is not known whether the property of being $\mathscr{P}$ passes from arbitrary finite index subgroups to the group itself. However, the authors show directly the following two results.

Theorem 6.7 (5.9 in [6]). *If a group G contains a finite index subgroup isomorphic to $\mathbb{Z}^n \rtimes_\psi \mathbb{Z}$ then G is $\mathscr{P}$.*

Theorem 6.8 (5.6 in [6]). *If H is a $\mathscr{P}$ group with $\psi : H \to Aut(\mathbb{Z}^n)$, then $\mathbb{Z}^n \rtimes_\psi H$ is $\mathscr{P}$.*

This gives the desired result.

Theorem 6.9 (5.10 in [6]). *If M is a compact 3-manifold, then $\pi_1(M)$ is a $\mathscr{P}$ group.*

6.2 Combings

Finally we discuss combings of groups, which is another way to generalize automatic structure. While $\mathscr{P}$ groups were seen to have quite different properties from automatic groups, combable groups are much more similar in nature. Once again we have a language of normal forms for group elements, but the key difference is that combings have more to do with the geometry of the Cayley graph. The defining property is that the paths in the Cayley graph must stay close together, if the group elements they represent differ by a generator. This is essentially the K-fellow traveller property defined above, however we do not require that the paths synchronously fellow travel.

To address this, we think of the paths in the Cayley graph as having a continuous parametrization, where an edge is traversed at unit speed. In other words, given a group with a finite generating set X and $w = a_1 \dots a_n \in A^*$, we parametrize $\widehat{w}$ by $t \in [0, \infty)$ so that the length of $\widehat{w}(t)$ is t when $t < |w|$. For all $t \geq |w|$ we have $\widehat{w}(t) = \overline{w}$, and we think of the path as remaining at the end point of the word w. For $t \in \mathbb{N}$ we see that the length of $\widehat{w}(t)$ is equal to the word length of the prefix, $|w(t)|$. We may now restate the definition of what it means to synchronously fellow travel.

Definition 6.10. Let G be finitely generated by X, and $v, w \in A^*$. As continuously parametrized paths in the Cayley graph we say that $\widehat{v}$ and $\widehat{w}$ synchronously K-fellow travel if there exists $K \in \mathbb{N}$ such that:

$$d(\widehat{v}(t), \widehat{w}(t)) \leq K, \text{ for all } t \in [0, \infty)$$

Remark 6.11. In the previous definition of fellow travelling, we checked that the distance between all prefixes of equal length was bounded with respect to the word metric . Here the distance is the path metric, of the shortest path which joins the end-points of $\widehat{v}(t)$ and $\widehat{w}(t)$. The first definition is preferred by the author as being more natural to the discrete and combinatorial aspects of the Cayley graph, as well as group presentations and the theme of this thesis. The second is necessary to coherently introduce to the notion of asynchronous fellow travelling.

Definition 6.12. Let G be finitely generated by X and let $v, w \in A^*$ be two words with corresponding paths $\widehat{v}, \widehat{w}$ parametrized by $t \in [0, \infty)$. We call $h : \mathbb{R} \to \mathbb{R}$ a speed function if it is differentiable, maps $[0, |v| + 1]$ onto $[0, |w| + 1]$, and is strictly increasing on $[0, |v| + 1]$.

Remark 6.13. The reason we add one to the length of v and w when considering the intervals is that we want to be able to consider cases where v is the empty word of length zero, but w has positive word length.

Definition 6.14. Let G be finitely generated by X, and $v, w \in A^*$. As continuously parametrized paths in the Cayley graph we say that $\widehat{v}$ and $\widehat{w}$ asynchronously K-fellow travel if there exists $K \in \mathbb{N}$ and speed function $h : \mathbb{R} \to \mathbb{R}$ such that :

$$d(\widehat{v}(t), \widehat{w}(h(t)) \leq K, \text{ for all } t \in [0, \infty)$$

Now we are able to define combings in their most general sense. Recall that automatic groups could be defined by having a regular language, where words that represented group elements differing by a generator were required to synchronously fellow travel.

Definition 6.15. Let G be finitely generated by X, and $\mathscr{L}$ be a formal language class. The group has an $\mathscr{L}$-combing (with respect to A) if there exists a language $L \in \mathscr{L}$ over A, and a constant $K \in \mathbb{N}$ such that:

1. the natural map $\pi : L \to G$ is surjective
2. if $w_1, w_2 \in L$ satisfies $\overline{w_1 x} = \overline{w_2}$ for $x \in X \cup \epsilon$, then $\widehat{w_1}, \widehat{w_2}$ asynchronously K-fellow travel

We call a group combable if it admits an $\mathscr{L}$-combing, and note that when the speed function $h : \mathbb{R} \to \mathbb{R}$ is the identity, and L is a regular language then the group is automatic. The following properties show that combable groups are closely related to automatic groups. The results are true for regular and context-free languages, but also other formal languages that we have not discussed including indexed, context-senstive, and real-time languages.

Theorem 6.16. *Let G be finitely generated by X_1 and X_2. Then G is combable with respect to A_1 if and only if G combable with respect to A_2.*

Theorem 6.17.
1. *If G and H are combable, then $G \times H$ is combable.*
2. *If G and H are combable, then $G * H$ is combable.*
3. *G is combable if and only if H is combable, for any subgroup $H \leq G$ of finite index in G*

Despite these similarities, combable groups are indeed a generalization of automatic groups. One important distinction from automatic groups arises from allowing asynchronous combings, as the following example shows.

Example 6.18. The Baumslag-Solitar groups are used as examples in many areas of combinatorial group theory, and are deceivingly simple in their description. For integers $p, q \in \mathbb{Z}$ we have the associated Baumslag-Solitar group $BS(p, q)$ defined by the presentation $\langle a, b \mid a^p b = b a^q \rangle$. This class of groups includes the free abelian group on two generators $BS(1, 1)$ (which has been a recurring example in this thesis), as well as the fundamental group of

the Klein bottle $BS(1,-1)$. For any p,q we can construct a regular language of normal forms L, where $\pi : L \to BS(p,q)$ is in fact bijective. The language L always gives a combing of $BS(p,q)$, which is sychronous for $p = \pm q$ (hence showing the group is automatic in this case), and asynchronous otherwise. Furthermore, for $p \neq \pm q$ it is shown in [8] that the isoperimetric inequality is bounded below by a polynomial of arbitrary degree, and so $BS(p,q)$ cannot be automatic.

Chapter 7

Nested Words and Group Theory

We have outlined a range of topics in geometric group theory where the central concept is formal languages. Visibly Pushdown Languages (VPLs) are a relatively new class of language introduced in [2] that fits somewhere between regular and context-free languages, and we will show how they relate to some of this research. One of the distinguishing features of VPLs is they are comprised of nested words, which are words with an associated matching relation. One way to interpret a VPL is as being generated by a certain type of PDA. The key difference is that stack operations are completely determined by the input symbol being read in. Any symbol in the alphabet is designated to be a call, return, or internal symbol. On reading a call, the machine can write to the top of the stack, and this is the only time it is allowed to do so. Alternatively, reading a return forces the machine to erase the top of stack symbol, and reading an internal symbol leaves the stack unchanged. What we are really processing with this type of PDA is not a word in the sense of chapter 2, but something called a nested word.

7.1 Theory of VPLs

Definition 7.1. A nested word is a pair $(w, \frown)$ where $w = a_1 \dots a_n \in A^*$ and $\frown$ is a subset of $\{-\infty, 1, \dots, n\} \times \{1, \dots, n, \infty\}$ satisfying:

- (Matching edges go forward) $i \frown j \Rightarrow i < j$
- (Uniqueness) $\forall\ 1 \leq i \leq n$, $\#\{j \mid i \frown j\} \leq 1$ and $\#\{j \mid j \frown i\} \leq 1$
- (Nesting Property) If $i_1 \frown j_1$ and $i_2 \frown j_2$ with $i_1 < i_2$, then either $j_2 < j_1$ or $j_1 < i_2$

If $i \frown j$ then we say that a_i is a call and a_j is return. When $i \frown \infty$ then a_i is a pending call, and similiary $-\infty \frown j$ makes a_j a pending return. If a_i is neither a call nor a return, then we say it is an internal symbol. In the case where each call position has a corresponding return position (i.e. none of the calls or returns are pending) the word is called well-matched. We let $NW(A)$ be the set of all nested words over A, and $WM(A)$ be the subset of well-matched nested words. It will be convenient to encode nested words over A by extending the alphabet to the tagged alphabet $\tilde{A} = A^{\mathfrak{c}} \cup A \cup A^{\mathfrak{r}}$. The alphabets $A^{\mathfrak{c}}$ and $A^{\mathfrak{r}}$ are disjoint copies of A where each element $a \in A$ is replaced with $a^{\mathfrak{c}}$ and $a^{\mathfrak{r}}$ respectively. The idea is that we can tag the letters of any normal word $w \in A^*$ to be either a call or return, or left as an internal symbol. We denote tagged words by $\widetilde{w} \in \widetilde{A}$, and use a different script for letters $\mathfrak{a} \in \widetilde{A}$, where $\mathfrak{a} \in \{a^{\mathfrak{c}}, a^{\mathfrak{r}}, a\}$.

Lemma 7.2 (2.1 in [2])**.** *There is a natural bijection $\tau : NW(A) \to \tilde{A}^*$ given by extending the map:*

$$a_i \mapsto \begin{cases} a_i^{\mathfrak{c}} & \textit{if } a_i \textit{ is a call} \\ a_i^{\mathfrak{r}} & \textit{if } a_i \textit{ is a return} \\ a_i & \textit{if } a_i \textit{ is an internal symbol} \end{cases}$$

Example 7.3. When we work over the tagged alphabet $\widetilde{A}$, this lemma says that for any tagged word $\widetilde{w} \in \widetilde{A}^*$, there is precisely one way to interpret the call/return tags that yields a matching relation satisfying the three properties of Definition 7.1. Consider the alphabet $A = \{a, b, c\}$, which we extend to the tagged alphabet $\widetilde{A} = \{a, a^{\mathfrak{c}}, a^{\mathfrak{r}}, b, b^{\mathfrak{c}}, b^{\mathfrak{r}}, c, c^{\mathfrak{c}}, c^{\mathfrak{r}}\}$. An example of a tagged word is $\widetilde{w} = a^{\mathfrak{c}}ab^{\mathfrak{c}}c^{\mathfrak{r}}$. To determine $\tau^{-1}(\widetilde{w})$ we must specify the matching relation that accompanies the word $aabc \in A^*$. If the pair $1 \frown 4$ were to belong to the matching relation this means we match the first a with the c; but this would imply that we would also have $3 \frown \infty$, or the b as a pending call. This would in turn contradict the nesting property that a matching relation must posses. The correct matching relation associated to the tagged word must be $\{1 \frown \infty, 3 \frown 4\}$.

When we are considering a language of nested words over an alphabet A, Lemma 7.2 allows us to think of it as a traditional word language over the extended alphabet $\widetilde{A}$. This leads to a convenient way to define regular languages of nested words, or visibly pushdown languages.

Definition 7.4. Given an alphabet A, we consider a special type of PDA over $\widetilde{A}$. A visibly pushdown automata (VPA) $\widetilde{M}$ over $\widetilde{A}$ is a tuple $\widetilde{M} = (\widetilde{A}, S, \Gamma, s_0, \gamma_0, Y, \Gamma_y, \delta)$ with interpretations the same as for a PDA, except:

- $\Gamma_y \subset \Gamma$ is the set of hierarchical accept states
- $\delta = \delta_c \cup \delta_i \cup \delta_r$ is the transition function, which depends on the symbol $\mathfrak{a} \in \{a, a^{\mathfrak{c}}, a^{\mathfrak{r}}\}$

$$\delta(s, \mathfrak{a}) = \begin{cases} \delta_c(s, a^{\mathfrak{c}}) = (s', \gamma_i) & \text{if } \mathfrak{a} = a^{\mathfrak{c}} \\ \delta_i(s, a) = s' & \text{if } \mathfrak{a} = a \\ \delta_r(s, a^{\mathfrak{r}}, \gamma) = s' & \text{if } \mathfrak{a} = a^{\mathfrak{r}} \end{cases}$$

Remark 7.5. The key difference from a PDA is of course the nature of the transition function. We are restricting the stack operations in a serious way.

Reading in a call allows the machine to transition based on the current state and the symbol being read in, and the machine is forced to write a non-trivial symbol γ_i to the top of the stack. Reading an internal symbol means the stack will remain unchanged; a VPA which processes only internal symbols would reduce to a FSA since no memory storage is possible. Finally, when reading in a return symbol the machine makes a transition based on the current state, the symbol being read in, and the current top of stack symbol. The current top of stack symbol is then erased (or popped off), exposing the one below. If the top of the stack is γ_0 however the transition takes place but the bottom of stack symbol remains unchanged. This allows us to process nested words with pending returns. To allow for acceptance of words with pending calls, we include $\Gamma_y \subset \Gamma$. This also is flexible notation in that we have acceptance by empty stack precisely when $\Gamma_y = \emptyset$.

Definition 7.6. Given a VPA $\widetilde{M} = (\widetilde{A}, S, \Gamma, s_0, \gamma_0, Y, \Gamma_y, \delta)$, as before we define an instantaneous description (ID) of $\widetilde{M}$ to be a triple $(s, \widetilde{w}, \chi)$ where $s \in S$, $\widetilde{w} \in \widetilde{A}^*$ and $\chi \in \Gamma^*$. We have:

$$(s, \mathfrak{a}\widetilde{w}, \chi\gamma) \vdash \begin{cases} (t, \widetilde{w}, \chi\gamma\gamma') & \text{if } \mathfrak{a} \in A^{\mathfrak{c}} \text{ and } \delta_c(s, \mathfrak{a}) = (t, \gamma') \\ (t, \widetilde{w}, \chi\gamma) & \text{if } \mathfrak{a} \in A \text{ and } \delta_i(s, \mathfrak{a}) = t \\ (t, \widetilde{w}, \chi) & \text{if } \mathfrak{a} \in A^{\mathfrak{r}} \text{ and } \delta_r(s, \mathfrak{a}, \gamma) = t \end{cases}$$

Then we take $\vdash^*$ as the reflexive and transitive closure of $\vdash$. The language of words accepted by $\widetilde{M}$ is

$$L(\widetilde{M}) = \{\widetilde{w} \in \widetilde{A}^* \mid (s_0, \widetilde{w}, \gamma_0) \vdash^* (t, \epsilon, \chi) \text{ for some } t \in S \text{ and } \chi \in \Gamma_y^*\}$$

Definition 7.7. Given an alphabet A, a language $L \subset \widetilde{A}^*$ is a visibly push-down language if there exists a VPA $\widetilde{M}$ such that $L = L(\widetilde{M})$.

We will often refer to VPLs as regular languages of nested words. The next two results clarify the somewhat subtle relationship between regular languages of nested words and context-free languages.

Theorem 7.8 (5.1 in [2])**.** *If $L \subset \widetilde{A}^*$ is a visibly pushdown language, then it is also a context-free language over the alphabet $\widetilde{A}$.*

Theorem 7.9 (5.2 in [2])**.** *If $L \subset A^*$ is a context-free language over the alphabet A, then there exists a visibly pushdown language $L' \subset \widetilde{A}^*$ such that $\rho(L') = L$, where $\rho : \widetilde{A}^* \to A^*$ is the re-naming function given by extending:*

$$\begin{cases} a_i^{\mathfrak{c}} \mapsto a_i \\ a_i^{\mathfrak{r}} \mapsto a_i \\ a_i \mapsto a_i \end{cases}$$

Despite the above theorems, there are languages which are naturally languages of nested words but which are not context-free in the strict sense. We illustrate this point in the following example.

Example 7.10. As mentioned in Remark 2.15, the language of words given by $L_{pal} = \{ww^R \mid w \in A^*\}$ is not context-free according to our definition because there is no deterministic PDA that is able to recognize it. However when we consider nested words over A, it is possible to construct a VPA that recognizes this as a visibly pushdown language. More precisely, fixing an alphabet A, the language $\{a_1^{\mathfrak{c}} \dots a_n^{\mathfrak{c}} a_n^{\mathfrak{r}} \dots a_1^{\mathfrak{r}} \mid a_i \in A\}$ is a VPL. Under the renaming function ρ this would map precisely to L_{pal}. The underlying matching relation that defines this set of words matches the i-th letter of w with the $(n-i)$-th letter of w^R, and a very simple VPA can check the condition that these are equal. Although this language corresponds to a context-free language by Theorem 7.8, the crucial point is that we would have to extend the alphabet to be $\widetilde{A}$. This suggests that working with nested words can be a more natural way to think about certain languages.

The other important implication of Lemma 7.2 is that it gives a natural way to define operations on nested words, since we can just use standard word operations over the tagged alphabet. We fix an alphabet A and let $\widetilde{w_j} \in \widetilde{A}^*$ for $j = 1, 2$. Concatenation is defined as $\widetilde{w_1}\widetilde{w_2}$, and the prefix of

length i is $\widetilde{w_j}[i]$, where we use the standard word operations over $\widetilde{A}$. Finally, to define reversal we consider a word $\widetilde{w} = \mathfrak{a}_1 \dots \mathfrak{a}_n$ where $\mathfrak{a}_i \in \{a_i^{\mathfrak{c}}, a_i^{\mathfrak{r}}, a_i\}$ and $a_i \in A$. The reversal of $\widetilde{w}$ is given by $\widetilde{w}^R = \mathfrak{b}_n \dots \mathfrak{b}_1$, where

$$\mathfrak{b}_i = \begin{cases} a_i^{\mathfrak{r}} & \text{if } \mathfrak{a}_i = a_i^{\mathfrak{c}} \\ a_i^{\mathfrak{c}} & \text{if } \mathfrak{a}_i = a_i^{\mathfrak{r}} \\ a_i & \text{if } \mathfrak{a}_i = a_i \end{cases}$$

In other words, we reverse the order of the letters, change calls to returns and vice-versa, while leaving internal symbols unchanged. This simplifies the notation in the example directly above; the VPL there can be described as $\{\widetilde{w}\widetilde{w}^R \mid \widetilde{w} \in (A^{\mathfrak{c}})^*\}$, now that we have specified what it means to take the reversal of a nested word. The reason we sometimes refer to VPLs as regular languages of nested words, is that they share the same nice closure properties held by regular languages. We have the analogue of Theorem 2.9.

Theorem 7.11 (3.5-3.7 in [2])**.** *Let L, L_1, and L_2 be VPLs over the alphabet A. The following languages are also visibly pushdown:*

1. $L_1 \cap L_2$
2. $L_1 \cup L_2$
3. $L_1 L_2$
4. L^*
5. $L^R = \{w^R \mid w \in L\}$
6. $L_{comp} = \widetilde{A}^* \setminus L$
7. $L_{pre} = \{w \mid wu = v \in L\}$, *the prefix closure of* L

7.2 VPLs and the Word Problem

Here we present new results that seek to clarify how regular languages of nested words can be useful for understanding the language of words representing the identity in a finitely generated group.

A standard question to ask of any formal language class is how it behaves with respect to regular languages. We record the following lemma for VPLs.

Lemma 7.12. *Let L_r be a regular language over A, and $L \subset \widetilde{A}$ be a visibly pushdown language of nested words over A. The following are also visibly pushdown languages:*

1. $L_r \cap L$
2. $L_r \cup L$

Proof. This follows by interpreting the FSA M such that $L_r = L(M)$ as a VPA, where no stack operations take place. The words in L_r can be thought of as nested words with an empty matching relation. We then apply Theorem 7.11

□

Given a VPL $L \subset \widetilde{A}^*$ and a regular language $L_r \subset B^*$, there are other ways to generate new VPLs besides taking the union or intersection. As an example, consider a word $\widetilde{w} = \mathfrak{a}_1 \dots \mathfrak{a}_n \in L \subset \widetilde{A}^*$. If we take a symbol $b \notin A$, we can think of inserting the new symbol between any two of the symbols in $\widetilde{w}$. Because this doesn't affect the matching relation, our intuition is that

$$\{\mathfrak{a}_1 \dots \mathfrak{a}_i b \mathfrak{a}_{i+1} \dots \mathfrak{a}_n \mid \widetilde{w} = \mathfrak{a}_1 \dots \mathfrak{a}_n \in L, 1 \leq i \leq (n-1)\}$$

is also a VPL. The idea for the following came from [10] where the formal idea of a language scramble was encountered. The operation described below is actually referred to there as a language shuffle, but we have used that terminology elsewhere.

Definition 7.13. Given two (disjoint) alphabets A and B, we consider a VPL $L \subset \widetilde{A}^*$ and a word language $L_0 \subset B^*$. The scramble of L and L_0 is denoted by $L \bowtie L_0$ and is given by

$$L \bowtie L_0 = \{\mathfrak{u}_1 v_1 \mathfrak{u}_2 v_2 \dots \mathfrak{u}_n v_n \mid \mathfrak{u}_1 \dots \mathfrak{u}_n \in L, v_1 \dots v_n \in L_0, \mathfrak{u}_i \in \widetilde{A}^*, v_i \in B^*\}$$

Theorem 7.14. *Visibly pushdown languages are closed under scramble with regular languages, i.e. if $L \subset \widetilde{A}^*$ is a VPL and $L_r \subset B^*$ is regular then $L \bowtie L_r$ is a VPL.*

Proof. We construct a VPA $M_\bowtie$ accepting $L \bowtie L_r$, from the machine $\widetilde{M} = (\widetilde{A}, S, \Gamma, s_0, \gamma_0, Y, \Gamma_y, \delta)$ accepting L, and $M_r = (B, S_r, s_0^r, Y_r, \delta_r)$ accepting L_r. We think of these as operating side by side so that whenever we read in letters from $\widetilde{A}$ we make transitions within $\widetilde{M}$, and likewise when we read letters from B we make transitions in M_r. To do this, we take the set of states of $M_\bowtie$ to be the product $S \times S_r$, the initial state to be (s_0, s_0^r), and the stack alphabet and hierarchical accept states to be Γ and Γ_y respectively. The transition function Δ for $M_\bowtie$ is then defined based on the symbol being read in. If in state (s, s_r) and reading in $\mathfrak{a} \in \widetilde{A}$, the transition is given by $\Delta((s, s_r), \mathfrak{a}) = (\delta(s, \mathfrak{a}), s_r)$, where the appropriate stack operation takes place as it would have in M. On reading in $b \in B$ in state (s, s_r), the transition is given by $\Delta((s, s_r), b) = (s, \delta_r(s_r, b))$. The set of accept states is $Y_\bowtie = \{(y, y_r) \mid y \in Y, y_r \in Y_r\}$, and a word is accepted if $M_\bowtie$ is in an accept state with stack contents $\chi_y \in \Gamma_y^*$. □

Remark 7.15. Note that the scramble $L \bowtie L_r$ mixes together words in a VPL with words in a regular language in the most general way possible. But as long as we are working over disjoint alphabets $\widetilde{A}$ and B we can use a similar argument as the theorem above to have more control over how we scramble words. For example, take the language of Example 7.10 $\{ww^R \mid w \in (A^{\mathfrak{c}})^*\}$ and a regular language $L_r \subset B^*$. The language of words given by $\{wuw^R \mid w \in (A^{\mathfrak{c}})^*, u \in L_r\}$ is also a VPL that is a strict subset of the full scramble $L \bowtie L_r$. The construction is similar to above except we are restricting the degree to which we may switch back and forth between the machine accepting the VPL and the FSA.

The nested structure of words was first noticed to connect with group theory when discussing the area of a word and isoperimetric functions (see section 5.2). Let $G = \langle X \mid R \rangle$ be a finitely presented group. Given a word

$w \in A^*$ such that $\overline{w} = e$, we know that we can decompose the word as $w = u_1 r_1^{\epsilon_1} u_1^{-1} \dots u_k r_k^{\epsilon_k} u_k^{-1}$. Recall that $\epsilon_i = \pm 1$, and for a word $u = a_1 \dots a_n \in A^*$ the inverse is $u^{-1} = a_n^{-1} \dots a_1^{-1}$. Words of this form have a very natural matching relation, and we have:

Theorem 7.16. *Let $G = \langle X \mid R \rangle$ be a finitely presented group. There is a VPL $L \subset \widetilde{A}^*$ over the tagged alphabet, such that the renaming function*

$$\rho : L \to L_{Dehn}$$

is a bijection, where $L_{Dehn} = \{u_1 r_1^{\epsilon_1} u_1^{-1} \dots u_k r_k^{\epsilon_k} u_k^{-1} \mid u_i \in A^*, r_i \in R\}$.

The idea of the matching relation defining this VPL comes from considering the path that $\widehat{w}$ traces out in the Cayley graph. Each $\widehat{u_i}$ is a path starting from the identity and traveling to the vertex defined by $\overline{u_i}$. The relator $r_i^{\epsilon_i}$ is a closed path which starts and ends at $\overline{u_i}$, and then we follow u_i^{-1}, which is the same as going backwards along u_i to the identity. So every letter in u_i must have a matching inverse letter in u_i^{-1}, and we must also be able to recognize relators in between this matching. To prove this we first show that a finite presentation guarantees that the language of all relators is regular, and that the language of words followed by their inverses is a VPL.

Lemma 7.17. *Let $G = \langle X \mid R \rangle$ be a finitely presented group. The relator language $L_{rel} = \{r_i^{\epsilon_i} \mid r_i \in R, \epsilon_i = \pm 1\}$ is regular.*

Proof. Let $R = \{r_1, \dots, r_m\}$ be the finite set of relators, where each $r_i \in R$ is a word over $A = X \cup X^{-1}$. Each r_i can be considered as a regular language over A having a single word; the same is true of the inverse r_i^{-1}. By Theorem 2.9, this implies that $L_{reg} = \{\bigcup_{i=1}^m r_i\} \cup \{\bigcup_{i=1}^m r_i^{-1}\}$ is regular. □

Lemma 7.18. *Let $G = \langle X \mid R \rangle$ be a finitely presented group. The language of words given by $L_{inv} = \{uu^{-1} \mid u \in (A^{\mathfrak{c}})^*, u^{-1} \in (A^{\mathfrak{r}})^*\}$ is a VPL.*

Proof. The VPA accepting L_{inv} has states $\{s_0, s_c, s_r, s_f\}$, with the initial state being s_0 and $Y = \{s_0, s_r\}$. The alphabet is $\widetilde{A}$ while the stack alphabet is A, and we set $\Gamma_y = \emptyset$. Any transition not described below is by default assigned to be a transition to the failure state s_f, from which the machine will never leave once entered. Starting in s_0 the machine may transition to s_c on reading a call, and the letter in A being tagged as a call is written to the top of the stack. In s_c the machine may read in either calls or returns. On reading a call the machine transitions back to s_c and writes the corresponding letter to the stack each time. Reading in a return, the machine transitions to s_r as long as the letter being popped off the stack is the inverse of the letter being read in. In s_r the machine may read in only returns, and again transitions back to s_r as long as the letter being read in matches the inverse of the letter being popped off the stack. □

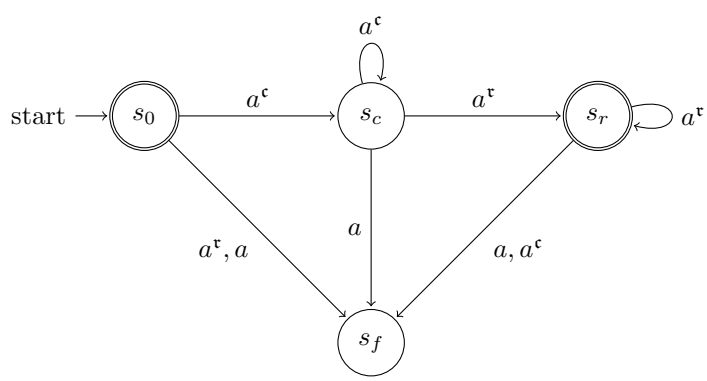


Figure 7.1: The VPL recgonizing L_{inv}; the stack is not depicted. The transitions are labelled by arbitrary elements in $\widetilde{A}$. This means, for example, that the machine will transition from s_0 to s_c on reading any element of $A^{\mathfrak{c}}$, and from s_0 to s_f on reading any element of $A \cup A^{\mathfrak{r}}$.

Proof of Theorem 7.16. Following Remark 7.15 above, we notice that $L' = \{ur^{\epsilon}u^{-1} \mid uu^{-1} \in L_{inv}, r^{\epsilon} \in L_{rel}\}$ is a particular scramble contained in $L_{inv} \bowtie$

L_{rel}. Although it is not quite the case that we are working over disjoint alphabets as in the remark, the argument still goes through since $L_{rel} \subset A^*$ and $L_{inv} \subset (A^{\mathfrak{c}} \cup A^{\mathfrak{r}})^*$. The alphabets are effectively disjoint in that we can recognize when to switch between the FSA and the VPL. We then take $L = (L')^*$ as the nested word language that maps bijectively to L_{Dehn}; it is a VPL by closure under the Kleene-star operation by Theorem 7.11. $\square$

As we have already noted, the language L_{Dehn} is a subset of the word problem $W_A(G)$, and this should lead us to ask what groups are characterized by having a visibly pushdown word problem. First we make this precise.

Definition 7.19. Let G be a finitely generated group. We say that G has a visibly pushdown word problem if there is a finite set of generators X and a VPL $L \subset \widetilde{A}^*$ such that

$$\rho : L \to W_A(G)$$

given by the renaming function is a bijection.

Remark 7.20. If G has a visibly pushdown word problem with respect to one generating set X_1, it is not clear whether it must also have a visibly pushdown word problem with respect to any other generating set X_2, and this is a fundamental question that we have not been able to resolve. The reason we choose to define the word problem with respect to the map ρ is that it highlights the subtle but important difference between word languages (such as regular and context-free) and languages of nested words (such as VPLs). Speaking colloquially we still say that the word problem is a VPL, but acknowledge that such languages consist of words accompanied by a matching relation. The advantage of working with VPLs is that matching relations are a very natural model for certain groups, as we intend to show.

The fact that VPLs are languages of nested words whereas CFLs are word languages means that we cannot identify any straighforward results following from the established research discussed in section 4.2. To see why this is

true, consider a group $G = \langle X \mid R \rangle$ with a visibly pushdown word problem $L \subset \widetilde{A}^*$. From Theorem 7.8, the underlying word language is context-free when considered over the alphabet $\widetilde{A}$. But in defining groups with a context-free word problem we only consider the word language over A, and passing to the tagged alphabet $\widetilde{A}$ changes the set of generators and therefore the word problem. For a group with a visibly pushdown word problem, a word $w \in W_A(G)$ corresponds to unique nested word $(w, \frown) \in L$. But any tagged word mapping to w under the renaming function also represents the identity, so the context-free language over $\widetilde{A}$ corresponding to L cannot be the word problem.

Despite this, we look to virtually free groups as a place to explore groups that have a visibly pushdown word problem. We have described how free groups have a context-free word problem, but the following shows how working with nested words can be a more natural model.

Example 7.21. Consider the free group on n generators, $F_n = \langle a_1, a_2, \ldots, a_n \rangle$. Any word over A that represents the identity can be reduced to the empty word by successively deleting pairs of the form xx^{-1} or $x^{-1}x$ for $x \in \{a_1, a_2, \ldots, a_n\}$, and this gives rise to a natural matching relation that defines such words. Take for example the following nested word, where the matching relation is indicated by the braces

$$\overbrace{a_1^{\mathfrak{c}} \underbrace{(a_3^{\mathfrak{c}})^{-1} \overbrace{(a_n^{\mathfrak{c}})^{-1} a_n^{\mathfrak{r}}} a_3^{\mathfrak{r}}} (a_1^{\mathfrak{r}})^{-1}}$$

Theorem 7.22. *The free group on n generators $F_n = \langle a_1, \ldots, a_n \rangle$ has a visibly pushdown word problem.*

Proof. We construct a VPA to recognize the matching relation defining words that represent the identity. The machine has states s_0 and s_f, with s_0 the initial and accept state, and s_f the fail state. The alphabet is $\widetilde{A}^*$ while the stack alphabet is A, and $\Gamma_y = \emptyset$. There are transitions from s_0 to s_f described

below, but there are no transitions out of s_f. The machine only reads in calls or returns, and transitions to the fail state on reading an internal symbol. It also transitions to the fail state on reading a return if the bottom of stack symbol γ_0 is exposed. Otherwise, the action of the machine is essentially determined by what is read in. On reading a call, that symbol is written to the stack and the machine remains in s_0. On reading a return, the letter being read in is compared to the letter on the top of the stack that is being popped off. If the pair is of the form xx^{-1} or $x^{-1}x$ for $x \in \{a_1, \ldots a_n\}$ the machine remains in s_0; if not the machine transitions to s_f. □

The question is whether this extends to virtually free groups as it does for context-free languages, and we have a partial result in that direction. We consider the symmetric group acting on the free group $F_n = \langle a_1, \ldots, a_n \rangle$ by permuting the generators. That is, we have a map $\psi : S_n \to Aut(F_n)$, where $\sigma \mapsto \psi_\sigma$ and $\psi_\sigma(a_i) = a_{\sigma(i)}$ for each generator a_i of F_n. We utilize the interpretation of the semi-direct product as the direct product with component wise multilplication adjusted by the map ψ. That is, elements are of the form (g, σ) with $g \in F_n$ and $\sigma \in S_n$, and multiplication is given by $(g_1, \sigma_1)(g_2, \sigma_2) = (g_1\psi_{\sigma_1}(g_2), \sigma_1\sigma_2)$.

Theorem 7.23. *The semi-direct product $F_n \rtimes_\psi S_n$ has a visibly pushdown word problem.*

Proof. We take the standard presentation for the free group $F_n = \langle a_1, \ldots, a_n \rangle$, and letting $m = n!$ we have the multiplication table presentation for the symmetric group $S_n = \langle \sigma_1, \ldots, \sigma_m \mid \sigma_i\sigma_j = \sigma_k \rangle$. As before we take $A = X \cup X^{-1}$ with $X = \{a_1, \ldots, a_n\}$, as well as $\Sigma = \{\sigma_1, \ldots, \sigma_m\}$. We then have the alphabet of semi-group generators for $F_n \rtimes_\psi S_n$ given by

$$\mathbb{A} = \{(a, e) \mid a \in A\} \cup \{(\epsilon, \sigma) \mid \sigma \in S_n\}$$

.

The construction is very similar to the machine accepting the scramble $W_A(F_n) \bowtie W_\Sigma(S_n)$ described in Theorem 7.14. We use the VPA described in Theorem 7.22 which we denote M_{free}, and the FSA for the finite group S_n outlined in Example 4.3 which we denote M_{sym}. Letting (s, σ) be an arbitrary state of our machine (with $s \in \{s_0, s_f\}, \sigma \in S_n$), we examine what happens on reading a letter from $\mathbb{A}$. On reading (ϵ, σ') the state is updated to $(s, \overline{\sigma\sigma'})$, where the second component keeps track of tranistions in M_{sym}, which we know corresponds to group multiplication in S_n. If (a, e) is designated as a call, the first component is updated as it would be in M_{free} to s_0 or s_f, but we write $\sigma(a)$ to the stack instead of a. Similary if (a, e) is a return we compare $\sigma(a)$ to the top of the stack and make the appropriate transition in the first component. Acceptance is by state (s_0, e) and having an empty stack.

□

7.3 Conclusion

In summary, this thesis has accomplished two important things. We have presented a framework for surveying topics in group theory that has allowed us to draw out important connections between group theory and formal languages. In addition, introducing regular languages of nested words into group theory has given a unique perspective on the structure of words representing the identity in F_n. The matching relation inherent in such words makes use of both the linear and hierarchical structure that was precisely the motivation for Alur and Madhusudan in introducing nested words. By combining this with known results about the word problem of finite groups and examining the relationship between VPLs and regular languages, we arrive at Theorem 7.23, along with the potential for some interesting further work.

We now touch briefly on a few possible directions for future research that

arise naturally from this thesis. The first question is whether there is a nice characterization for the class of groups that admit a VPL word problem that is analogous to the result of Theorem 4.10 of Muller and Schupp [13]. For instance, is it true that every virtually free group admits a VPL word problem? A fundamental question related to this is whether the property of having a VPL word problem can be made independent of the generating set. If so, then it would be interesting to know whether this property also satisfies the stronger condition of being a quasi-isometric invariant of the group (cf. section 3.2 and remark 7.20). One can also ask which aspects of group theory discussed earlier in this thesis can be developed for visibly pushdown languages. For example, $\mathscr{P}$ groups are essentially defined by using context-free languages in place of regular ones in the formulation of automatic structures. What kind of structures are obtained if one instead using VPLs? This same approach could be relevant for investigating combings of groups with regular languages of nested words.

References

[1] D. Allcock, *An isoperimetric inequality for the Heisenberg groups,* Geometric and Functional Analysis (2) **8** (1998), 219–233.

[2] R. Alur and P. Madhusudan, *Adding Nested Structure to Words,* JACM. (3) **56** (2009), 1–43.

[3] A.V. Anisimov, *Group Languages,* Kibernetika. **4** (1971), 18-24.

[4] G. Baumslag, *Wreath products and finitely presented groups,* Math. Zeitschr. **75** (1961), 22–28.

[5] M. Bridson and R.H. Gilman, *Formal language theory and the geometry of 3-manifolds,* Commentarii Mathematici Helvetici. (1) **71** (1996), 525–555.

[6] G. Baumslag, M. Shapiro and H. Short, *Parallel poly-pushdown groups,* J. Pure and Applied Algebra. **140** (1999), 209–227.

[7] M.J. Dunwoody, *The accessibility of finitely presented groups,* Inventiones Mathematicae (3) **81** (1985), 449–457.

[8] D.B.A. Epstein, J.W. Cannon, D.F. Holt, S.V.F. Levy, M.S. Paterson and W.P. Thurston, *Word Processing in Groups,* Jones and Bartlett, Boston, MA., 1992.

[9] R.H. Gilman, D.F. Holt and S. Rees, *Combing Nilpotent and Polycyclic Groups,* IJAC. (2) **9** (1999), 135–155.

[10] D.F. Holt, S. Rees, C.E. Rover and R.M. Thomas, *Groups with Context-Free Co-word Problem,* London Math Soc. (2000), 1–21.

[11] J.E. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation,* Addison-Wesley, Reading, MA., 1979.

[12] L. Mosher, *Mapping class groups are automatic,* Annals of Mathematics **142** (1995), 303–384.

[13] D. Muller and P. Schupp, *Groups, the Theory of Ends, and Context-Free Languages,* J. Comp. and System Sci. **26** (1983), 295–310.

[14] J. Nielsen, *The commutator group of the free product of cyclic groups,* Mat. Tidskr. **B** (1948), 49–56.

[15] J. Porti, *Geometrization of three manifolds and Perelman's proof,* RACSAM (1) **102** (2008), 101–125.

[16] S. Rees, *Hairdressing in groups: a survey of combings and formal languages,* Geom. and Top. Monographs. **1** (1998), 493–509.