The Word Problem For Groups And Formal Language Theory

Fabricio Dos Santos
McGill University

Abstract

The word problem of a finitely generated group G with presentation $\langle S \mid R \rangle$ is defined, from a language theoretic perspective, as the set of words on S that represent the identity in G. This survey paper studies the connections between group theoretic properties of a group, and language theoretic properties of its word problem. In particular, a partial classification of finitely generated groups is provided based on the nature of their word problem. Finally, a short introduction on the conjugacy problem for groups is given, along with analogous classification results.

1. Introduction

In this paper, we describe a fundamental problem lying at the intersection of combinatorial group theory and computability theory: the word problem for groups. Given a finitely generated group G with presentation $\langle S \mid R \rangle$, this is the problem of deciding whether a word on S represents the identity in G. This question was first posed by Max Dehn in 1911 [3], and has since been extensively studied algorithmically. In particular, it was proven by Pyotr Novikov in 1955 [8] and William Boone in 1958 [2], that the word problem for finitely presented groups is, in general, undecidable. In this paper, we take a different approach and study the word problem for groups from a language theoretic perspective. More precisely, we define the word problem of G as the set of words on S which represent the identity in G. This allows us to provide a classification of finitely generated groups based on the language theoretic properties of their word problem. We also introduce another question originally posed by Max Dehn, the conjugacy problem for groups, and briefly study it from this perspective.

The reader is only expected to have a basic familiarity with group theory, and some exposure to computability theory. The first two sections of this paper provide the necessary background to study the word problem of a group. Section 2 consists of the necessary group theoretic background, such as free groups and presentations. Section 3 consists of a short survey of formal language theory, where different families of languages are discussed, along with their respective automata.

A more experienced reader may skip the first two sections and start with section 4. This section introduces the word problem for groups, and contains the main theorems of this paper. In particular, we prove in theorem 4.1 that a group has regular word problem if and only if it is finite. Theorem 4.5 asserts that a group has context-free word problem if and only if it is virtually free. Finally, some results regarding the conjugacy problem for groups are also presented in section 5. In particular, we prove in theorem 5.1 that a group has synchronously regular conjugacy problem if and only if it is finite, and in theorem 5.3 that if a group is virtually cyclic then it has synchronously one counter conjugacy problem.

2. Free groups and presentations

As one might expect, in order to discuss word problems for groups, we first need to formalize the concept of a word in a group. This notion comes from defining a group by a tuple consisting

of a set of generators and relations between these generators. A group described in this way is said to be described by a *group presentation*, and words for a given presentation of a group are simply concatenations of generators and their inverses. In this section, we introduce the essential concept of a *free group*, and show how these groups allow us to formalize the idea of group presentations.

2.1. Free groups

Free groups are a fundamental concept in many areas of mathematics and computer science. Essentially, a free group is a group that is generated by a set of elements without any relations between them. Free groups are essential to the definition of group presentations, and thus provide a good framework for studying groups and their properties. We will now describe how to construct a free group given a set, and discuss some special properties of these groups.

We start by considering a set S of symbols which will serve as a basis for the free group we are constructing. We then need to choose another disjoint set S^{-1} such that there exists a bijection from S to S^{-1}. That is, to each symbol $s \in S$, we assign some symbol $s^{-1} \in S^{-1}$, called the *formal inverse* of s. In particular, we require that if s^{-1} is the formal inverse of s, then s is the formal inverse of s^{-1}. We then define the set $S^{\pm 1} = S \cup S^{-1}$, which will serve as an alphabet for our words.

A *word* w on S is a finite sequence of symbols denoted by $w = s_1...s_n$, where each $s_i \in S^{\pm 1}$. The number n is called the *length* of the word w, and is also denoted by $|w|$. In particular, the word with $n = 0$ is called the *empty word*, and is usually denoted by ϵ. We say that a word $w = s_1...s_n$ is *reduced* if whenever $s_i \in S$, neither s_{i-1} nor s_{i+1} are the formal inverse of s_i. In other words, w contain no subwords of the type ss^{-1} or $s^{-1}s$.

To construct a free group with basis S, we need to describe the notion of *reduction*, which transforms an arbitrary word into a reduced one. An *elementary reduction* of a word w consists of deleting a subword of the type $s^{-1}s$ or ss^{-1} where $s \in S^{\pm 1}$. For example, an elementary reduction transforms a word $abb^{-1}a$ into a new word aa. A *reduction* of a word w is a sequence of elementary reductions starting at w and ending at a reduced word. In particular, we can prove that the order of such elementary reductions does not matter, as they result in the same reduced word.

Lemma 2.1. *There is only one reduced form of a given word w on S.*

Proof. We can use induction on the length of w. If w is already reduced, then there is nothing to show. So, let $|w| = n$ and assume that every word of length less than n has an unique reduced form. Let $x \in S^{\pm 1}$ with formal inverse x^{-1} and suppose $w = uxx^{-1}v$ where u and v are words on S. Let $\overline{w}$ be a reduced form of w. We claim that $\overline{w} = \overline{uv}$, where $\overline{uv}$ denotes the unique reduced form of uv. First, suppose that this occurrence of xx^{-1} is cancelled at some point of the reduction. Then, we can perform this elementary reduction first and we get that $\overline{w} = \overline{uv}$, which is unique by the inductive hypothesis. Thus, assume this specific pair is not cancelled during the reduction. Then, since xx^{-1} cannot be a subword of $\overline{w}$, we must have one of two cases. If $w = u_1x^{-1}xx^{-1}v_1$, then $u = u_1x^{-1}$ and $v = v_1$, and thus $\overline{w}$ is given by the reduced form of $u_1x^{-1}v_1$. But $u_1x^{-1}v_1 = uv$, and thus $\overline{w} = \overline{uv}$. If $w = u_2xx^{-1}xv_2$, then $u = u_1$ and $v = xv_2$, and thus $\overline{w}$ is given by the reduced form of u_2xv_2. But $u_2xv_2 = uv$, and thus $\overline{w} = \overline{uv}$. In either case, we have $\overline{w} = \overline{uv}$, and it follows by the inductive hypotheses that $\overline{w}$ is unique. □

Now, for a word w on S, denote by $\overline{w}$ the unique reduced form of w. Let $F(S)$ be the set of all reduced words on S, and define multiplication of two words $u, v \in F(S)$ by $u \cdot v = \overline{uv}$, that is we concatenate u with v and reduce this word. Then, we get the following result.

Theorem 2.2. *The set $F(S)$ forms a group with respect to multiplication defined as above.*

Proof. We first check that the operation $\cdot$ is associative. Let $u, v, w \in F(S)$. Then, we have $\overline{\overline{uv}w} = \overline{u\overline{vw}}$ since the order of elementary reductions does not matter. Since the empty word $\epsilon \in F(S)$, and we have $\epsilon \cdot w = w \cdot \epsilon = w$ for all $w \in F(S)$, it is the identity element of the group. Finally, we show that every $w = s_1...s_n \in F(S)$ has an inverse element. Consider $w^{-1} = s_n^{-1}...s_1^{-1}$. Then, $w \cdot w^{-1} = \overline{s_1...s_n s_n^{-1}...s_1^{-1}} = \epsilon$. Hence, $F(S)$ is a group with respect to the operation $\cdot$ defined above. □

The group $F(S)$ is called the *free group* on the set S. A group F is *free* if there is some set S such that $F \cong F(S)$. In this case, we call S a set of *free generators* (or *free basis*) of F.

Another way of thinking of free groups with a basis S is to consider instead the set of equivalence classes of words on S, where two words are equivalent if their reduced form is the same word. In this setting, multiplication works in the same way by concatenating the representatives and reducing the result to find the corresponding equivalence class.

We can now state an essential property of free groups, called the *universal property* of free groups.

Theorem 2.3 (Universal Property of Free Groups)**.** *Let G be a group, S a set and $f : S \to G$ a map. Then, there is an unique group homomorphism $\varphi : F(S) \to G$ such that the following diagram commutes:*

$$\begin{array}{ccc} S & \hookrightarrow & F(S) \\ & \searrow^{f} & \downarrow{\scriptstyle \varphi} \\ & & G \end{array}$$

Proof. Define the map φ by $\varphi(s_1^{\epsilon_1} s_2^{\epsilon_2}...s_n^{\epsilon_n}) = f(s_1)^{\epsilon_1} f(s_2)^{\epsilon_2}...f(s_n)^{\epsilon_n}$, where each $s_i \in S$ and $\epsilon_i \in \{\pm 1\}$. Note that this map is unique since any other homomorphism will also be equal to f on the generating set S. We can now prove that φ is a homomorphism. Given two words $u = s_1^{\epsilon_1}...s_n^{\epsilon_n}$ and $w = r_1^{\varepsilon_1}...r_m^{\varepsilon_m}$ in $F(S)$, we have two cases. If uw is reduced, then

$$\begin{aligned} \varphi(s_1^{\epsilon_1}...s_n^{\epsilon_n} r_1^{\varepsilon_1}...r_m^{\varepsilon_m}) &= f(s_1)^{\epsilon_1}...f(s_n)^{\epsilon_n} f(r_1)^{\varepsilon_1}...f(r_m)^{\varepsilon_m} \\ &= \varphi(s_1^{\epsilon_1}...s_n^{\epsilon_n})\varphi(r_1^{\varepsilon_1}...r_m^{\varepsilon_m}) \end{aligned}$$

Otherwise, there exists some word $v \in F(S)$ such that $u = u'v$ and $w = v^{-1}w'$ where u', w' are reduced. Then, $\overline{uw} = u'w'$ and we have

$$\varphi(\overline{uw}) = \varphi(u'w') = \varphi(u')\varphi(w')$$

by the first case. Since v, u' and w' are reduced, we have that

$$\begin{aligned} \varphi(u) &= \varphi(u'v) = \varphi(u')\varphi(v) \\ \varphi(w) &= \varphi(v^{-1}w') = \varphi(v)^{-1}\varphi(w') \end{aligned}$$

and hence

$$\varphi(u)\varphi(w) = \varphi(u')\varphi(v)\varphi(v)^{-1}\varphi(w') = \varphi(u')\varphi(w')$$

Therefore, $\varphi(\overline{uw}) = \varphi(u)\varphi(w)$ and we are done. □

Corollary 2.3.1. *The group $F(S)$ is unique up to isomorphism, which is the identity map on the set S.*

The corollary above implies that any group F satisfying the universal property is isomorphic to the free group $F(S)$ we constructed. This allows us to identify a free group freely generated by S with the group $F(S)$. We can thus define the *rank* of a free group F to be the cardinality of its generating set. It can be easily proven that two free groups of same rank are isomorphic. Hence, we can denote by F_n a free group generated by a set S with $|S| = n$.

Another important result on free groups which we will need later is stated in the following theorem. The proof, however, is not given since it requires some more advanced machinery.

Theorem 2.4 (Schreier's Theorem)**.** *Subgroups of free groups are free.*

It is worthy to mention that, unlike the behavior of dimensions of vector spaces in linear algebra, a subgroup of a free group F_n can have rank strictly greater than n. One such example is the subgroup generated by the elements $\{a^n b^n : n \in \mathbb{N}\}$ of the free group F_2 with basis $\{a, b\}$. In particular, it can be shown that this subgroup has infinite rank even though F_2 has finite rank 2.

One last group theoretic definition we will need is that of *virtually* P groups, where P is a property. We say that a group G is virtually P if G has a subgroup of finite index in G which is P. Note that if a group has a property P, then it is also virtually P, since the group is a subgroup of index 1 of itself. Also, any finite group G is virtually P, where P is any property held by a subgroup of G, since all subgroups will trivially have finite index in G. In particular, we will discuss two special cases of such groups: virtually free groups and virtually cyclic groups. One way of constructing a virtually P group is by taking the direct product of a finite group H with a group H' which is P. The resulting group G is virtually P since the subgroup given by $\{1_H\} \times H'$ has finite index $|H|$ in G, and is P since it is isomorphic to H'.

2.2. Presentations

Group presentations are a method of specifying a group in terms of words over a generating set. More precisely, a *presentation* $P = \langle\, S \,|\, R \,\rangle$ is a pair consisting of a set S of symbols called *generators* and a set R of words on S called *relators*.

Given a group G, we define the *normal closure* of a subset $A \subseteq G$ to be the smallest normal subgroup of G containing A, which we denote $N(A)$. In other words, $N(A)$ is characterised by the following three properties:

1. $A \subseteq N(A)$
2. $N(A) \trianglelefteq G$
3. If $H \triangleleft G$ and $A \subseteq H$ then $N(A) \leq H$

In particular, $N(A)$ is generated by the set of all conjugates of elements of A, that is

$$N(A) = \left\langle \{gag^{-1} : a \in A, g \in G\} \right\rangle$$

This definition is general, but we are mostly interested in the case of free groups.

We can now use this concept to associate to a presentation $\langle\, S \,|\, R \,\rangle$ a group G. Given a set S and a subset $R \subseteq F(S)$, we say that a group G has presentation $\langle\, S \,|\, R \,\rangle$ if it is isomorphic to the quotient $F(S)/N(R)$. If $u, v \in F(S)$ are words on S, we write $u =_G v$ if they are representatives of the same coset of $N(R)$ in $F(S)$. Note that if $r \in R$, then $r \in N(R)$ and thus $r =_G 1$. In other words, the elements of R describe the identity of G. Going forward, if a group G has presentation $\langle S|R\rangle$, we will write $G = \langle\, S \,|\, R \,\rangle$.

A presentation $P = \langle\, S \,|\, R \,\rangle$ is said to be *finitely generated* if S is a finite set and to be *finitely related* if R is a finite set of words. If both S and R are finite, P is said to be a *finite presentation*. Likewise, we say that a group G is finitely generated if it admits a finitely generated presentation.

Given $S = \{a_1, a_2, ...\}$ and $R = \{r_1, r_2, ...\}$, we will usually use the notation $\langle\, a_1, a_2, ... \,|\, r_1, r_2, ... \,\rangle$, or $\langle\, a_1, a_2, ... \,|\, r_1 = 1, r_2 = 1, ... \,\rangle$ in which case the equations $r_i = 1$ are called *relations*. We can also use the notation $\langle\, a_1, a_2, ... \,|\, u_1 = v_1, u_2 = v_2, ... \,\rangle$ if $r_i = u_i v_i^{-1}$.

We can now give examples of group presentations which will be discussed or mentioned later.

1. The free group $F(S)$ with basis S has a presentation $P = \langle\, S \,|\, \emptyset \,\rangle$. Indeed, since the generators have no relations between them, the set R is empty and the only reduced word representing the identity is the empty word ϵ.

2. The finite cyclic group C_n of order n has a presentation $P = \langle\, x \,|\, x^n \,\rangle$. Indeed, we can see that $G = \langle\, x \,|\, x^n \,\rangle$ has n elements $\{1, x, ..., x^{n-1}\}$ and $x^n =_G 1$.

3. The group $\mathbb{Z}^2$ has presentation $P = \langle\, x, y \,|\, xyx^{-1}y^{-1} \,\rangle = \langle\, x, y \,|\, xy = yx \,\rangle$. Note that $xyx^{-1}y^{-1} = 1 \implies x(yx^{-1})y^{-1} = 1 \implies yx^{-1} = x^{-1}y$, and in a similar way $y^{-1}x = xy^{-1}$ and $x^{-1}y^{-1} = y^{-1}x^{-1}$. Thus, x, y and their inverses commute. Given a word w on these generators, we can thus write w in an unique form $x^n y^m$ by commuting x's and y's. Note that here x^n means n x's concatenated together (or $|n|$ x^{-1}'s if $n < 0$). We can thus define an isomorphism from $G = \langle\, x, y \,|\, xyx^{-1}y^{-1} \,\rangle$ to $\mathbb{Z}^2$ by mapping $w = x^n y^m$ to $(n, m) \in \mathbb{Z}^2$. This group is called the *free abelian group* of rank 2.

4. More generally, the group presentation of the free abelian group of rank n is $P = \langle\, S \,|\, R \,\rangle$, where $|S| = n$ and $R = \{xyx^{-1}y^{-1} : x, y \in S, x \neq y\}$. Hence, to get a presentation of this group we enforce that every generator commutes with the others.

5. Let $G = \{g_1, ..., g_n\}$ be finite with identity element g_1. The *multiplication table presentation* of G is the finite presentation $\langle\, g_1, ..., g_n \,|\, ..., g_i g_j = g_k, ... \,\rangle$, in which for each generator g_i, we add the relations $g_i g_j = g_k$, for $1 \leq j \leq n$, where g_k is the product of g_i with g_j.

3. Formal languages

So far, we have discussed the concepts of free groups and group presentations. Both require the notion of words in groups, which are strings of letters coming from a fixed alphabet. Our next step involves building some theory so that we can formally study sets of these words. This will be done through the study of the theory of formal languages.

Formal language theory is an essential subject in both theoretical computer science and several branches of mathematics. It involves the study of sets of words over a given alphabet, which we refer to as *languages*. A hierarchy of languages, known as the *Chomsky hierarchy*, is established based on the expressive capacity of these languages. This is particularly interesting since it allows us to examine the word problem of a group as a formal language.

It is also important to mention that formal language theory is very closely related to *computability theory*, as languages are characterized by the computational power of the "abstract machines" that recognize them.

We can start the discussion on formal languages by defining the following concepts. We call a set of symbols Σ an *alphabet*, and denote by Σ^* the set of all *words* over Σ. Here, a word is a finite sequence of elements of Σ, and this includes the empty word ϵ. A *language* L is simply a subset $L \subseteq \Sigma^*$. We will study the expressive power of languages through the use of abstract machines, called *automata*, which can recognize words pertaining to these languages.

3.1. Regular languages

The first class of languages we discuss is the one of *regular languages* (RL). This is the simplest type of languages we consider, and these are exactly the languages recognized by finite state automata.

For a fixed alphabet Σ, a *deterministic finite automaton* (DFA) is a 4-tuple $M = (S, s_0, \delta, F)$. Here, S is a set called the *states* of M, $s_0 \in S$ is a *start state*, $F \subseteq S$ is a set of *accept states* and $\delta : S \times \Sigma \to S$ is a *transition function*. A DFA M reads a word $w \in \Sigma^*$ letter by letter, starting in state s_0 and changing states according to δ. We can represent a DFA graphically as in the example shown in figure 1.

In this example, the alphabet is $\Sigma = \{a, b\}$, we have 3 states $S = \{s_0, s_1, s_2\}$ where s_0 is the start state and only s_2 is an accept state. In addition, the labeled arrows represent the transition function δ. When reading the word *abb*, for example, the machine would start in

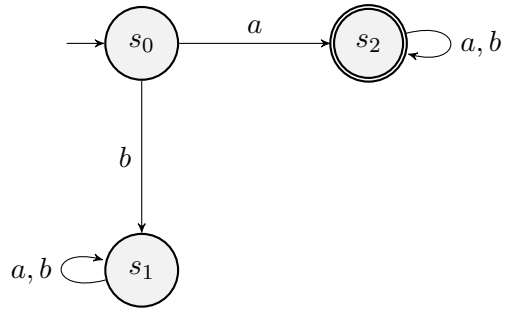

Figure 1: DFA to recognize words starting in a

s_0, change states to s_2 and then stay in s_2. This word would be accepted because when the machine is done reading aab, it is in an accept state. It can be easily seen that this DFA only accepts words starting with the letter a and followed by any other word (potentially empty).

In order to describe the language recognized by a DFA, we can first inductively define $\delta^* : S \times \Sigma^* \to S$ for any $a \in \Sigma$, $w \in \Sigma^*$ and $q \in S$ by

$$\delta^*(q, aw) = \begin{cases} \delta(q, a) & \text{if } w = \epsilon \\ \delta^*(\delta(q, a), w) & \text{otherwise} \end{cases}$$

Then, the language L accepted by a DFA M is

$$L = L(M) = \{w \in \Sigma^* : \delta^*(s_0, w) \cap F \neq \emptyset\}$$

That is, a DFA accepts all words for which the final state after reading the word is one of the accept states. Hence, we can say a language L is *regular* if there exists a DFA M such that $L = L(M)$.

It is often convenient to work with *nondeterministic finite automata* (NFA) instead of DFA. This type of machine consists of a DFA which allows for nondeterminism (δ maps (q, a) to a set of states) and for ϵ-moves. These moves are transitions from one state of the machine to another without reading any input letter. More precisely, given a state q and a set of states Q, an ϵ-move is a transition $\delta(q, \epsilon) = Q$, meaning that at state q the machine can change to any state in Q without reading any input letters. It can be shown that any NFA can be described by a (potentially more complicated) DFA, and thus a language is regular if and only if it is recognized by a NFA.

A classical example of a language which is not regular is $L = \{a^n b^n : n \in \mathbb{N}\}$, and this can be proven using the *pumping lemma* for regular languages (see [9]). Essentially, a DFA or NFA cannot keep track of how many letters it has already seen at a point in time, which prevents the machine from matching the number of a's and b's. This motivates the introduction of a stack, and this defines the family of languages we see next.

3.2. Context-free languages

Next, we discuss a more general class of languages, the one of *context-free languages* (CFL). These languages can be described by *context-free grammars*, or by finite state automata accompanied by a stack. We will focus on the latter as it will prove more useful to us later. In particular, we use the same convention as in [7].

A *nondeterministic pushdown automaton* (PDA) is a 7-tuple $M = (Q, \Sigma, Z, \delta, q_0, z_0, F)$. Here, Q is a finite set of states, Σ is a finite *input alphabet*, Z is a finite *stack alphabet* with $Z \supseteq \Sigma$, $q_0 \in Q$ is the *initial state*, $z_0 \in Z \cup \{\epsilon\}$ is the *start symbol*, $F \subseteq Q$ is the set of *accept states* and δ is the *transition function*. For us, δ is a function from $Q \times (\Sigma \cup \{\epsilon\}) \times (Z \cup \{\epsilon\})$ to finite subsets of $Q \times Z^*$.

The interpretation of $(q_i, \zeta_i) \in \delta(q, a, z)$ is that when the PDA is in state q reading the input symbol a and z is the top symbol on the stack, then M can change to state q_i, replace z by ζ_i and move the reading head one square to the right. In particular, if $a = \epsilon$ this is called an ϵ-move and the machine transitions states and modifies the stack without reading any input letters. We consider ζ_i as being placed on the stack from left to right so that the rightmost symbol of ζ_i (if $\zeta_i \neq \epsilon$) becomes the top symbol of the stack.

We write $M \vdash_w^* (q, \zeta)$ if it is possible for M to be in state q with ζ written on the stack after reading the input w. We can then define the language L accepted by M to be

$$L = L(M) = \{w \in \Sigma^* : M \vdash_w^* (q, \epsilon) \text{ for some } q \in F\}$$

The languages accepted by PDA are called *context-free languages*. However, unlike the case of DFA and NFA, enforcing determinism on a PDA gives us *deterministic context-free languages* (DCFL), which are a proper subset of context-free languages.

One example of a CFL is the language $L = \{a^n b^n : n \in \mathbb{N}\}$. In particular, a PDA for L can use its states to verify that the order of a's and b's is appropriate, and use the stack to verify that the numbers match. This can be done by first pushing the a's into the stack, and then deleting one a for every occurrence of b. Hence, a word is accepted if and only if the machine ends in some accept state with empty stack.

However, this procedure does not work for $L' = \{a^n b^n c^n : n \in \mathbb{N}\}$, since we cannot match the number of c's with those of a's and b's. In particular, when the machine is done verifying that the number of a's is equal to the number of b's (by deleting a's from the stack), the machine has empty stack and has already read all the b's, and thus it cannot remember how many b's and a's it has read. Hence, L' is not a CFL, which again can be proven formally by using the *pumping lemma* for context-free languages (see [9]).

If we restrict the stack of a PDA to contain only the start symbol and one extra symbol, we get the class of languages called *one counter languages* (OCL). One counter languages are a proper subset of context-free languages, and will also be of interest to us later. In particular, the language $L = \{a^n b^n : n \in \mathbb{N}\}$ is a OCL, but $L' = \{a^n b^m a^m b^n : n \in \mathbb{N}\}$ is not (even though it is a CFL), since a PDA for L' would need two stack symbols (one for matching $b^m a^m$ and the other for matching $a^n b^n$).

3.3. Recursively enumerable languages

The final and more general class of languages we discuss is the one of *recursively enumerable languages*. We say that a language $L \subseteq \Sigma^*$ is recursively enumerable (RE) if there exists an algorithm that lists all the members of L, and only the members of L, in some arbitrary order.

The notion of an algorithm that lists the members of L can be formalized through the concept of a *Turing machine* (TM), and we present the definition given in [9]. A TM is essentially composed of a set of states, a "semi-infinite" tape made of cells, and a tape head which can read one cell of the tape at a time. The machine starts by reading a tape with a word w written on it, and proceeds to transition states and modify cells according to the word w. When reading a word, A TM may accept, reject or never halt. This is contrary to DFA and PDA, and gives TM's more computational power.

More formally, a Turing machine is a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, s, a, r)$ where Q is a finite set of states, Σ is a finite set of input symbols, $\Gamma \supseteq \Sigma$ is a finite set of tape symbols containing the blank symbol $\sqcup \in \Gamma \setminus \Sigma$, $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function, $s \in Q$ is the start state, $a \in Q$ is the accept state and r is the reject state. The notation $\delta(q, b) = (q', c, L)$ means that if the machine is in state q and reads the symbol b, then it changes to state q', erases b and replaces it with c, and moves one step to the left (or right if we have R).

A *configuration* is a description of the machine at an instant of time. It can be represented by a string $uaqbv$ where $u, v \in \Gamma^*$, $a, b \in \Gamma$ and $q \in S$. This string can be interpreted as follows:

the machine is in state q reading b and the word $uabv$ is written on the tape (the tape contains only blanks following the last symbol of v). An accept configuration is any configuration with state a in it, and a reject configuration is any configuration with state r in it. A configuration which has a or r in it is called a halting configuration.

We say that M accepts $w \in \Sigma^*$ if there is a finite sequence of configurations $c_1, ..., c_n$ where c_1 is the start configuration sw, each c_i yields c_{i+1} through the transition function δ, and c_n is an accept configuration. Similarly, we say that M halts on w if there is a finite sequence of configurations such that the first two conditions above are satisfied, and the last configuration is a halting configuration. The language accepted by a TM M is thus $L(M) = \{w \in \Sigma^* : M \text{ accepts } w\}$.

Hence, we can formally define a recursively enumerable language L as a language such that there exists a TM M with $L = L(M)$. In particular, if M halts on every input, L is called *recursive*, or *decidable*. It is usually complicated to work directly with Turing machines, so often the description of a procedure to enumerate elements of a set will be enough to show that it is recursively enumerable.

We can finish the discussion on formal languages by relating all the classes of languages we have defined. In particular, we have:

$$\text{RL} \subsetneq \text{OCL} \subsetneq \text{CFL} \subsetneq \text{RE}$$

In the following section, we will provide a characterization of groups based on which of the above classes its word problem falls into.

4. The word problem for groups

We are now ready to discuss the main topic of this paper. Given a group defined by a presentation, it is natural to ask what we can learn about this group by just looking at its presentation. One of the most important questions of this type, posed by Max Dehn in 1911 [3], is the following.

The Word Problem: Let G be a finitely generated group given by the presentation $\langle S \mid R \rangle$.
Can we decide whether a given word in S represents the identity in G?

The question above is presented as a decision problem, and is often studied from an algorithmic perspective. We will, however, take a different approach and study this problem from a language theoretic point of view. Given a group G as described above, we will consider the set of all words in S that are trivial in G, and investigate the properties of this set as a formal language.

The *word problem* $\text{WP}(G, \Sigma)$ of a finitely generated group G with presentation $\langle S \mid R \rangle$ is defined as the following set:

$$\text{WP}(G, \Sigma) = \{w \in \Sigma^* : w =_G 1\}$$

where $\Sigma = S^{\pm 1}$. This set depends on the given presentation of G, more specifically on its generating set. Most of the times, however, the choice of generators is clear and we just write $\text{WP}(G)$ for the word problem of G.

Given a group G, the first question about the word problem of G we investigate is that of knowing what are the conditions on G that make $\text{WP}(G)$ a regular language. This question was first answered by Anisimov in 1971 [1], who proved that the groups with a regular word problem are exactly the finite groups. This result is known as Anisimov's theorem, and the proof we give closely follows the one in [7].

Theorem 4.1 (Anisimov's Theorem)**.** *A group has regular word problem if and only if it is finite.*

Proof. Let $G = \{g_1, ..., g_n\}$ be a finite group with identity element g_1, and consider the multiplication table presentation $\langle\, g_1, ..., g_n \,|\, ..., g_i g_j = g_k, ... \,\rangle$ of G. Using this presentation, we can build a DFA $M = (S, s_0, \delta, F, \Sigma)$ which accepts the word problem of G. The machine M has input alphabet $\Sigma = \{g_1, ..., g_n\}$, and a set of states $S = \{1, ..., n\}$ where state i corresponds to the generator g_i. The starting state is $s_0 = 1$, and the transition function δ maps (i, g_j) to state k if $g_i g_j = g_k$. Finally, the set of accept states F consists only of $\{1\}$. When reading a word $w = g_{i_1} ... g_{i_m}$, M will change states by multiplying the current generator with the next, and thus w is accepted by M if and only the product of all generators $g_{i_1}, ..., g_{i_m}$ is equal to g_1.

Now, let $G = \langle\, S \,|\, R \,\rangle$ be any infinite group with regular word problem. Note that this implies that G is finitely generated, as we need a finite alphabet to obtain a regular language. Then, there exists a DFA M which accepts the word problem of G with input alphabet $\Sigma = S^{\pm 1}$. We claim that there exist words w on $S^{\pm 1}$ of arbitrarily long length such that no nonempty subword of w represents the identity of G. That is, we say that for all $n \in \mathbb{N}$ there exists some w with $|w| \geq n$ and such that no nonempty subword of w represents the identity of G. This is the case since if there was a limit n on the length of such w, then we could reduce any word w' (by omitting trivial subwords) to a word of length less than n, making G finite (as it would only contain elements represented by words of length less than n). Therefore, we can pick a word w such that the length of w is greater than the number of states in M, and w has no nonempty subword which represents the identity of G. If M begins reading w, then M must be in the same state after reading two distinct initial segments, say u and v, of w. However, the word uu^{-1} gets accepted by M since $uu^{-1} = 1$ in G, but the word uvu^{-1} gets rejected since $v \neq 1$ in G. This is a contradiction, and so G must be finite. □

The first example we give is that of the cyclic group of order 3, denoted by C_3. The multiplication table presentation of C_3 is given by

$$\left\langle\, 1, x, x^2 \,|\, 1 \cdot 1 = 1, 1x = x, x1 = x, 1x^2 = x^2, x^2 1 = x^2, xx = x^2, xx^2 = 1, x^2 x = 1, x^2 x^2 = x \,\right\rangle$$

We can therefore set $\Sigma = \{1, x, x^2\}$ and give the following DFA for the word problem of C_3:

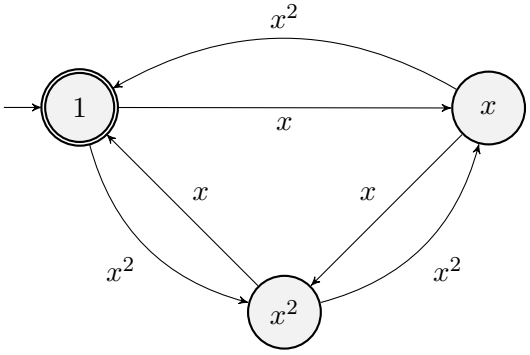

Figure 2: DFA to recognize WP(C_3)

As we can see from the previous example, working with the multiplication table of a group can become very impractical. For a finite group G, we can choose instead a simpler finite presentation $\langle\, S \,|\, R \,\rangle$ and use the multiplication table of G to build a smaller DFA. In particular, we let the input alphabet of this DFA be $S^{\pm 1}$, and let the states be the elements of G, with the identity being the start state and only accept state. Then, for each state g_i corresponding to $g_i \in G$, we add the transitions $\delta(g_i, s) = g_j$ and $\delta(g_i, s^{-1}) = g_k$ for all $s \in S$ where $g_i s = g_j$ and $g_i s^{-1} = g_k$ in G. If we interpret this DFA as a directed graph (by letting the states be vertices and each transition be a directed edge), this is known as the *Cayley digraph* of G with respect to the generating set S.

Consider the following example of this construction. We let $G = \mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/3\mathbb{Z}$ and consider the presentation $\langle x, y \,|\, x^2, y^3, x^{-1}y^{-1}xy \rangle$. The input alphabet is $\Sigma = \{x, y, x^{-1}, y^{-1}\}$. For clarity, we do not include arrows for the letters x^{-1} and y^{-1} as we can read backwards the arrows for x and y respectively.

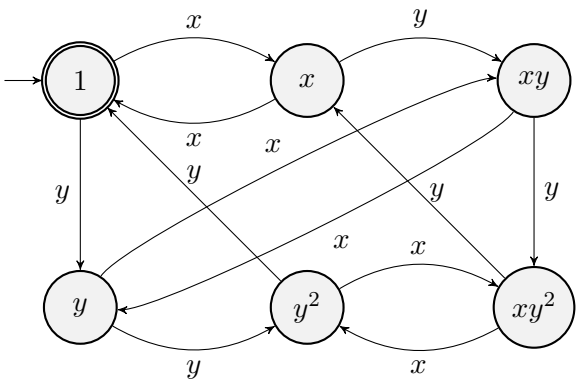

Figure 3: DFA to recognize $\mathrm{WP}(\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/3\mathbb{Z})$

The last example we provide is of the dihedral group D_3 of symmetries of a triangle. We can use the presentation $\langle r, s \,|\, s^2, r^3, (rs)^2 \rangle$ and the input alphabet $\Sigma = \{r, s, r^{-1}, s^{-1}\}$.

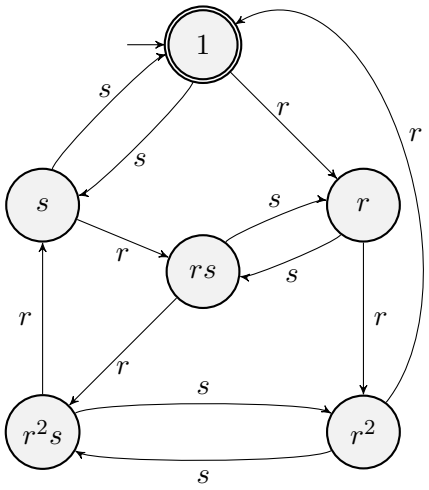

Figure 4: DFA to recognize $\mathrm{WP}(D_3)$

We now turn to the discussion of groups with context-free word problem. In particular, these are groups G such that there exists a push down automaton which accepts the word problem for a given presentation of G. Note that since regular languages are also context-free, finite groups also fall into this category. However, these are not the only groups with context-free word problem. More specifically, we will show that the groups with context-free word problem are exactly the virtually free groups. This property turns out to be crucial since it will allow us to create a PDA for the word problem of G.

First, we prove a lemma which will make our life easier, as it allows us to work with any given presentation of a group.

Lemma 4.2. *Let G be a finitely generated group with context-free word problem. If H is any finitely generated subgroup of G, then the word problem of H is also a context-free language.*

Proof. Let $\langle S \mid R \rangle$ be a finitely generated presentation of G with context-free word problem. Let $\langle Y = \{y_1, ..., y_n\} \mid T \rangle$ be a finitely generated presentation of a subgroup H of G. Then, there exists an injective homomorphism $\varphi : H \to G$ which, in terms of the given presentations, is completely defined by the information $\varphi(y_i) = u_i$, $i = 1, ..., n$, where each u_i is a word on $S^{\pm 1}$. Thus, since φ is an injective homomorphism, a word w in $Y^{\pm 1}$ represents the identity in $\langle Y \mid T \rangle$ if and only if $\varphi(w)$ represents the identity in $\langle S \mid R \rangle$.

Let M be the PDA recognizing the word problem of G for the presentation $\langle S \mid R \rangle$. We can construct a PDA M' for the word problem of $\langle Y \mid T \rangle$ as follows. The machine M' has the same states as M (including start state and accepting states), same stack alphabet as M, and same starting stack symbol. On reading input symbol y_i^ϵ (where $\epsilon = \pm 1$), M' simulates the behaviour of M on input $u_i^\epsilon = \varphi(y_i^\epsilon)$. That is, if M is in state q with word w on its stack, and after reading u_i^ϵ it transitions to state q' with word w' on the stack, then, by reading y_i^ϵ at state q with w on the stack, M' will also transition to state q' with w' on its stack. Note that this simulation will require ϵ-moves if multiple letters need to be deleted from the stack during one transition. We thus have that M' accepts a word w if and only if $\varphi(w)$ is accepted by M. □

Corollary 4.2.1. *Let G be a finitely generated group. If the word problem is a context-free language in one finitely generated presentation of G, then it is a context-free language in every finitely generated presentation of G.*

Proof. Let $\langle S \mid R \rangle$ be a finitely generated presentation of G with context-free word problem, and let $\langle S' \mid R' \rangle$ be another finitely generated presentation of G. Since G is a subgroup of itself, we can use the same construction as above to build a PDA for the presentation $\langle S' \mid R' \rangle$ of G. □

The corollary above allows us to refer to a group G as context-free if G is finitely generated and there exists a presentation for which the word problem of G is a context-free language.

The first and simplest example of a context-free group we can provide is that of a free group $F_n = \langle x_1, ..., x_n \mid \emptyset \rangle$ on n generators. Since F_n is free, a word w on $X = \{x_1, ..., x_n\}$ will represent the identity if and only if we can successively delete subwords of w of the form $x_i x_i^{-1}$ or $x_i^{-1} x_i$ until w becomes empty. We can thus create a PDA M with only one state (which is both a start and accept state), and which uses its stack to perform these deletions and recognize the word problem of F_n. More precisely, when reading a letter x_i^ε (where $\varepsilon \in \{\pm 1\}$), M either adds it to the stack if its inverse is not on top, or deletes its inverse which is on top of the stack. For example, when processing the word $x_1 x_2 x_2^{-1} x_1^{-1} x_1$, the contents of the stack are: $x_1, x_1 x_2, x_1, \epsilon, x_1$. Hence, at any point, the word read so far is equal to the identity in F_n if and only if the stack is empty. Thus, M accepts a word w if and only if the stack is empty when M is done processing w.

This example is important as it serves as a building block in the proof of the main theorem to come. More precisely, a PDA for the word problem of a virtually free group G will use its stack to keep track of the "free part" of a word (as in this example), and will use its states to keep track of the "rest" of the word. We can now state two lemmas, without proof, needed to prove the main result of this section.

Lemma 4.3. *Let G be a group and H a subgroup of G of finite index. Then, there exists a normal subgroup $N \subseteq H$ in G, which is also of finite index in G.*

Lemma 4.4 (Schreier's Lemma)**.** *Let G be a finitely generated group, and H a subgroup of G of finite index. Then, H is also finitely generated.*

We are now ready to discuss and prove a characterization of context-free groups. This theorem was first proven by Muller and Schupp in 1983 [7].

Theorem 4.5 (Muller-Schupp Theorem)**.** *A finitely generated group G has context-free word problem if and only G is virtually free.*

Proof. We will only prove that a finitely generated virtually free group G has context-free word problem, as the other direction is too long for the interests of this paper. Let H be a free subgroup which has finite index in G. Then, by the two previous lemmas, there exists a finitely generated normal subgroup $N \trianglelefteq H$ of finite index in H. In particular, it follows by theorem 2.4 that N is free, since it is a subgroup of a free group. Hence, N is a finitely generated free subgroup which is normal and has finite index in G.

We shall use the subgroup N to construct a particular presentation of G. Let N have free generators $y_1, ..., y_n$, where each y_i is an element of N. The quotient group $B = G/N$ is finite since N has finite index in G. Let $|B| = t$, and consider the natural homomorphism $\eta : G \to B$. We can construct the multiplication table presentation of B, given by $\langle\, b_1, ..., b_t \,|\, ..., b_i b_j = b_k, ... \,\rangle$, and choose elements $d_i \in G$ such that $\eta(d_i) = b_i$ for $i = 1, ..., t$. That is, we choose a representative $d_i \in G$ for each coset $b_i \in G/N$.

Since N is normal in G, we have that for all generators y_j of N and representatives d_i, the relation $d_i y_j d_i^{-1} = u_{i,j}$ holds for some $u_{i,j} \in N$. In addition, by considering multiplication of right cosets we get that for each pair of representatives d_i, d_j, we have $d_i d_j^{\epsilon} = z_{i,\epsilon,j} d_k$ for some $z_{i,\epsilon,j} \in N$ and representative d_k (and $\epsilon \in \{\pm 1\}$). Using $y_1, ..., y_n$ and $d_1, ..., d_t$ as letters, we claim that

$$\langle\, y_1, ..., y_n, d_1, ..., d_t \,|\, d_i y_j d_i^{-1} = u_{i,j}, d_i d_j^{\epsilon} = z_{i,\epsilon,j} d_k \,\rangle$$

is a presentation of G. Note that every word on these generators can be transformed using these relations to an unique word of the form $w d_i$ where w is a freely reduced word in the y's. For example, the word $y_1 d_i y_2 d_j^{-1}$ gets transformed as follows: $y_1 d_i y_2 d_j^{-1} \to y_1 d_i y_2 d_i^{-1} d_i d_j^{-1} \to y_1 u_{i,2} d_i d_j^{-1} \to y_1 u_{i,2} z_{i,-1,j} d_k \to w d_k$ where $w = \overline{y_1 u_{i,2} z_{i,-1,j}}$ is a freely reduced word on the y's. Indeed, since the right cosets of N in G partition the group G, every element $g \in G$ can be expressed by $g = w d_i$ where $w \in N$ and d_i is the representative of the coset containing g. Hence, a word $w d_i$ represents the identity in G if and only if w is empty and d_i is the symbol d_1 with $\eta(d_1)$ the identity of B.

We can now construct a PDA M for the word problem of this presentation of G. The machine M works by keeping track of the "free part" of a word in its stack, and keeping track of the image in B by its states. We let M have states $q_1, ..., q_t$ corresponding to the elements $b_1, ..., b_t \in B$, and potentially other "working" states. In particular, M starts in state q_1 (corresponding to identity of B) with empty stack. If M is in state q_i and reads input letter y_j^{ϵ}, then, since $d_i y_j^{\epsilon} = u_{i,j}^{\epsilon} d_i$ in G, M uses its working states to make a sequence of ϵ-moves which process the word $u_{i,j}^{\epsilon}$ onto the stack (as in the example of free groups) and then returns to state q_i. In particular, the working states are necessary if more than one symbol needs to be deleted from the stack (in the case of multiple reductions). If M is in state q_i and reads input letter d_j^{ϵ}, then, since $d_i d_j^{\epsilon} = z_{i,\epsilon,j} d_k$ in G, M uses its working states to process the word $z_{i,\epsilon,j}$ onto the stack and then changes state to q_k.

Hence, after reading an arbitrary word v, which is equal in G to $w d_i$ with w a word on the the y's, M has w on its stack and is in state q_i. Thus, v represents the identity of G if and only if M is in state q_1 with empty stack after reading v. □

We can now provide a simple example of a PDA which accepts a context-free word problem. We consider the group $G = F_2 \times \mathbb{Z}/3\mathbb{Z}$. Note that G is virtually free since the subgroup $N = \{(w, 0) : w \in F_2\}$ of G is free (N is isomorphic to F_2) and of finite index $[G : N] = |\mathbb{Z}_3| = 3$. Also note that N is normal in G since $(u, h)(w, 0)(u, h)^{-1} = (uwu^{-1}, h + (-h)) = (w', 0) \in N$ for all $(u, h) \in G$ and $(w, 0) \in N$. Since N is isomorphic to F_2, it has presentation $\langle\, a, b \,|\, \emptyset \,\rangle$. Now, denote $(\epsilon, 0) \in G$ by x, $(\epsilon, 1) \in G$ by y and $(\epsilon, 2) \in G$ by z. Then, $G/N = \{xN, yN, zN\}$. This gives the following presentation of G:

$$< a, b, x, y, z | xy = y, xz = z, yz = x, xx = x, yy = z, zz = y, xax^{-1} = a,$$
$$xbx^{-1} = b, yay^{-1} = a, yby^{-1} = b, zaz^{-1} = a, zbz^{-1} = b >$$

Note that in this example, relations of the form $d_i d_j^{\epsilon} = z_{i,\epsilon,j} d_k$ have $z_{i,\epsilon,j}$ empty. This is due to our choice of representatives x, y, z and allows us to create a simpler PDA. This PDA for WP(G) has input and stack alphabets $\{a, b, x, y, z\}^{\pm 1}$, and has three states. Each state corresponds to one of x, y, z and the state representing x is a start and accept state. If the input letter is a, b, a^{-1}, b^{-1}, we stay in the same state and process the letter into the stack by performing reductions if necessary. If the input letter is x, y, z or their inverses, we do not touch the stack and change states according to the relations above. The word is then accepted if and only if the machine ends in the accept state with empty stack. For example, when processing the word $axzby b^{-1} a^{-1}$, the contents of the stack are: $\epsilon, a, ab, a, \epsilon$; and the states traversed are: x, z, x. Hence, this word is accepted by the machine.

The family of groups with one counter word problem turns out to be a special case of context-free groups. In fact, since the stack is restricted to only one symbol, this only allows the machine to keep track of the "free part" of words consisting of only one generator. Hence, groups G with one counter word problem are either finite groups, or those containing a free subgroup of rank 1. Note that every finite group G is virtually cyclic since the trivial subgroup is cyclic and has finite index in G. Also, since F_1 is isomorphic to the infinite cyclic group $\mathbb{Z}$, we can further say that groups G containing a free subgroup of rank 1 are virtually cyclic. We can thus state the following theorem, which is studied in detail in [4].

Theorem 4.6. *A finitely generated group G has one counter word problem if and only G is virtually cyclic.*

Proof. Again, we only prove one direction as the other is too long for the scope of this paper. If the group G is finite, then its word problem is regular by theorem 4.1, and it is thus also one counter. Hence, assume the group G is infinite, finitely generated, and virtually $\mathbb{Z}$ (which is the same as infinite virtually cyclic). Using lemmas 4.3 and 4.4, we can extract a normal subgroup Z of finite index in G which is isomorphic to $\mathbb{Z}$ (since $\mathbb{Z}$ is free of rank 1, Z must also be free of rank 1). Then, let x be the free generator of the subgroup Z. The construction of the PDA M for the word problem of G is very similar to the one in the proof of theorem 4.5, using alphabet $\Sigma = \{d_1, ..., d_n, x\}^{\pm 1}$ where $\{d_1, ..., d_n\}$ is a complete set of coset representatives. However, since M is restricted to only one stack symbol x, this can lead to problems when trying process the input symbol x^{-1} onto an empty stack.

This can be solved by considering two copies of the states in M, one "positive" and one "negative" (where the symbol x is interpreted as x^{-1} on the stack). The transition function is then modified as follows. If the stack is empty or has symbols corresponding to x, then M is in a positive state and behaves normally. If the stack is empty and M wants to process x^{-1} onto the stack (that is it wants to perform a transition $d_i d_j = x^{-1} d_k$), it changes to the corresponding negative state and pushes x onto the stack. Similarly, if the stack has one symbol corresponding to x^{-1} (that is x is on the stack but M is in a negative state), and M wants to process x onto the stack, then M pops the x from the stack and changes to the corresponding positive state. Finally, if M is in a negative state with multiple symbols on the stack, it pops the stack when processing x and pushes when processing x^{-1}.

The machine M will thus accept WP(G) using only one stack symbol x. Hence, the PDA constructed is a one counter machine, and the word problem of G is a one counter language. □

It is worthy to mention that the free abelian group $\mathbb{Z}^2 = \langle x, y \mid xyx^{-1}y^{-1} \rangle$ is not context-free (and thus not one counter). Indeed, a word of the form $x^m y^n x^{-i} y^{-j}$ represents the identity if and only if $m = i$ and $n = j$. However, it is impossible for a PDA to match these two quantities at the same time since it only has one stack. This can be proven formally using the pumping lemma for CFL's, and makes $\mathbb{Z}^2$ the "simplest" example of a group which is not context-free.

This leads to the discussion of groups with recursively enumerable word problem. This family of groups includes $\mathbb{Z}^2$ and many other more "complicated" groups. In particular, given

a group $G = \langle S \,|\, R \rangle$, we will see that as long as we can enumerate every word on S which is trivial in G, this will make $\mathrm{WP}(G)$ recursively enumerable.

We first state an important definition. We say a finitely generated group is *recursively presentable* if it has a presentation of the type $\langle s_1, ..., s_M \,|\, R \rangle$ where R is a recursively enumerable set of words on the generators. A characterization of recursively presentable groups was proven by Higman in 1961 [5], and we state it here for completion.

Theorem 4.7 (Higman's Embedding Theorem)**.** *A finitely generated group G is recursively presentable if and only if it embeds into some finitely presented group.*

We can now state and prove our next result, which classifies groups with recursively enumerable word problem.

Theorem 4.8. *The word problem $\mathrm{WP}(G)$ of a finitely generated group G is recursively enumerable if and only if G embeds into some finitely presented group.*

Proof. Let G be generated by a finite set $X \subseteq G$, and let $\mathrm{WP}(G)$ be recursively enumerable. To each element x_i of X assign a symbol X_i and let X' be the set of such symbols. Note that $\mathrm{WP}(G)$ is a subset of $(X' \cup (X')^{-1})^*$ by the correspondence $x_i \mapsto X_i$. We will show that $G \cong G' = \langle X' \,|\, \mathrm{WP}(G) \rangle$ and is thus recursively presentable, which implies that G embeds into some finitely presented group by the previous theorem. Let $g = x_1...x_n \in G$, and define $\varphi : G \to G'$ by $\varphi(g) = X_1...X_n$. This map is well-defined since if $g = x_{i_1}...x_{i_n} = x_{j_1}...x_{j_m}$, then $\varphi(x_{i_1}...x_{i_n}x_{j_m}^{-1}...x_{j_1}^{-1}) = \varphi(1) = 1$. Therefore, $X_{i_1}...X_{i_n}X_{j_m}^{-1}...X_{j_1}^{-1} \in \mathrm{WP}(G)$ and thus $x_{i_1}...x_{i_n}$ and $x_{j_1}...x_{j_m}$ are mapped to the same element. Now, let $g = x_1...x_n \in \ker(\varphi)$. Then, $X_1...X_n \in \mathrm{WP}(G) \implies x_1...x_n =_G 1$ and thus $\ker(\varphi) = \{1\}$. Hence, φ is injective. Finally, φ is surjective since any word $w = X_1^{\epsilon}...X_n^{\epsilon}$ is the image of $g = x_1^{\epsilon}...x_n^{\epsilon} \in G$ (where $\epsilon \in \{\pm 1\}$). Thus, φ is an isomorphism and the group G is recursively presentable.

Now, assume G embeds into some finitely presented group. Then, by Higman's theorem, G has some recursive presentation $\langle S \,|\, R \rangle$. We want to show that $\mathrm{WP}(G)$ is recursively enumerable. Note that $G \cong F(X)/\mathrm{WP}(G)$ since $N(R) = \mathrm{WP}(G)$ by definition. Therefore, $w \in \mathrm{WP}(G)$ if and only if $\exists\, n \in \mathbb{N}, r_{i_1}, ..., r_{i_n} \in R$ and $w_{j_1}, ..., w_{j_n} \in F(S)$ such that

$$w = \prod_{k=1}^{n} w_{j_k} r_{i_k} w_{j_k}^{-1}$$

This allows us to enumerate each $w \in \mathrm{WP}(G)$ in terms of $n, r_{i_1}, ..., r_{i_n}$ and $w_{j_1}, ..., w_{j_n}$. The enumerating procedure is rather complicated, but intuitively it consists of listing words w up to some n, only using r_i and w_j for $1 \leq i, j \leq n$, and increase the value of n after some time. This is possible since R and $F(S)$ are recursively enumerable, and can thus be enumerated. Given enumerations $r_1, r_2, ...$ of R and $w_1, w_2, ...$ of $F(S)$ this procedure works as follows.

- In step 1, produce the element $w_1 r_1 w_1^{-1}$.
- In step 2, produce all words with $n \leq 2$, using r_1, r_2 and w_1, w_2:
  - $w_1 r_1 w_1^{-1}$, $w_1 r_2 w_1^{-1}$, $w_2 r_1 w_2^{-1}$, $w_2 r_2 w_2^{-1}$;
  - $w_1 r_1 w_1^{-1} w_1 r_1 w_1^{-1}$, $w_1 r_1 w_1^{-1} w_1 r_2 w_1^{-1}$, $w_1 r_1 w_1^{-1} w_2 r_1 w_2^{-1}$, $w_1 r_1 w_1^{-1} w_2 r_2 w_2^{-1}$;
  - $w_1 r_2 w_1^{-1} w_1 r_1 w_1^{-1}$, $w_1 r_2 w_1^{-1} w_1 r_2 w_1^{-1}$, $w_1 r_2 w_1^{-1} w_2 r_1 w_2^{-1}$, $w_1 r_2 w_1^{-1} w_2 r_2 w_2^{-1}$;
  - $w_2 r_1 w_2^{-1} w_1 r_1 w_1^{-1}$, $w_2 r_1 w_2^{-1} w_1 r_2 w_1^{-1}$, $w_2 r_1 w_2^{-1} w_2 r_1 w_2^{-1}$, $w_2 r_1 w_2^{-1} w_2 r_2 w_2^{-1}$;
  - $w_2 r_2 w_2^{-1} w_1 r_1 w_1^{-1}$, $w_2 r_2 w_2^{-1} w_1 r_2 w_1^{-1}$, $w_2 r_2 w_2^{-1} w_2 r_1 w_2^{-1}$, $w_2 r_2 w_2^{-1} w_2 r_2 w_2^{-1}$
- In step 3, produce all words with $n \leq 3$, using r_1, r_2, r_3 and w_1, w_2, w_3 in a similar way as above.

⋮

- In step t, produce all words with $n \leq t$, using $r_1, ..., r_t$ and $w_1, ..., w_t$ in a similar way as above.

Clearly, any word $w \in \mathrm{WP}(G)$ determined by $(n, r_{i_1}, ..., r_{i_n}, w_{j_1}, ..., w_{j_n})$ will be produced in step number $\max\{n, I, J\}$, where $I = \max\{i_1, ..., i_n\}$ and $J = \max\{j_1, ..., j_n\}$. Therefore, this procedure, called *dovetailing*, shows that $\mathrm{WP}(G)$ is recursively enumerable.

□

5. The conjugacy problem for groups

We now briefly investigate another question posed by Max Dehn in 1911, known as the conjugacy problem.

The Conjugacy Problem: Let G be a finitely generated group given by the presentation $\langle S \,|\, R \rangle$.
Can we decide whether two words in S are conjugate in G?

This question is also presented as a decision problem, but we will study it from a language theoretic perspective. Given a group G described as above, we can consider the set of all pairs of words on S that are conjugate in G, and investigate the properties of this set as a formal language. Note that if we restrict the second word of a pair (u, v) to be the empty word, we get the word problem of the group G. Hence, we should expect the conjugacy problem to be at least as complex as the word problem of a group. Indeed, since we now consider pairs of words, things become more delicate as the machines can read those pairs of words in different ways.

In fact, we can now consider two-variable machines which read pairs of words $(u, v) \in \Sigma^* \times \Sigma^*$. Such a machine may be synchronous or asynchronous, leading to different group theoretic results later on. For brevity, we will only consider the case of synchronous machines. A more extensive study of this topic can be found in [6].

A synchronous machine reads simultaneously, at any given point, a single symbol from each of the two input words, where the shorter of the two words is padded at the end with a *padding symbol* \$. In particular, the padding symbol \$ maps onto the identity element of the group G. We shall call a language of pairs of strings *synchronously regular* if it is accepted by a finite state automaton reading the two input words synchronously. In a similar way, we define *synchronously context-free* and *synchronously one counter* languages.

Synchronous two-variable machines are equivalent to a special kind of one-variable machine. Indeed, if a synchronous two-variable machine reads pairs of words over an alphabet X, it can be regarded as a one-variable machine with input alphabet $(X \cup \{\$\}) \times (X \cup \{\$\})$. This type of machine is often more convenient to work with, and therefore we state our definition of the conjugacy problem of a group based on this convention.

The *conjugacy problem* $\mathrm{CP}(G, \Sigma)$ of a finitely generated group G with presentation $\langle S \,|\, R \rangle$ is defined as the following set:

$$\mathrm{CP}(G, \Sigma) = \{w = (x_1, y_1)...(x_n, y_n) \in \Sigma^* : \exists\, g \in G, g x_1 ... x_n g^{-1} =_G y_1 ... y_n\}$$

where $\Sigma = (S^{\pm 1} \cup \{\$\})^2$. This set depends on the given presentation of G, more specifically on its generating set. Most of the times, however, the choice of generators is clear and we just write $\mathrm{CP}(G)$ for the conjugacy problem of G. Note that this definition is equivalent to considering the set of pairs of words on $S^{\pm 1}$ such that they are conjugate in G.

We can now state and prove a first result, which is of similar nature to theorem 4.1.

Theorem 5.1. *A group has synchronous regular conjugacy problem if and only if it is finite.*

Proof. Suppose that the conjugacy problem of a group G is synchronously regular. Then, there is a DFA $M = (S, s_0, \delta, F)$ which accepts $\mathrm{CP}(G)$. We will modify M in such a way that it accepts the word problem of G. We first add a "dead" state d to M. That is, d is not an accept state, and on reading any letter in state d, M remains in d. Then, we change δ in such a way that for any state $q \in S$: $\delta(q, (a, b)) = d$ if $b \neq \$$. Hence, this new automaton M' accepts the pair (u, v) if and only if $u =_G 1$ and v is empty (that is $v = \$^m$ for some $m \in \mathbb{N}$). We can then use M' to create a new one-variable DFA N which on input w simulates the behaviour of M' on input (w, ϵ). Thus, N accepts a word w if and only if $w =_G 1$. Hence, $L(N) = \mathrm{WP}(G)$, and by theorem 4.1 G must be finite.

Now, suppose G is finite. We can then create a one-variable DFA M, based on the Cayley graph of $G \times G$, which accepts $\mathrm{CP}(G)$. Note that since G is finite, $G \times G$ it is generated by the finite set $X = \{(1, g) : g \in G\} \cup \{(g, 1) : g \in G\}$. We fix the alphabet $\Sigma = X^{\pm 1}$, and let the states of the machine be the set $G \times G$. In particular, the start state is the state corresponding to $(1, 1) \in G \times G$, and the accept states are $F = \{(u, v) : \exists\, g \in G, gug^{-1} = v\}$. On reading some letter $(1, h)$ on state (g_1, g_2), M transitions to state (g_1, g_2h), and on reading some letter $(h, 1)$ on state (g_1, g_2), M transitions to state (g_1h, g_2). Hence, a word $(u, v) = (g_1, h_1)^{\varepsilon_1}...(g_n, h_n)^{\varepsilon_n}$ (where $\varepsilon_i \in \{\pm 1\}$, $u = g_1^{\varepsilon_1}...g_n^{\varepsilon_n}$ and $v = h_1^{\varepsilon_1}...h_n^{\varepsilon_n}$) is accepted if and only if u is conjugate to v. Hence, the machine M can be used to recognize the conjugacy problem for G. □

Things become more complicated when we move to one counter and context-free languages. In particular, synchronous and asynchronous machines will lead to different group theoretic properties. First, we can state the following lemma, proven in [6], which allows us to work with any generating set of a group.

Lemma 5.2. *If the conjugacy problem of a group is synchronously context-free with respect to one generating set, then the same is true with respect to any other generating set.*

We can now state and prove the following theorem about one counter conjugacy problems.

Theorem 5.3. *Every virtually cyclic group has synchronously one-counter conjugacy problem.*

Proof. Since regular languages are also one counter, finite groups have one counter conjugacy problem. Now, assume a group $G = \langle X \rangle$ is virtually infinite cyclic, that is virtually $\mathbb{Z}$. Then, by lemmas 4.3 and 4.4, G has a normal cyclic subgroup Z of finite index in G. Let Z be generated by the element z, and let T be a set of coset representatives for $Z = \langle z \rangle$ in G. Since the right cosets of Z in G partition the group G, every element $g \in G$ is uniquely representable by a word $z^k t$ for $k \in \mathbb{Z}$ and $t \in T$. We shall refer to this expression of a word as its normal form.

Note that since Z is normal in G, conjugation of z by any element of G defines an automorphism of Z. Let $\varphi_g : Z \to Z$ be the automorphism of Z defined by $g \in G$. Then, φ_g maps generators of Z to generators of Z, and thus $\varphi_g(z) = z$ or $\varphi_g(z) = z^{-1}$, since z and z^{-1} are the only generators of $Z \cong \mathbb{Z}$. Thus, for any $g \in G$, we either have $g^{-1}zg = z \implies zg = gz$, in which case g commutes with z, or we have $g^{-1}zg = z^{-1} \implies z^{-1}g = gz$, in which case g inverts z.

The automaton constructed for the conjugacy problem for G will have alphabet $\Sigma = (T \cup \{z\})^{\pm 1}$, and use the normal form of a word to process it. Let $u = z^i r$ and $v = z^k t$ be two words in normal form, where $r, t \in T$ and $i, k \in \mathbb{Z}$. We consider the conjugate of $z^i r$ by $z^k t$, that is the word $(z^k t)^{-1} z^i r z^k t$. This word can take four forms, depending on the following cases.

(a) r, t both commute with z

Then, $(z^k t)^{-1} z^i r z^k t = t^{-1} z^{-k} z^i z^k r t = t^{-1} z^i r t = z^i t^{-1} r t$.

(b) r commutes with z but t inverts z

Then, $(z^k t)^{-1} z^i r z^k t = t^{-1} z^{-k} z^i r z^k t = t^{-1} z^i r t = z^{-i} t^{-1} r t$.

(c) t commutes with z but r inverts z

Then, $(z^k t)^{-1} z^i r z^k t = t^{-1} z^{-k} z^i r z^k t = t^{-1} z^{-k} z^i z^{-k} r t = z^{i-2k} t^{-1} r t$.

(d) r, t both invert z

Then, $(z^k t)^{-1} z^i r z^k t = t^{-1} z^{-k} z^i r z^k t = t^{-1} z^{i-2k} r t = z^{2k-i} t^{-1} r t$.

Hence, if we use the notation r^t for $t^{-1} r t$, conjugation of any two words $z^i r$ and $z^k t$ gets one of the four following forms: $z^i r^t, z^{-i} r^t, z^{i-2k} r^t, z^{2k-i} r^t$. We are now ready to describe a two-variable one counter automaton M that accepts the conjugacy problem of G.

Suppose words u, v over Σ are input, and that u and v have normal forms $z^i r$ and $z^j s$ with $r, s \in T$. Before starting to read, M will guess one of the four cases above, as well as a specific choice of t, and attempt to verify that v is a conjugate of u of that type. It will then use its states to find the representatives r and s, and will use its stack in a different way for each of the four cases above.

More precisely, first M guesses one of the four cases. This can be done by using 4 "working" states c_1, c_2, c_3, c_4, one for each case, all connected to the start state by ϵ-moves. Then, each of those states is connected by ϵ-moves to n different copies of $T \times T$, where $n = |T|$ and each copy corresponds to a different $t_q \in T$. By copy of $T \times T$, we refer to a set of states where each state corresponds to an element of $T \times T$. In particular, each c_i is connected to the state corresponding to the identity $(1, 1) \in T \times T$. Hence, for each case p and choice of coset representative t_q, there is a set of states $S_{p,q}$ reachable from the start by ϵ-moves, corresponding to a copy of $T \times T$. For technical reasons that will become clear later, each subset of states $S_{p,q}$ has two versions of $T \times T$, one "positive" and the other "negative", as in the proof of theorem 4.6.

Therefore, when M starts reading (u, v), it is at a state corresponding to the identity, in some $S_{p,q}$ for case p and element t_q. When reading the pair of letters (a, b) in state (g, h), M transitions to state (g', h'), where we have $ga = z^{l_1} g'$ and $hb = z^{l_2} h'$. Hence, after reading u and v, M is in state $(r, s) \in S_{p,q}$, where $u = z^i r$ and $v = z^j s$. The contents of the stack will, however, depend on which of the four cases M has guessed.

The machine can then check, for the element t_q it has guessed, that r^{t_q} and s are in the same coset of Z, and whether each of r, t_q commute with or invert z, as per the case selected. This can be done by considering the following accept states: $(g, h) \in S_{p,q}$ such that $t_q^{-1} g t_q$ and h are in the same coset, and such that g, t_q commute with z or invert z as specified by case p. Note that this information is available when constructing M. If either of these checks fail, the pair (u, v) is rejected by the machine. Otherwise, it is the case that $r^{t_q} = z^l s$ for some known $l \in \mathbb{Z}$, and we can check it against the contents of the stack for each of the four cases.

(a) When transitioning states as described above, M pushes its stack symbol onto the stack l_2 times, and pops it l_1 times. If it needs to pop when the stack is empty, it uses its negative states, as in the proof for theorem 4.6, to process the correct amount into the stack. Hence, when M is done reading $(z^i r, z^j s)$, it will have the equivalent of $j - i$ on its stack. Then, these words are conjugate if and only if $i + l = j$, and thus M needs to check that $j - i = l$. This can be done by popping from the stack l times (through a sequence of ϵ-moves) and checking if it's empty (if it is empty before that then it fails).

(b) When transitioning states as described above, M pushes its stack symbol onto the stack $l_1 + l_2$ times (using negative states if necessary). Therefore, after reading $(z^i r, z^j s)$, M will have the equivalent of $i + j$ on its stack, so it checks if $i + j = l$ by popping from the stack l times and checking if it's empty as in the previous case.

(c) In this case, the two words are conjugate if and only there exists some $k \in \mathbb{Z}$ such that $i - 2k + l = j \iff i - j = 2k - l \iff i - j \equiv l \mod 2$. Thus, M needs to check if $i - j$ and l have the same parity. That is, it checks if $i - j$ is even when l is even, and if $i - j$ is odd when l is odd. When transitioning states, this case works in a similar way

than (a), but uses it stack to indicate if $l_1 - l_2$ is even or odd. In particular, after reading $(z^i r, z^j s)$, M will contain one symbol on its stack if $i - j$ is odd, and empty stack if it is even. Hence, the pair of words is accepted if and only if the contents of the stack match the parity of l.

(d) Finally, this case is similar to (d), but compares the parity of $i + j$ and l.

Therefore, the machine M accepts the conjugacy problem for G. Hence, if G is virtually cyclic, then $\mathrm{CP}(G)$ is one counter.

□

The converse of this theorem is also true, and a proof can be found in [6].

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Dr. Jeremy Macdonald, for his invaluable guidance and support throughout the development of this research project. I would also like to thank Prof. Marcin Sabok for acting as a second supervisor for the project.

References

[1] A. Anisimov. Groups languages. *Kybernetika*, 4:18–24, 1971.

[2] W. W. Boone. The word problem. *Annals of Mathematics*, 70(2):207–265, 1959.

[3] M. Dehn. Über unendliche diskontinuierliche gruppen. *Mathematische Annalen*, 71:116–144, 1912.

[4] T. Herbst. On a subclass of context-free groups. *RAIRO Theor. Informatics Appl.*, 25:255–272, 1991.

[5] G. Higman. Subgroups of finitely presented groups. *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, 262:455 – 475, 1961.

[6] D. Holt, S. Rees, and C. Röver. Groups with context-free conjugacy problems. *IJAC*, 21:193–216, 02 2011.

[7] D. E. Muller and P. E. Schupp. Groups, the theory of ends, and context-free languages. *Journal of Computer and System Sciences*, 26(3):295–310, 1983.

[8] P. S. Novikov. On the algorithmic unsolvability of the word problem in group theory. *Trudy Matematicheskogo Instituta imeni VA Steklova*, 44:3–143, 1955.

[9] M. Sipser. *Introduction to the Theory of Computation*. Course Technology, Boston, MA, third edition, 2013.