Asia Pac. J. Math. 2024 **11**:6

COMPLEXITY STUDY OF LANGUAGE OPERATIONS USING FINITE GROUP AUTOMATA

G. SATHIYASORUBINI, R. VENKATESAN*

Department of Mathematics, College of Engineering and Technology, SRM Institute of Science and Technology, Kattankulathur, Chengalpattu - 603203, Tamilnadu, India

*Corresponding author: venkater1@srmist.edu.in

Received Oct. 15, 2023

Abstract. We have explored the intricacies of deterministic finite automata for regular languages derived from applying various operations to languages accepted by finite group automata. These operations include Union, quotient, complement, difference, intersection, Kleene star, Kleene plus, and reversal. Our analysis has involved defining three types of finite group automata structures and examining their complexities. Specifically, we have looked into four complexities of finite group automata: state complexity, accepting state complexity, syntactic complexity, and quotient complexity. Our findings show that the accepting state complexity is the same for all operations defined in the finite group automata structures. Moreover, we have identified the precise values of the syntactic and quotient complexity of the languages accepted by finite group automata when the group is finite cyclic and have determined the range for other groups.

2020 Mathematics Subject Classification. 68Q70; 68Q45; 18B20; 20M32; 20M35;

Key words and phrases. finite group automata; state complexity; accepting state complexity; syntactic complexity; quotient complexity.

1. Introduction

State complexity is a measure of descriptive complexity used for regular languages. It is the smallest number of states in any deterministic finite automaton (DFA) accepting the language, and it is also defined as the smallest number of states in any nondeterministic finite automaton (NFA) for the language [12]. State complexity is not about computational complexity but rather the concise description of an object, in this case, a regular language. The practical implications of state complexity are not related to time and space requirements by the machine but rather to the size of the automaton required to recognize the language [1]. Sheng Yu has researched the state complexity of regular languages and their operations. Their research compares the state complexity results of regular languages with those of finite languages. They also investigate the similarity and equivalence relations over the alphabet

DOI: 10.28924/APJM/11-6

set. The research on the state complexity of combined operations was initiated based on the results of individual operations [2].

Jurgen Dassow thoroughly investigated the number of accepting states required to accept a regular language using deterministic finite automata. He introduced the notation "$asc(L)$" to represent the smallest number of accepting states required to accept a regular language L using deterministic finite automata and "$nasc(L)$" to represent the corresponding nondeterministic automata measure. Dassow proved that for any non-negative integer n, a regular language L exists, such as $asc(L) = n$. He also proved that for any regular language R, $nasc(R) \leq 2$. It means the nondeterministic automata measure for any regular language is at most 2 [3]. Michal Hospodr and Markus Holzer studied the minimum number of accepting states required for deterministic finite automata to process languages resulting from unary and binary operations. They focused on determining the accepting state complexity as a parameter and built upon the previous work of J. Dassow. The study addressed several open problems and provided precise ranges of accepting state complexities for various operations, including intersection, symmetric difference, right and left quotients, reversal, and permutation on finite languages [4]. Additionally, they examined the symmetric difference in unary finite languages and identified a noncontiguous range of accepting state complexities. The study contributes to the research on the accepting state complexity of regularity-preserving language operations [5].

Christian Rauch and Markus Holzer researched the accepting state complexity (asc) of deterministic finite automata for regular languages. They focused on specific operations in languages that permutation automata accept. The operations they investigated were complement, union, quotient, difference, intersection, Kleene plus, Kleene star, and reversal. The authors found that for most operations (excluding reversal and quotient), there is no difference in ASC for permutation automata compared to deterministic finite automata. However, they discovered that achieving certain accepting state complexities for reversal and quotient operations was impossible. The authors solved the ASC for the intersection operation of unary languages accepted by permutation finite automata (PFA) and DFA. The study contributes to the research on accepting state complexity of regularity preserving language operations [6]. Regular language's syntactic complexity is determined by its syntactic semigroup's cardinality, which is isomorphic to the Myhill-Nerode relation. Studying the syntactic complexity of star-free languages is another area of research in this field. Researchers have studied the state complexity of operations on regular languages and introduced a new approach based on derivatives of regular expressions [7].

The researchers explored the quotient complexity of a language $h(M, N)$, where M and N are regular languages, and h is a regular operation such as union or concatenation. As state complexity is a language property, it is appropriate to define it in terms of formal language and refer to it as "quotient complexity," which is the number of distinct quotients of the language [10]. The researchers

used derivatives to represent quotients and obtained a formula for the typical quotient of $h(M, N)$ using the quotients of M and N. They only needed to count the possible quotients to determine the upper bound on the number of $h(M, N)$ quotients, making automaton constructions unnecessary. The Nerode equivalence is closely related to quotient complexity, similar to state complexity [8]. In contrast, the syntactic complexity of a language is defined by the Myhill equivalence, which defines the syntactic semigroup of a language and its size [9]. Researchers have noted that languages with the same quotient complexity can exhibit significant variations in syntactic complexity. A finite group automaton introduced by A. V. Kelarev [11].

The paper introduces new finite group automata with initial and final states [13]. It explores the complexities of deterministic finite automata for regular languages resulting from various operations applied to languages accepted by finite group automata. The operations investigated include union, complement, quotient, difference, intersection, Kleene plus, Kleene star, and reversal. The authors define three types of finite group automata structures and examine their complexities: state complexity, accepting state complexity, syntactic complexity, and quotient complexity. The research concludes that there is no noteworthy distinction in the complexity of accepting states across all operations except the structure-1 complement operation. Additionally, the authors determine the syntactic and quotient complexity range for languages accepted by finite group automata, where the group is finite abelian or finite non-abelian. They also identify the precise value when the group is finite cyclic. The paper presents relevant lemmas and theorems to these complexities and provides illustrations to aid understanding.

In this paper, Section 1 gives a comprehensive introduction. Section 2 covers basic definitions. Section 3 explores the characteristics of finite group automata. Section 4 outlines the structure of these automata. Section 5 analyzes the complexity of state and accepting states in new finite group automata, along with relevant theorems. Section 6 discusses theorems related to the syntactic and quotient complexity. Section 7 focuses on generalization, and finally, Section 8 presents the paper's conclusion.

2. Preliminaries

This section aims to comprehensively understand the topic by covering its fundamental aspects. We will analyze the key concepts, background details, and essential terminology to establish a solid groundwork for further exploration of the subject.

Definition 1. *A finite group automaton is an algebraic system* $\mathfrak{A} =$

$(Q,\ G,\ \delta)$, *where*

- Q *is a finite set of states.*
- G *is a finite group of input symbols.*

- $\delta : Q \times G \to Q$ *is a transition function satisfying the equality*

  $\delta\ (q,\ g\ h) = \delta\ (\delta\ (q,\ g),\ h)$ *for all* $q \in Q$, $g, h \in G$.

  Notation $q\ g = q \cdot g$ *is also used for* $\delta\ (q,\ g)$, $q \in Q$, $g \in G$.

Definition 2. *The accepting state complexity of a language* L *accepted by a DFA refers to the number of final states denoted by* $ASC(L)$ *in the automaton* $\mathfrak{A}$.

Definition 3. *The syntactic complexity of a language* L *is the cardinality of its syntactic semigroup* $\sum^+ / \approx_L$, *which is the set of equivalence classes of the relation* $\approx_L$.

Definition 4. *The syntactic monoid of a language* L *over* Σ^* *is obtained by taking the quotient of* Σ^* *by the syntactic congruence of* L. $u \approx_L v$ *if and only if* $\forall\ x, y$ *in* $\sum^*$, xvy *in* $L \Leftrightarrow xuy$ *in* L.

Definition 5. *The left quotient or quotient of a language* L *by a word* w *is the language* $w^{-1}L = \{x\ \in \sum^*\ |\ w\ x\ \in\ L\}$. *The number of distinct quotients of* L *is the quotient complexity of* L *denoted by* $\kappa(L)$.

3. Characteristics of finite group automata

This section will define finite group automata and explain the group operation. To ensure clarity, we will provide two examples - one that satisfies the condition and one that does not. Once we have established these examples, we will introduce some lemmas to aid in the construction of the structure of finite group automata.

3.1. Concatenation in group elements.

In theoretical computer science, "concatenation" refers to combining two or more words. This is achieved by taking an alphabet set of symbols, such as letters, and creating finite sequences of those symbols called "strings". For example, if we have two words, $u = a_1\ a_2 \ldots a_n$ *and* $v = b_1\ b_2 \ldots b_n$, *then their concatenation, denoted as* $u\ v$, *is equal to* $a_1\ a_2 \ldots a_n\ b_1\ b_2 \ldots b_n$. *Finite group automata use a similar concatenation process but with a different type of alphabet. Instead of using letters, the symbols used are group elements, and the alphabet itself is a finite group, which has a binary operation that makes it a group under certain conditions. Here, the concatenation is operating the elements under the respective group operation. In this case, the strings are simply collections of group elements. For example, let us say we have a group* G *of order* n, *where* $G = \{g_1,\ g_2,\ \ldots,\ g^n\}$. *If we take two group elements,* g_1 *and* g_2, *their concatenation, denoted as* $g_1\ g_2$, *is also a string in* G *since* G *is closed under its group operation. This means that the concatenation of two strings* u *and* v *in finite group automata is simply the group operation between those strings, resulting in another group element. In summary, the concatenation of strings in finite group automata is equivalent to the usual group operation between two strings, which results in a new group element. This means that the alphabet used in finite group automata, denoted as* $\sum$, *is equal to the group* G, *and all strings in the automaton are simply elements of that group. So* $\sum = \sum^2 = \ldots = \sum^* = G$.

Definition 6. *In a finite group automaton, the transition function δ uses the respective group operation $\circ$ for the input pair $(q, g \circ h)$. The automaton is represented by $\mathfrak{GA} = (Q,\ G,\ \delta)$*

Example 1. *Let us consider $\mathfrak{GA} = (Q,\ G,\ \delta)$ where $Q = \{a,\ b\}$, $G = \{e\}$-Identity element and $\delta : Q \times G \to Q$ is a transition function defined a $\delta\ (a,\ e) = b$ and $\delta\ (b,\ e) = b$. We need to verify whether the following equality holds to show that it is a group automaton. $\delta\ (q,\ g\ h) = \delta\ (\delta\ (q,\ g),\ h)\ \forall\ q \in Q$, $g, h \in G$. Here, G denotes a group with a single element, the identity element $\{e\}$, and Q consists of two states. According to the definition of δ, the following equalities must be satisfied: $\delta\ (a,\ e \circ e) = \delta\ (\delta\ (a,\ e),\ e)$ and $\delta\ (b,\ e \circ e) = \delta\ (\delta\ (b,\ e),\ e)$.*
(i) $\delta\ (a,\ e \circ e) = \delta\ (a,\ e) = b$. $\delta\ (\delta\ (a,\ e),\ e) = \delta\ (b,\ e) = b$. $\Rightarrow \delta\ (a,\ e \circ e) = \delta\ (\delta\ (a,\ e),\ e)$.
(ii) $\delta\ (b,\ e \circ e) = \delta\ (b,\ e) = b$. $\delta\ (\delta\ (b,\ e),\ e) = \delta\ (b,\ e) = b$. $\Rightarrow \delta\ (b,\ e \circ e) = \delta\ (\delta\ (b,\ e),\ e)$.
So $\mathfrak{GA} = (Q,\ G,\ \delta)$ is a finite group automaton. The corresponding DFA diagram is follows.

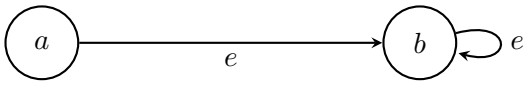


Figure-1

Example 2. *Consider $\mathfrak{GA} = (Q,\ G,\ \delta)$ where $Q = \{a,\ b\}$, $G = \{e\}$-Identity element and $\delta : Q \times G \to Q$ is a transition function defined as $\delta\ (a,\ e) = b$, $\delta\ (b\ , e) = a$.*

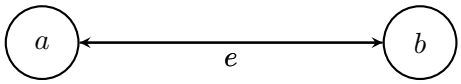


Figure-2

It is not a finite group automaton. Since it does not satisfy the equality $\delta\ (a,\ e \circ e) = \delta\ (\delta\ (a,\ e),\ e)$ i.e $\delta\ (a,\ e \circ e) = \delta\ (a,\ e) = b$ where $\delta\ (\delta\ (a,\ e),\ e) = \delta\ (b,\ e) = a$.

Lemma 1. *In finite group automata, each state must process every input symbol.*

Proof. Let $\mathfrak{GA} = (Q,\ G,\ \delta)$ be finite group automata with a transition function that satisfies the equality $\delta\ (q,\ g\ h) = \delta\ (\delta\ (q,\ g),\ h)$ for all $q \in Q$, $g, h \in G$. If there exists a state that does not read any input, and $q, q_1 \in Q$ and $g, f, h \in G$ such that $\delta\ (q,\ g) = q_1$, $\delta\ (q,\ f) = q_1$, $\delta\ (q_1,\ h) = \emptyset$, and $g \circ h = f$, then we have a contradiction. This is because $\delta\ (q,\ g \circ h) = \delta\ (q,\ f) = q_1$, but $\delta\ (\delta\ (q,\ g),\ h) = \delta\ (q_1,\ h) = \emptyset$. Thus, $\mathfrak{GA} = (Q,\ G,\ \delta)$ cannot be finite group automata. □

Lemma 2. *Suppose Q is a set with only one element, specifically $Q = \{a\}$. Let G be any group of size n, and let $\delta : Q \times G \to Q$ be a transition function. Then we can define $\delta\ (a,\ g) = a$ for all $a \in Q$ and $g \in G$, making $\mathfrak{GA} = (Q,\ G,\ \delta)$ a finite group automaton.*

Lemma 3. *Assuming that Q is a set of two elements, specifically $Q = \{a,\ b\}$, and G is a group of order n, with a transition function $\delta : Q \times G \to Q$. $\mathfrak{GA} = (Q,\ G,\ \delta)$ is finite group automata always. This is achieved by*

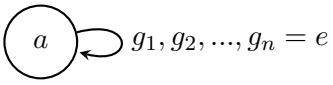
defining $\delta\ (a,\ g) = b$ *and* $\delta\ (b,\ g) = b$ *for* $a, b \in Q$, $g \in G$.

Alternatively, $\delta\ (a,\ g) = a$ *and* $\delta\ (b,\ g) = a$ *for* $a, b \in Q$, $g \in G$ *is also a valid definition.*

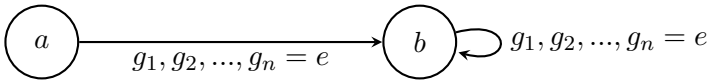
Lemma 4. *Suppose* Q *be a set of* n*- states i.e* $Q = \{a_1,\ a_2,\ \ldots,\ a_n\}$, G *is any group of order* n *and* $\delta : Q \times G \to Q$ *is a transition function. If* $\delta\ (a_i,\ g_i) = a_i + 1$, $\delta\ (a_i,\ e) = a_i$, *and* $\delta\ (a_i,\ g_j) = a_{i+j}$, *where* $i, j \in \mathbb{N}$, *then* $\mathfrak{GA} = (Q,\ G,\ \delta)$ *is considered to be a finite group automata.*

4. Structure of finite group automata

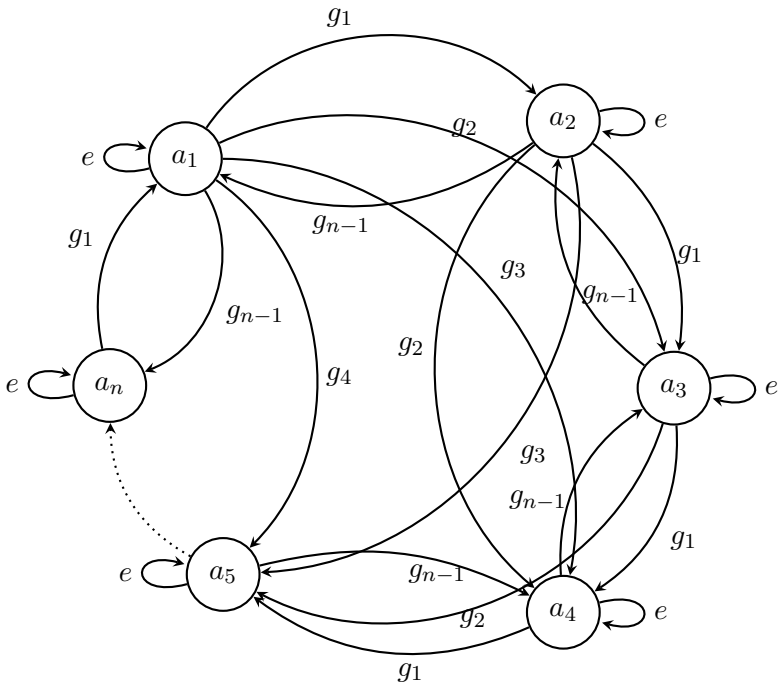
In the previous section, we constructed three distinct structures for finite group automata using various lemmas. These structures are visually represented in a diagram, which serves as a helpful reference for understanding their properties and applications.



Stucture/Type-1



Stucture/Type-2



Stucture/Type-3

5. State complexity and accepting state complexity of new finite group automata

In an earlier discussion, we looked at finite group automata that did not have initial or final state concepts. But now, we will discuss a new type of finite group automata with both an initial and final state. These new automata will allow us to determine the state complexity and the accepting state complexity for the operations mentioned earlier. Each structure has its own set of theorems, and the complexity of the state and accepting state will vary depending on the operation performed.

Definition 7. *In a finite group automaton, the initial state I and final state F. The automaton is represented by $\mathfrak{GA} = (Q,\ G,\ \delta,\ I,\ F)$.*

Definition 8. *Researchers utilize a specific notation to analyze how complex systems behave under certain operations. This notation involves a k-ary operation $\circ$ that preserves regularity in languages, along with natural numbers $n_1, n_2, ..., n_k$ and a parameter c that can be either SC or ASC. The function $g^c_\circ(n_1, n_2, ..., n_k)$ is defined as the set of integers r that satisfy the following conditions: there must be k regular languages $L_1, L_2, ..., L_k$ with $c(L_i) = n_i$ (where $1 \leq i \leq k$), and $c(\circ(L_1, L_2, ..., L_k)) = r$. If only unary languages $L_1, L_2, ..., L_k$ are considered, the function is denoted as $g^{c,u}_\circ$. If the underlying languages are accepted by new finite group automata (NFGAs), $g^c_{\circ,NFGA}$ and $g^{c,u}_{\circ,NFGA}$, respectively.*

Now calculating the state complexity and accepting state complexity for the following operations in new finite group automata. Here the operations are unary.

(1) *Complement*

(2) *Kleene star*

(3) *Kleene plus*

(4) *Union*

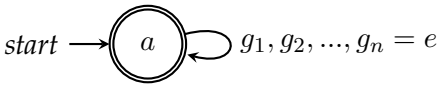
(5) *Difference*

(6) *Quotient*

(7) *Reverse*

Structure − 1

This new finite group automaton has the same structure as the previously defined automaton, including initial and final states.



Stucture/Type-1

The following theorem pertains to the structure-1 of the new finite group automata.

Theorem 1. *The accepting state complexity of structure-1 is 0 for complement and 1 for all other operations.*

Proof. Structure-1 consists of only one state, serving as the initial and final states. As a result, regardless of the language used, the state complexity and accepting state complexity are equal to 1. By analyzing the structure of the new finite group automata, it becomes clear how to compute the state and the accepting state complexity for the mentioned operations.

Complement

- $g^{SC}_{C,\ NFGA}(1) = 1$
- $g^{ASC}_{C,\ NFGA}(1) = 0$

Kleenestar

- $g^{SC}_{*,\ NFGA}(1) = 1$
- $g^{ASC}_{*,\ NFGA}(1) = 1$

Kleeneplus

- $g^{SC}_{+,\ NFGA}(1) = 1$
- $g^{ASC}_{+,\ NFGA}(1) = 1$

Union

- $g^{SC}_{\cup,\ NFGA}(1,\ 1) = 1$
- $g^{ASC}_{\cup,\ NFGA}(1,\ 1) = 1$

Difference

- $g^{SC}_{/,\ NFGA}(1,\ 1) = 1$
- $g^{ASC}_{/,\ NFGA}(1,\ 1) = 1$

Quotient

- $g^{SC}_{-1,\ NFGA}(1,\ 1) = 1$
- $g^{ASC}_{-1,\ NFGA}(1,\ 1) = 1$

Reverse

- $g^{SC}_{R,\ NFGA}(1) = 1$
- $g^{ASC}_{R,\ NFGA}(1) = 1$

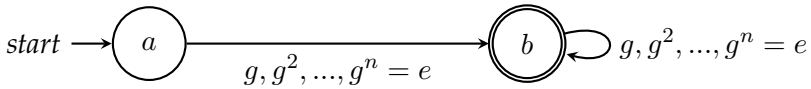
Intersection

- $g^{SC}_{\cap,\ NFGA}(1,\ 1) = 1$
- $g^{ASC}_{\cap,\ NFGA}(1,\ 1) = 1$

□

Structure – 2

A recently developed finite group automaton now includes both initial and final states. Notably, it closely resembles the previous automaton.



Stucture/Type-2

This theorem relates to the structure-2 of the developed finite group automata.

Theorem 2. *The accepting state complexity of structure-2 is 1 for all the operations.*

Proof. In all languages, Structure 2 includes both an initial and final state, meaning that regardless of the language used, the state complexity will always remain at 2 with only one level of accepting state complexity. Moving forward, we will calculate the state complexity and the accepting state complexity of the operations previously mentioned. The structure of the new finite group automata reflects these calculations.

Complement

- $g^{SC}_{C,\ NFGA}(2) = 2$
- $g^{ASC}_{C,\ NFGA}(2) = 1$

Kleenestar

- $g^{SC}_{*,\ NFGA}(2) = 2$
- $g^{ASC}_{*,\ NFGA}(2) = 1$

Kleeneplus

- $g^{SC}_{+,\ NFGA}(2) = 2$
- $g^{ASC}_{+,\ NFGA}(2) = 1$

Union

- $g^{SC}_{\cup,\ NFGA}(2,\ 2) = 2$
- $g^{ASC}_{\cup,\ NFGA}(2,\ 2) = 1$

Difference

- $g^{SC}_{/,\ NFGA}(2,\ 2) = 2$
- $g^{ASC}_{/,\ NFGA}(2,\ 2) = 1$

Quotient

- $g^{SC}_{-1,\ NFGA}(2,\ 2) = 2$
- $g^{ASC}_{-1,\ NFGA}(2,\ 2) = 1$

Reverse

- $g^{SC}_{R,\ NFGA}(2) = 2$
- $g^{ASC}_{R,\ NFGA}(2) = 1$

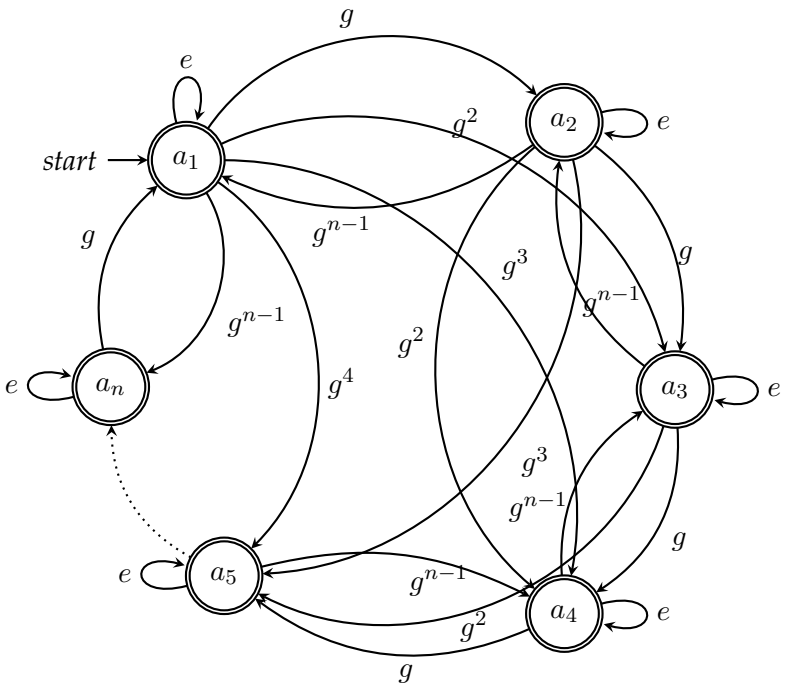
Intersection

- $g^{SC}_{\cap,\ NFGA}(2,\ 2) = 2$
- $g^{ASC}_{\cap,\ NFGA}(2,\ 2) = 1$

□

Structure − 3

In this structure, the number of states is the same as the order of the group. This means that the state complexity is equal to n, which is the number of states. The accepting state can vary from 0 to n, depending on the language used.



Stucture/Type-3

Theorem 3. *The State complexity of the above structure depends on the number of states, denoted by n. In contrast, the accepting state complexity is m, where m is a natural number or zero, and n is any natural number.*

Proof. To prove this, we must consider that any state within the structure may be the final state, and there may be several final states. Therefore, the state complexity is n, while the accepting state complexity cannot exceed n and is denoted by $m \leq n$. To apply this to the operations mentioned earlier, we must determine their respective state complexity and the accepting state complexity.

Complement

- $g^{SC}_{C,NFGA}(n) = n$
- $g^{ASC}_{C,NFGA}(n) = 0 \leq m \leq n$

Kleenestar

- $g^{SC}_{*,NFGA}(n) = n$
- $g^{ASC}_{*,NFGA}(n) = 0 < m \leq n$

Kleeneplus

- $g^{SC}_{+,NFGA}(n) = n$
- $g^{ASC}_{+,NFGA}(n) = 0 < m \leq n$

Union

- $g^{SC}_{\cup,NFGA}(n,\ l) = n$
- $g^{ASC}_{\cup,NFGA}(n,\ l) = 0 < m \leq n$

Difference

- $g^{SC}_{/,NFGA}(n,\ l) = n$
- $g^{ASC}_{/,NFGA}(n,\ l) = 0 \leq m \leq n$

Quotient

- $g^{SC}_{-1,NFGA}(n,\ l) = n$
- $g^{ASC}_{-1,NFGA}(n,\ l) = 0 \leq m \leq n$

Reverse

- $g^{SC}_{R,NFGA}(n) = n$
- $g^{ASC}_{R,NFGA}(n) = 0 \leq m \leq n$

Intersection

- $g^{SC}_{\cap,NFGA}(n,\ l) = n$
- $g^{ASC}_{\cap,NFGA}(n,\ l) = 0 \leq m \leq n$

The accepting state complexity vary from $0 \leq m \leq n$ depending on the languages. There is no notable variation in the accepting state complexity of structure-3 operations.

□

6. Syntactic complexity and Quotient complexity of new finite group automata

In this discussion, we will examine finite group automata's syntactic and quotient complexity. Specifically, we will begin by exploring the syntactic complexity related to the languages recognized by the automata. Our primary focus will be on the cyclic group of order n, represented by the symbol $\sum$ and denoted by the group G.

Theorem 4. *If L is a singleton set, then its syntactic complexity is n, i.e., the group's order.*

Proof. Let $L = \{g^t \mid g^t \in G$ and $t \in \mathbb{N}$ is fixed $\}$. Syntactic complexity refers to the number of equivalence classes of $\sum^*$ under Myhill congruence. The congruence $u \approx_L v$ if and only if $x\ u\ y$ in $L \Leftrightarrow x\ v\ y$ in $L\ \forall\ x$, $y \in \sum^*$. Here, $\sum^* = G$, where G is a cyclic group of order n. So G expressed as $G = \{g,\ g^2,\ \ldots,\ g^n = e\}$, where e is the identity element and g is the generator. Our goal is to prove that for every $u, v \in \sum^* = G$, there exists $x, y \in \sum^* = G$ such that either $x\ u\ y$ in L and $x\ v\ y$ not in L, or $x\ v\ y$ in L and $x\ u\ y$ not in L. Since $x,\ y,\ u$, and v are all in some form of g powers, we can take $x = g^i,\ y = g^j,\ u = g^k,$ and $v = g^l$, where $i,\ j,\ k$, and $l \in \mathbb{N}$. Since G is a group, $x\ u\ y = g^i\ g^k\ g^j = g^{(i+k+j)} = g^r \in G$ and $x\ v\ y = g^i\ g^l\ g^j = g^{(i+l+j)} = g^s \in G$. Here $r \neq s$ since $i + k + j = w + k$ and $i + l + j = w + l$ where $k \neq l$ implies $w + k \neq w + l$ and therefore $g^r \neq g^s$. Thus, if $g^r \in L$ or $g^r \notin L$, then it follows that $g^s \notin L$ or $g^s \in L$. Since L is a singleton, this results in $u \not\approx_L v$, i.e., $[u] \neq [v]$ for every $u, v \in G$.
Suppose g^r and $g^s \notin L$. In this case, we must choose x as a inverse of u i.e $x = g^{-k}$ and y as a element of L i.e $y = g^t$ then $x\ u\ y = g^{-k}\ g^k\ g^t = g^t \in L$ and $x\ v\ y = g^{-k}\ g^l\ g^t = g^{(l+t-k)} \notin L$ since $l \neq k$ implies $l + t - k \neq t$. Thus, $u \not\approx_L v$, i.e., $[u] \neq [v]$ for every $u, v \in G$. Every element in $\sum^* = G$ is in a separate equivalence class. The number of equivalence classes is the syntactic complexity, which forms a monoid since the elements in the group are in separate equivalence classes, which is again G. As every element in the group has a separate equivalence class, the number of equivalence classes equals the group's order, n. Therefore, the syntactic complexity is n. □

Theorem 5. *If $L = G$, then its syntactic complexity is 1.*

Proof. Let L equal G, where G is a cyclic group with an order of n. This means that G is composed of the elements $\{g, g^2, ..., g^n = e\}$, where e is the identity element and g is the generator.
To prove that $u \approx_L v$ for every u and v in $\sum^* = G$, and $\forall\ x, y \in \sum^* = G$, such that xuy in $L = G \Leftrightarrow xvy\ in\ L = G$, we need to prove $[u] = [v]$ for every u and v. Let $x = g^i$, $y = g^j$, $u = g^k$, and $v = g^l$, where $i, j, k, l \in \mathbb{N}$. Since G is a group, we can see that $xuy = g^i g^k g^j = g^{(i+k+j)} = g^r \in G = L$ and $xvy = g^i g^l g^j = g^{(i+l+j)} = g^s \in G = L$. It means that $\forall\ u, v, xuy$ in L and xvy in $L\ \forall\ x, y \in \sum^* = G$. Therefore, $u \approx_L v$ and $[u] = [v]\ \forall\ u, v$ in G. So, there is only one equivalence class. Only the identity element e can form a monoid's equivalence class represented as $[e]$. Hence, the syntactic complexity is 1. □

Theorem 6. *If $L = \{\ g^n | n$-odd$\}$, then its syntactic complexity is 2.*

Proof. Consider the set $L = \{g^n | n \text{ is odd}\}$ and a cyclic group G of order n, where $G = \{g, g^2, ..., g^n = e\}$, with e being the identity element and g being the generator.

To demonstrate the theorem, we need to examine four different cases. Here $u, v, x, y \in \sum^* = G$.

Case − 1

If $u = g^i$ where i is odd and $v = g^j$ where j is odd, take $x = g^k$ and $y = g^l$. Then, $xuy = g^{k+i+l}$ and $xvy = g^{k+j+l}$, where $k, l \in \mathbb{N}$ can be odd or even.

(i) If k, l are odd $\Rightarrow k + i + l$, $k + j + l$ are odd. Thus, xuy and xvy are in L.

(ii) If k is odd, l is even $\Rightarrow k + i + l$, $k + j + l$ are even. Thus, xuy and xvy are not in L.

(iii) If k is even, l is odd $\Rightarrow k + i + l$, $k + j + l$ are even. Thus, xuy and xvy are not in L.

(iv) If k, l are even, $\Rightarrow k + i + l$, $k + j + l$ are odd. Thus, xuy and xvy are in L.

According to the definition of Myhill congruence, $u \approx_L v$ if and only if xuy in $L \Leftrightarrow xvy$ in $L \; \forall \; x, y \in \sum^* = G$, where $\sum^*$ is the Kleene closure of G which is equal to G. It implies that all odd powers are in one equivalence class.

Case − 2

If $u = g^i$ where i is even and $v = g^j$ where j is even, take $x = g^k$ and $y = g^l$. Then, $xuy = g^{k+i+l}$ and $xvy = g^{k+j+l}$, where $k, l \in \mathbb{N}$ can be odd or even.

(i) If k,l are odd $\Rightarrow k + i + l$, $k + j + l$ are even. Thus, xuy and xvy are not in L.

(ii) If k is odd, l is even $\Rightarrow k + i + l$, $k + j + l$ are odd. Thus, xuy and xvy are in L.

(iii) If k is even, l is odd $\Rightarrow k + i + l$, $k + j + l$ are odd . Thus, xuy and xvy are in L.

(iv) If k, l are even $\Rightarrow k + i + l$, $k + j + l$ are even. Thus, xuy and xvy are not in L.

Similarly, we can conclude that all even powers are in one equivalence class.

Case − 3

Suppose $u = g^i$ where i is odd, and $v = g^j$ where j is even. Let $x = g^k$ and $y = g^l$, where $k, l \in \mathbb{N}$ can be either odd or even. Then, we have $xuy = g^{k+i+l}$ and $xvy = g^{k+j+l}$.

(i) If k, l are odd $\Rightarrow k + i + l$ is odd, $k + j + l$ is even. Thus, xuy is in L, while xvy is not in L.

(ii) If k is odd, l is even $\Rightarrow k + i + l$ is even, $k + j + l$ is odd. Thus, xuy is not in L, while xvy is in L.

(iii) If k is even, l is odd $\Rightarrow k + i + l$ is even, $k + j + l$ is odd. Thus, xuy is not in L, while xvy is in L.

(iv) If k, l are even $\Rightarrow k + i + l$ is odd, $k + j + l$ is even. Thus, xuy is in L, while xvy is not in L.

Therefore, we have $u \not\approx_L v$, which means $[u] \neq [v]$. Specifically, the equivalence class of odd powers does not contain even powers, and vice versa.

Case − 4

Following case-3, if $u = g^i$ where i is even, and $v = g^j$ where j is odd, then $u \not\approx_L v$ i.e $[u] \neq [v]$. It indicates that the odd powers do not belong to the even power equivalence class, and the even powers do not belong to the odd power equivalence class. Hence, there are only two equivalence classes: one with odd powers and the other with even powers. Consequently, the syntactic complexity is 2. If a

group G has an odd order n, its equivalence classes are $[e]$ and $[g^w]$, where w is an even number. If the order of G is even and equals n, then its equivalence classes consist of $[e]$ and $[g^w]$, where w is odd. In either situation, the group of two equivalence classes creates a monoid, and the syntactic complexity is 2. □

Theorem 7. *If $L = \{ g^n | n\text{-even}\}$ then its syntactic complexity is 2.*

Proof. We can use the same steps we used for theorem 3 to prove the current theorem. The difference is that this theorem applies only to even numbers, unlike theorem 3, which was for odd numbers. It is essential to mention that we still need to provide the proof here, as we believe that the readers should work through and comprehend the steps themselves. □

Theorem 8. *If a set L has m elements where m is less than n, L and L' are not subgroup that excludes the identity element $\{e\}$ and the whole group G and does not follow the structure outlined in theorem-3 or theorem-4, then the syntactic complexity of L is n.*

Proof. Consider the set L, which consists of elements of the form g^{n-m_i} for some $i \in \mathbb{N}$. All other elements are denoted as L', the set of elements of the form g^{n-m_j} for some $j \in \mathbb{N}$, where $n-m_i \neq n-m_j$. The proof has two separate cases.
(i) If u in L and v not in L or vice versa, it is clear that u and v are not equivalent, meaning $[u] \neq [v]$. To prove this for any $u, v \in \sum^* = G$, we need to show that $\exists\ x, y \in \sum^* = G\ ni$ either $xuy \in L$ and $xvy \notin L$, or $xvy \in L$ and $xuy \notin L$. Since we know that u and v belong to different sets, we can choose $x = y = e$, where e is the identity element. This gives us either $xuy = eue = u$ in L, $xvy = eve = v$ not in L, or $xuy = eue = u$ not in L, $xvy = eve = v$ in L, proving that u, v are not equivalent and belongs to separate equivalence classes.
(ii) If u in L and v in L or u not in L and v not in L.
To prove $u \not\approx_L v$ i.e $[u] \neq [v]$. For this we have to show for every $u, v \in \sum^* = G\ \exists\ x, y \in \sum^* = G \ni xuy$ in L, xvy not in L or xvy in L, xuy not in L. We can choose $x = u^{-1}$ or v^{-1} (or) $y = u^{-1}$ or v^{-1}. Since $u, v \in L$ then the form of $u = g^{n-m_{i_1}}$ and $v = g^{n-m_{i_2}}$. Let us take $x = u^{-1}$ then $xuy = u^{-1}uy = y$ and $xvy = u^{-1}vy = (g^{n-m_{i_1}})^{-1}g^{n-m_{i_2}}y = g^{-(n-m_{i_1})}g^{n-m_{i_2}}y = g^{m_{i_1}-m_{i_2}}y$. If $g^{m_{i_1}-m_{i_2}} \in L$ then choose $y \notin L$ i.e $y \in L'$ such that $g^{m_{i_1}-m_{i_2}}g^{n-m_j} \in L$ for some $j \in \mathbb{N}$ or if $g^{m_{i_1}-m_{i_2}} \notin L$ then choose $y \in L$ i.e $y \notin L'$ such that $g^{m_{i_1}-m_{i_2}}g^{n-m_{i_3}} \notin L$. It is possible since L and L' is not a subgroup. So there is atleast one pair of element such that $g^{n-m_{i_1}}g^{n-m_{i_2}} = g^{n-m_j}$ for some j or $g^{m_{j_1}}g^{m_{j_2}} = g^{n-m_j}$ for some j. So we can choose the elements which satisfy the condition. It results in $u \not\approx v$, i.e. $[u] \neq [v]$, i.e. each element is in a separate equivalence class. Since L has m elements where $m < n$ and L and L' are not subgroups, the syntactic complexity of L is n. Each element is in a separate equivalence class, and u and v are not equivalent. □

Theorem 9. *If L or L' is a subgroup which $\neq G, \{e\}$-identity element, then the syntactic complexity is 2.*

Proof. For the proof, it is necessary to demonstrate four cases.

(1) If u in L and v in L.

(2) If u not in L and v not in L.

(3) If u in L and v not in L.

(4) If u not in L and v in L.

The third and fourth cases are easy to understand. We can differentiate between u and v by selecting $x = y = e$, where e is the identity element. Therefore, we have $xuy = eue = u \in L$ and $xvy = eve = v \notin L$, or $xuy = eue = u \notin L$ and $xvy = eve = v \in L$. It leads to $u \not\approx v$, meaning that $[u] \neq [v]$, and each element belongs to a distinct equivalence class. We require information on finite cyclic groups and their subgroups to prove our argument. Let G be a cyclic group of order n represented by $G = \{g, g^2, \ldots, g^n = e\}$, where e is the identity element and g is the generator.A finite cyclic group of order n is isomorphic to $\mathbb{Z}_n$. The subgroups of $\mathbb{Z}_n$ are $\mathbb{Z}_n$ and $\{e\}$ (the identity element) if n is prime. However, if n is not prime, then the subgroups of $\mathbb{Z}_n$ are generated by the prime divisors of n along with trivial groups. For our purposes, we will not consider L to be $\mathbb{Z}_n$ or $\{e\}$ (the identity element). Instead, we define L as the set of all elements of the form g^{pm}, where p is a fixed prime and m is a positive integer.

a) If x not in L and y not in L then either xy in L or xy not in L.

$x \notin L \Rightarrow x = g^k$ where $k \in \mathbb{N}$ and $k \neq pm$ for any $m \in \mathbb{N}$ and $y \notin L \Rightarrow y = g^l$ where $l \in \mathbb{N}$ and $l \neq pm$ for any $m \in \mathbb{N}$. Now $xy = g^k g^l = g^{k+l} = g^r$ where $r \in \mathbb{N}$. If $r \neq pm$ for any $m \in \mathbb{N}$ then $xy \notin L$. But we have this case also that $k + l = pm$ for some m. Since there exist two composite numbers such that $k + l = pm_i$ for some i, prime product and composite such that $p_1 z + w = pm_i$ for some i, prime product and another prime product such that $p_1 z + p_2 w = pm_i$ for some i, prime product with same prime product $p_1 z + p_1 w$ where $z, w \in \mathbb{N}$. In this case $xy \in L$.

b) If x in L (x not in L) and y not in L (y in L) then xy not in L.

$x \in L \Rightarrow x = g^{pm_1}$ where p-fixed prime and $m_1 \in \mathbb{N}$ and $y \notin L \Rightarrow y = g^l$ where $l \in \mathbb{N}$ and $l \neq pm$ for any $m \in \mathbb{N}$. Now $xy = g^{pm_1} g^l = g^{pm_1 + l} \neq g^{pm}\ \forall\ m$. Since $pm_1 + l \neq pm$ for any m. Suppose $pm_1 + l = pm$ for some m then $l = pm - pm_1 \Rightarrow l = p(m - m_1) \Rightarrow l = pm_2$ which is contradiction since $l \in \mathbb{N}$ and $l \neq pm$ for any $m \in \mathbb{N}$. So $pm_1 + l \neq pm\ \forall\ m$. This results $xy \notin L$.

Now, we are going to solve the first two cases. Here $x \notin L \Rightarrow x = g^k$ where $k \in \mathbb{N}$ and $k \neq pm$ for any $m \in \mathbb{N}$, $y \notin L \Rightarrow y = g^l$ where $l \in \mathbb{N}$ and $l \neq pm$ for any $m \in \mathbb{N}$, $u \in L \Rightarrow u = g^{pm_1}$ and $v \in L \Rightarrow x = v^{pm_2}$.

1) Suppose if u, v in L. We have four subcases.

- If x, y in $L \Rightarrow xuy$, xvy in L. Since L is a subgroup.

- If x, y not in $L \Rightarrow xuy$ not in L, xvy not in L or xuy in L, xvy in L by a and b.
- If x in L, y not in $L \Rightarrow xuy$ not in L, xvy not in L by a and b.
- If x not in L, $y \in L \Rightarrow xuy$ not in L, xvy not in L by a and b.

2) If u, v not in L.

- If x, y in $L \Rightarrow xuy$, xvy not in L by a and b.
- If x, y not in $L \Rightarrow xuy$, xvy not in L or xuy,xvy in L by a and b.
- If x in L, y not in $L \Rightarrow xuy$, xvy not in L or xuy, xvy in L by a and b.
- If x not in L, y in $L \Rightarrow xuy$, xvy not in L or xuy, xvy in L by a and b.

After analyzing these cases, we can deduce that only two equivalence classes exist. It implies that the elements within the subgroup belong to one class, while those outside the subgroup belong to the other. As a result, the syntactic complexity is 2. The argument applies similarly when L' is a subgroup with minor modifications. □

According to the theorem mentioned above, if L has a syntactic complexity of $\sum^* = G$, where G is a cyclic group with an order of n, then its complexity can only be either 1, 2, or n (which is the order of the group).

Furthermore, the quotient complexity is influenced by the languages recognized by the finite group automata, with G being a cyclic group of order n.

Theorem 10. *If $L = G$, then its quotient complexity is 1.*

Proof. If a language $L = G$ a group and a word w, then according to the quotient definition, the language $w^{-1}G = \{x \in \sum^* = G | wx \in G\} = G$. Since G is a cyclic group of order n and $w \in G$ for all $x \in G \Rightarrow wx \in G$, there is only one quotient, which is G. The quotient complexity of $L = G$, also known as $\kappa(L = G)$, equals 1. It indicates that the group G represents only one distinct quotient of the language. □

Theorem 11. *If L is a singleton set, then its quotient complexity is n, i.e., the group's order.*

Proof. For the language $L = \{a\}$ where $a \in \sum^* = G$ and a given word w, the quotient can be defined as $w^{-1}a = \{x \in \sum^* = G | wx = a\}$. This set will have only one element for each w. Since $w \in G$ and $a \in G$, we can write $w = g^{n_1}$ where $n_1 \in \mathbb{N}$ and $a = g^{n_2}$ where $n_2 \in \mathbb{N}$. If we have $wx = a$, then $g^{n_1}x = g^{n_2}$ and $x = g^{n_3}$ where $n_3 \in \mathbb{N}$ and $n_1 + n_3 = n_2$. Since n_1 and n_2 are fixed, only one value of n_3 will satisfy the condition. For each word, there will only be one quotient. The group's order, represented by n, will equal the number of words. Therefore, the number of distinct quotients for $L = \{a\}$ is n. This means that the quotient complexity of $L = \{a\}$ is also equal to n, denoted by $\kappa(\{a\}) = n$. □

Theorem 12. *If $L = \{ g^n | n\text{-odd}\}$ then its quotient complexity is 2 and also if $L = \{ g^n | n\text{-even}\}$ then its syntactic complexity is 2.*

Proof. According to the definition of the quotient of a language, L is the set of all g^n where n is odd and a word w, then the language $w^{-1}L = \{x \in \sum^* = G | wx \in L\}$. Here $w \in G$, since G is a cyclic group $w = g^i$, where i can be even or odd. Suppose if $w = g^i$, where i is even then for $xw \in L$ thus $x = g^j$ where j is odd. Since only even+odd= odd or odd+even = odd. So if $w = g^i$, where i is even the quotient $w^{-1}L$ contains $x = g^j$ where j is odd. Suppose if $w = g^i$, where i is odd then for $xw \in L$ thus $x = g^j$ where j is even. So if $w = g^i$, where i is odd the quotient $w^{-1}L$ contains $x = g^j$ where j is even. It results that there will be two distinct quotients. So, the quotient complexity of L is 2. Similarly, for $L =$ $\{ g^n | n$-even$\}$ the quotient complexity is 2. □

Theorem 13. *If a set L has m elements where m is less than n, L and L' are not subgroup that excludes the identity element $\{e\}$ and the whole group G and does not follow the structure outlined in theorem-3 or theorem-4, then the quotient complexity of L is n and each quotient contains m elements.*

Proof. For the language L and a given word w, the quotient can be defined as $w^{-1}L = \{x \in \sum^* = G | wx \in L\}$. We know that G is a cyclic group of order n. If $w \in G$ implies $w = g^l$ for some $l \in \mathbb{N}$. Let $L = \{g^{j_1}, g^{j_2}, ..., g^{j_m}\}$, where $j_m \in \mathbb{N}$. For $wx \in L$ we get m number of x such that $x = g^{k_i}$. That is $wx = g^l g^{k_i} = g^{j_m}$ implies $wx = g^{l+k_i} = g^{j_m}$ this can be seen as $l + k_i = j_m$. Here l is fixed, so there is m number of chances for x to get m number of elements in L. It works for each w. So there will be a n distinct quotient with m number of elements. This results in each quotient of L containing m elements, with a total quotient complexity of n. □

Theorem 14. *If L is a subgroup which is not equal to G and $\{e\}$ is the identity element then the quotient complexity is 2.*

Proof. According to the definition of the quotient $w^{-1}L = \{x \in \sum^* = G | wx \in L\} = L$. Since L is a subgroup. If $w \notin L$ there can be $x \notin L$ such that $wx \in L$(see a) for reference). This forms another quotient of L. So number of distinct quotient is 2 one is subgroup and the other represented as $w^{-1}L$, where $w^{-1}L$ equal to L' or $\neq L'$. So the quotient complexity is 2. □

Theorems of syntactic complexity and quotient complexity suggest that if G is a cyclic group of order n, then the syntactic complexity of L is the same as its quotient complexity.

7. Generalization of syntactic and quotient complexity

Theorem 15. *Let G be a group of order n. If $L = G$, then the syntactic complexity = quotient complexity = 1.*

Proof. Consider $G = \{g_1, g_2, ..., g_n\}$. For syntactic complexity we prove that $u \approx_L v$ for every u, v in $\sum^* = G$, $\forall\, x, y \in \sum^* = G \ni xuy \in L = G \Leftrightarrow xvy \in L = G$, we need to prove $[u] = [v]$ for every u and v. Let $x = g_i$, $y = g_j$, $u = g_k$, and $v = g_l$, where $i, j, k, l \in \mathbb{N}$. Since G is a group, we can see that $xuy = g_i g_k g_j = g_r \in G = L$ and $xvy = g_i g_l g_j = g_s \in G = L$. It means that $\forall\, u, v \in \sum^* = G$, $\forall\, x, y \in \sum^* = G$, $xuy \in L$

and $xvy \in L$. Therefore, $u \approx_L v$ and $[u] = [v] \; \forall \; u, v \in G$. So, only one equivalence class is $[e]$, where e is the identity element forming a monoid. Therefore, the syntactic complexity is 1.
According to the definition of the quotient $w^{-1}G = \{x \in \sum^* = G | wx \in G\} = G$. Since G is a group of order n and $w \in G \; \forall \; x \in G \Rightarrow wx \in G$. Therefore, there is only one quotient, which is G.The complexity of the quotient of a language L by group G, denoted by $\kappa(L = G)$, is 1, meaning that only one distinct quotient of the language is represented by G. For any finite group G, the syntactic and quotient complexity equals 1 when $L = G$. □

Theorem 16. *Assume that G is a group with a finite order of n. If L is a subgroup, which is not equal to G, and $\{e\}$ is the identity element, then the syntactic complexity = quotient complexity = 2.*

Proof. Consider $G = \{g_1, g_2, ..., g_n\}$. For syntactic complexity, there will be four cases to solve.

- If u in L and v in L.
- If u not in L and v not in L.
- If u in L and v not in L.
- If u not in L and v in L.

The last two cases are obvious. Just choose $x = y = e$ implies $xuy \in L$ and $xvy \notin L$ or $xuy \notin L$ and $xvy \in L$. For both the cases $u \not\approx_L v$, i.e $[u] \neq [v]$. So, the subgroup elements are in one equivalence class, and the elements that are not in the subgroup are in another.
a) If x not in L and y not in L then either xy in L or xy not in L.
$x \notin L \Rightarrow x = g_k$ where $k \in \mathbb{N}$ and $k \neq m_i$ for any $m_i \in \mathbb{N}$ and $y \notin L \Rightarrow y = g_l$ where $l \in \mathbb{N}$ and $l \neq m_i$ for any $m_i \in \mathbb{N}$. Now $xy = g_k g_l = g_r$ where $r \in \mathbb{N}$. If $r \neq m_i$ for any $m_i \in \mathbb{N}$ then $xy \notin L$. However, we have this case also that $k + l = m_i$ for some m_i. In this case $xy \in L$.
b') If x in L (x not in L) and y not in L (y in L) then xy not in L.
$x \in L \Rightarrow x = g_m$ where $m \in \mathbb{N}$ and $y \notin L \Rightarrow y = g_l$ where $l \in \mathbb{N}$ and $l \neq m \; \forall \; m \in \mathbb{N}$. Now $xy = g_m g_l = g_k \notin L$. Suppose if $g_m g_l = g_k \in L$. Since L is a subgroup $g_m^{-1} \in L$. Operate with g_m^{-1} on both sides in $g_m g_l = g_k$ becomes $g_m^{-1} g_m g_l = g_m^{-1} g_k$. Then $g_l = g_m^{-1} g_k \in L$ which is the contradiction to that $g_l \notin L$. This results $xy \notin L$.
1) Suppose if u, v in L. We have four subcases.

- If x, y in $L \Rightarrow xuy$, xvy in L. Since L is a subgroup.
- If x, y not in $L \Rightarrow xuy$ not in L, xvy not in L or xuy in L, xvy in L by a' and b'.
- If x in L, y not in $L \Rightarrow xuy$ not in L, xvy not in L by a' and b'.
- If x not in L, $y \in L \Rightarrow xuy$ not in L, xvy not in L by a' and b'.

2) If u, v not in L.

- If x, y in $L \Rightarrow xuy$, xvy not in L by a' and b'.
- If x, y not in $L \Rightarrow xuy$, xvy not in L or xuy,xvy in L by a' and b'.

- If x in L, y not in $L \Rightarrow xuy$, xvy not in L or xuy, xvy in L by a' and b'.
- If x not in L, y in $L \Rightarrow xuy$, xvy not in L or xuy, xvy in L by a' and b'.

Based on these cases, we can conclude that only two equivalence classes exist. It means that the elements in the subgroup belong to one class, while the elements not in the subgroup belong to the other class. Therefore, the syntactic complexity is 2. The same proof follows in the other case where L' is a subgroup with minor modifications.

If the language is represented by a subgroup group of G and a $w \in L$, then the quotient $w^{-1}L = \{x \in \sum^* = G | wx \in L\} = L$. Since L is a subgroup. If $w \notin L$, there can be $x \notin L$ such that $wx \in L$(see a) for reference). It forms another quotient of L. So, the number of distinct quotients is 2. One is subgroup, and the other is represented as $w^{-1}L$, where $w^{-1}L$ is equal to L' or $\neq L'$. Consequently, the syntactic complexity equals the quotient complexity, which is 2 in any finite group G where L is a subgroup. □

Theorem 17. *If L is singleton then the syntactic complexity = the quotient complexity = n (order of the group).*

Proof. For syntactic complexity we have to prove for every $u, v \in \sum^* = G \; \exists \; x, y \in \sum^* \ni xuy \in L$ and $xvy \notin L$ or $xuy \notin L$ and $xvy \in L$. Choose $x = u^{-1}$ then $xuy = u^{-1}uy = y$ and $xvy = u^{-1}vy$. Now choose $y = a$ where $a \in L$. This implies $xuy = y = a \in L$ and $xvy = u^{-1}va \notin L$ since L is singleton. If $u^{-1}va \in L$ then $u^{-1}va = a$ implies $u = v$ which is a contraction. So $u \not\approx_L v$, i.e $[u] \neq [v] \; \forall \; u, v$. This results in the fact that each element has its equivalence class. So, the syntactic complexity is n. Considering $L = \{a\}$ where $a \in \sum^* = G$ and w in $\sum^* = G$, the quotient $w^{-1}a = \{x \in \sum^* = G | wx = a\}$. This set will have only one element for each w. We know that $w \in G$ and to get $wx = a$, $x = w^{-1}a$ which satisfies the condition. It works for each w. So, there will be n distinct quotients. This results in the quotient complexity being n. Therefore, the syntactic complexity = the quotient complexity = n (group order) if L is a singleton. □

8. Conclusion

To summarize, we have effectively established the finite group automata structure for every finite group. It includes using state complexity and accepting state complexity as fundamental measures. In addition, we have calculated the syntactic and quotient complexity of cyclic groups. Although our primary focus has been on this particular group, our methodology is adaptable and sets the groundwork for expanding these complexities to other finite groups. This extension will enable us to comprehensively research and determine the complexities of all finite groups, providing a comprehensive understanding of their automata-theoretic properties.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

References

[1] S. Yu, State complexity of regular languages, J. Autom. Lang. Comb. 6 (2001), 221–234. https://doi.org/10.25596/JALC-2001-221.

[2] S. Yu, Q. Zhuang, K. Salomaa, The state complexities of some basic operations on regular languages, Theor. Comput. Sci. 125 (1994), 315–328. https://doi.org/10.1016/0304-3975(92)00011-f.

[3] J. Dassow, On the number of accepting states of finite automata, J. Autom. Lang. Comb. 21 (2016), 55–67. https://doi.org/10.25596/JALC-2016-055.

[4] M. Hospodár, M. Holzer, The ranges of accepting state complexities of languages resulting from some operations, in: C. Câmpeanu (Ed.), Implementation and Application of Automata, Springer, Cham, 2018: pp. 198–210. https://doi.org/10.1007/978-3-319-94812-6_17.

[5] M. Holzer, C. Rauch, The range of state complexities of languages resulting from the cascade product-the unary case (extended abstract), in: S. Maneth (Ed.), Implementation and Application of Automata, Springer, Cham, 2021: pp. 90–101. https://doi.org/10.1007/978-3-030-79121-6_8.

[6] C. Rauch, M. Holzer, On the accepting state complexity of operations on permutation automata, Electron. Proc. Theor. Comput. Sci. 367 (2022), 177–189. https://doi.org/10.4204/EPTCS.367.12.

[7] J. Brzozowski, B. Li, D. Liu, Syntactic Complexities of Six Classes of Star-Free Languages, J. Autom. Lang. Comb. 17 (2012), 83–105. http://hdl.handle.net/10012/12625.

[8] J. Brzozowski, Quotient complexity of regular languages, Electron. Proc. Theor. Comput. Sci. 3 (2009), 17–28. https://doi.org/10.4204/eptcs.3.2.

[9] J. Brzozowski, B. Li, Y. Ye, Syntactic complexity of prefix-, suffix-, bifix-, and factor-free regular languages, Theor. Comput. Sci. 449 (2012), 37–53. https://doi.org/10.1016/j.tcs.2012.04.011.

[10] J. Brzozowski, G. Jirásková, B. Li, Quotient complexity of ideal languages, Theor. Comput. Sci. 470 (2013), 36–52. https://doi.org/10.1016/j.tcs.2012.10.055.

[11] A.V. Kelarev, On incidence rings of group automata, Bull. Austral. Math. Soc. 67 (2003), 407–411. https://doi.org/10.1017/s0004972700037217.

[12] J. E. Hopcroft, R. Motwani and J. D. Ullman, Introduction to automata theory, languages, and computation, Addison-Wesley, Reading, 2000.

[13] J.A. Gallian, Contemporary abstract algebra, Cengage India Private Limited, Noida, 2019.