Finite Automata and Their Graphs

Precious Pelumi Adelaja

Submitted to the
Institute of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Science
in
Applied Mathematics and Computer Science

Eastern Mediterranean University
July 2023
Gazimağusa, North Cyprus

Approval of the Institute of Graduate Studies and Research

Prof. Dr. Ali Hakan Ulusoy
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science in Applied Mathematics and Computer Science.

Prof. Dr. Nazım Mahmudov
Chair, Department of Mathematics

We certify that we have read this thesis and that in our opinion it is fully adequate in scope and quality as a thesis for the degree of Master of Science in Applied Mathematics and Computer Science.

Prof. Dr. Benedek Norbert Nagy
Supervisor

Examining Committee

1. Prof. Dr. Rashad Aliyev

2. Prof. Dr. Benedek Nagy

3. Asst. Prof. Dr. Sedef Emin

ABSTRACT

Finite automata are one of the most known topics of (theoretical) computer science. Their presentations by their graphs are well known and widely used. In this thesis, these representations will be analyzed from a graph theoretical point of view. In this way, the work is also related to graph theory and combinatorics. The finite automata are used to accept formal languages and the accepted language class may be characterized by some graph theoretical properties of the used automata.

We define five different types of Strongly connected automata as, Line directed, Directed cycle, Bidirected cycle, Starred, and Floral automata, and their constructions were studied from a graphical point of view. The class of languages accepted by these automata was also studied by regular expressions. While keeping the initial state constant, we vary each state of the automata to be the final state, and using the elimination method, we obtain the regular expression these automata accept. We also define the concept of Hamiltonian-like words as words accepted by automata that pass through all states of the automata. Then, we studied the Hamiltonian-like words accepted by the defined strongly connected automata while varying the final state. Properties of these Hamiltonian-like words, such as length, Kleene stars, and cycles, were investigated.

Keywords: Finite Automata, Strongly Connected Automata, Hamiltonian-like Words.

ÖZ

Sonlu otomatlar (sonlu durum otomatları) (teorik) bilgisayar bilimlerinin en bilinen konularından biridir. Grafikler aracılığı ile temsiliyetleri iyi bilinir ve yaygın olarak kullanılır. Bu tezde, bu sunumlar bir teorik grafik bakış açısıyla analiz edilecektir. Bu şekilde çalışma, grafik teorisi ve kombinatorik araştırma konularına da bağlıdır. Otomatlar, resmi dilleri kabul etmek için kullanılır ve kabul edilen dil sınıfı, kullanılan otomatların bazı grafik teorik özellikleri ile karakterize edilebilir.

Aralarında güçlü bağlantı bulunan beş farklı totomat çeşidini; Çizgi yönelimli, Yönlü çevrim, Çift yönlü çevrim, Yıldızlı ve Çiçekli otomatları tanımladık ve bunların yapıları grafiksel bir bakış açısıyla incelendi. Bu otomatlar tarafından kabul edilen dil sınıfı düzenli dillerle de incelenmiştir. İlk durumu sabit tutarak, otomatların her durumunu son durum olacak şekilde değiştirerek ve yok etme yöntemini kullanarak, bu otomatların kabul ettiği düzenli dil sınıfını elde ettik. Hamilton benzeri kelimeler kavramını da otomatların kabul ettiği ve otomatların tüm durumlarından geçen kelimeler olarak tanımlıyoruz. Daha sonra, son durumu değiştirirken yukarıda tanımlanan bağlantılı otomatlar tarafından kabul edilen Hamilton benzeri kelimeleri inceledik. Uzunluk, Kleene yıldızları ve döngüler gibi bu Hamilton benzeri kelimelerin özellikleri araştırıldı.

Anahtar Kelimeler: Sonlu Otomat, Yakın Bağlantılı Otomatlar, Hamiltoniyen Benzeri Kelimeler.

ACKNOWLEDGMENT

I want to express my gratitude for my supervisor Prof. Dr. Benedek Nagy 's efforts and his tolerance while I was preparing this thesis. I genuinely appreciate the help I've gotten from my parents and siblings, and I hope that someday I'll be able to make them proud.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

Σ, Δ	Alphabet
A	Automaton
DFA	Deterministic Finite Automata
d	Distance of an Automaton
$\emptyset$	Empty State
λ	Empty Word
E_w	Eulerian-like Word
EFA	Extended Finite Automata
f, F	Final State(s)
FA	Finite Automata
Q	Finite Set of States
GTG	Generalized Transition Graph
G, H	Graph
H_w	Hamiltonian-like Word
L	Language
NFA	Non-deterministic Finite Automata
Z	Number of Cycles
r	Regular Expression
ε	Set of Edges
R_Σ	Set of Regular Expression
V	Set of Vertices
h	Star Height

s	Starting State
SCA	Strongly Connected Automata
δ	Transition Function
w	Word

Chapter 1

INTRODUCTION

Fundamental models known as finite automata are used to depict systems with discrete input and output in computer science, mathematics, and engineering. The behavior of systems like software programs, communication protocols, and digital circuits are described and verified using finite automata. When data are stored in a particular structure, it is necessary that these data are connected to help reduce the time and space complexity of traversing these structures, no other concept help represent these structures than the strongly connected automata.

Even though a lot of work has been done on strongly connected automata, only a few have focused on the family of languages these automata accept, and close to no research on the relation between the family of languages and the structure of the strongly connected automata. To further investigate the structure of some strongly connected automata, we employ graph theory. Graph theory, which is the mathematical study of graphs, networks, and their properties, is closely related to finite automata, with both having related concepts such as nodes in a graph as states in automata and edges in a graph as transitions in automata. (Ádám, 1983) is one of the very few prominent researchers to have adopted the concept of graph theory to automata in this manner. Using existing definitions in graph theory and formal languages, we introduce some new concepts such as distance and Hamiltonian-like words in order to better explore this relationship.

Moreover, researching strongly connected automata can aid in system performance optimization. A system's performance can be improved by reducing the complexity of the system by limiting the number of states in an automaton. In this context, strongly connected automata are particularly helpful since they offer a framework for state minimization based on the graph's structure.

Algorithms such as searching and sorting on data structures like trees, binary trees, heaps, stacks, or queues will perform better as the connectedness of the structure increases. It is possible that we want to perform a task (or reach a state) but the task depends on every other task in the system, hence the need to traverse the whole system before performing this task (or accepting a word). This is known as *topological sorting*, even though it is mainly implemented in directed acyclic graphs, we introduce the same concepts to our strongly connected automata as Hamiltonian-like words and investigate the properties of these words.

In the next chapter, we define Hamiltonian-like words as words accepted by automata that pass through all states of the automata. We will then define some five subtypes of strongly connected automata and study the Hamiltonian-like words accepted by these automata, while varying the final state. We will investigate properties of these Hamiltonian-like words, such as their length, Kleene stars, and cycles. This will help us gain a deeper understanding of the behavior of these automata and how they relate to graph theory.

The study of Hamiltonian-like words in strongly connected automata has important implications for the performance of these systems. By analyzing the length of these words, we can identify areas where the system may be unnecessarily complex or redundant and simplify the design of the automaton to improve its performance. Similarly, by studying the cycles in these words, we can gain insights into the

connectivity of the automaton and identify critical states that are essential to its performance.

Chapter 2

PRELIMINARIES

2.1 Basic Definitions

An *alphabet* Σ is a nonempty set of finite elements called letters or symbols, and a finite sequence of zero or more letters are called a *word* w, or *string* over the alphabet. A word is said to be the *empty word* λ if it has zero letter sequence. If Σ is an alphabet of English letters, then all English language words and words like λ, aaa, bz, $czzzyy$ and so on are all words over the English alphabet Σ and can be represented as Σ^* or Σ^+(Σ^* excluding λ). Both Σ^* and Σ^+ are infinite.

The *catenation* (or *concatenation*) of two words x and y over an alphabet Σ is also a word over the alphabet written as xy, from the juxtaposition of x and y after one another. Concatenation is an associative operation, and with λ as identity element $w\lambda = \lambda w = w$, for all word w over Σ; w^i, for $i \geq 0$ can be used to represent i times catenation of w, with $w^0 = \lambda$.

The *length* of a word w denoted as $|w|$ is the number of letters in w, having each letter counted as many times as it occurs. The empty word λ has a length of zero, $|\lambda| = 0$.

A *subword* x of a word w is also a word over Σ such that $w = y_1 x y_2$ where y_1 and y_2 are also words over Σ and maybe empty words. If $y_1 = \lambda$ or $y_2 = \lambda$ then x is also called a *prefix* of w or *suffix* of w, respectively. It is worth noticing that a word w and λ are subwords, prefixes, and suffixes of w itself called trivial, while every other subword, prefix, or suffix is nontrivial.

We define a *language* L over alphabet Σ as a set of words over Σ, denoting the empty language as $\emptyset$, and the universal language as Σ^*, which is the language consisting of all words over Σ. The *cardinality* of language L is denoted by $|L|$.

Two languages $L_1, L_2 \subseteq \Sigma^*$ can be concatenated as

$$L_1 L_2 = \{w_1 w_2 \mid w_1 \in L_1 \; and \; w_2 \in L_2\}.$$

The n^{th} *power* of a language L denoted L^n, for a positive integer $n \geq 0$ can be defined as

(i) $L^0 = \{\lambda\}$.

(ii) $L^n = L^{n-1}L$, for $n > 0$.

(iii) $L^+ = \bigcup_{i=1}^{\infty} L^i$, Kleene plus.

(iv) $L^* = \bigcup_{i=0}^{\infty} L^i$, star or Kleene closure of a language.

A singleton language contains only one word, which can be used to denote the language if confusion is not made. xLy is equivalent to $\{x\}L\{y\}$, for $x, y \in \Sigma^*$ and $L \subseteq \Sigma^*$.

A mapping $h: \Sigma^* \longrightarrow \Delta^*$ of two finite alphabets Σ and Δ is called a mo*rphism* if $h(xy) = h(x)h(y)$ for all $x, y \in \Sigma^*$.

2.2 Finite Automata

There are two main categories of mechanisms for defining languages in formal language theory: *acceptors* and *generators*. The acceptors are finite automata (FA) while the generators are regular expressions, and grammars.

We discuss two main types of finite automata namely deterministic finite automata (DFA) and nondeterministic finite automata (NFA). A special case of nondeterministic finite automata which allows $\lambda - transition$ and called $\lambda - NFA$ will also be explained. (Rozenberg G., 1997) also shows that these automata accept exactly the same family of languages. (pp. 50)

2.2.1 Deterministic Finite Automata

Deterministic finite automata are finite automata whose next state is always uniquely determined by the current state and the current input symbol. We define a *deterministic finite automaton* (DFA) A_1 as a quintuple $(Q, \Sigma, \delta, s, F)$, where

Q is the finite set of states;

Σ is the input alphabet;

$\delta: Q \times \Sigma \rightarrow \mathrm{Q}$ is the state transition function;

$s \in Q$ is the starting state; and

$F \subseteq Q$ is the set of final states.

The transition function $\delta: Q \times \Sigma \rightarrow \mathrm{Q}$ shows how the automata accepts an input at the current state to transition to the next state. Let us consider as an example, a DFA A_1 that reads strings of $0's$ and $1's$ only, i.e. $\Sigma = \{0, 1\}$, with set of states $\mathrm{Q} = \{q_0, q_1, q_2\}$, if the initial state $s = q_0$, and the set of final states $F = \{q_2\}$, then, we define the transition function as $\delta(q_0, 0) = q_1$, $\delta(q_0, 1) = q_0$, $\delta(q_1, 0) = q_1$, $\delta(q_1, 1) = q_2$, $\delta(q_2, 0) = q_1$, and $\delta(q_2, 1) = q_0$. The transition function can also be represented as a table and diagram as shown in Table 1 and Figure 1 respectively.

Table 1: Transition table of a DFA

$Q \backslash \Sigma$	0	1
q_0	q_1	q_0
q_1	q_1	q_2
q_2	q_1	q_0

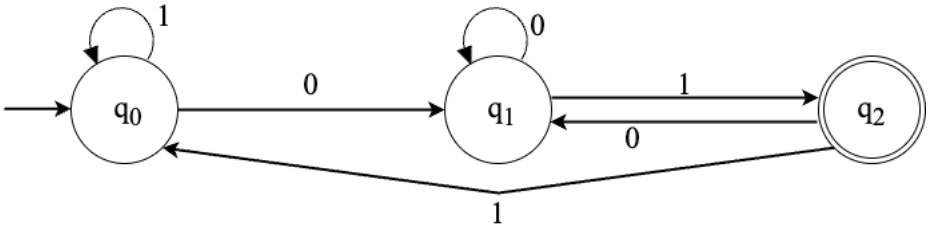

Figure 1: Transition diagram of a three states DFA

A word w is accepted by a DFA if it starts from the initial state and terminates in one of the final states. The automata A_1 accepts binary words that end in '01'. A_1 is also referred to as a *complete* DFA because its transition function is defined for every pair $Q \times \Sigma$, even though it is not a requirement for a deterministic finite automaton in general.

An Extended transition function $\delta^*: Q \times \Sigma^* \rightarrow Q$ reads a string rather than a single symbol as the second argument to transit from one state to another, inductively as follows. We set $\delta^*(q, \lambda) = q$ and $\delta^*(q, xa) = \delta(\delta^*(q, x), a)$, for $q \in Q$, $a \in \Sigma$, and $x \in \Sigma^*$.

The language accepted by A_1, $L(A_1)$, is the set of all words accepted by A_1 represented as $\{w \in \Sigma^* \mid \delta^*(s, w) = f \text{ for some } f \in F\}$. If L_{DFA} denotes the collection of all languages that some DFA accept, then there is a complete DFA that accepts each $L \in L_{DFA}$. Also, L_{DFA} is closed under union, intersection, and complementation. (Rozenberg G., 1997)

2.2.2 Nondeterministic Finite Automata

Nondeterministic finite automata (NFA) are a generalization of DFA with possible transitions greater than one for a given state and input symbol. We define *nondeterministic finite automaton* (NFA) A_2 as a quintuple $(Q, \Sigma, \delta, s, F)$, where

Q is the finite set of states;

Σ is the input alphabet;

$\delta: Q \times \Sigma \rightarrow 2^Q$ is the state transition function;

$s \in Q$ is the starting state; and

$F \subseteq Q$ is the set of final states.

The transition function is the only difference between the definition of DFA and NFA where the NFA with a single input can transition to more than one state represented as the power set of Q, 2^Q.

Let us consider as an example, an NFA A_2 that reads strings of $0's$ and $1's$ only, i.e. $\Sigma = \{0, 1\}$, with set of states $Q = \{q_0, q_1, q_2\}$, if the initial state $s = q_0$, and the set of final states $F = \{q_2\}$, then, we define the transition function as $\delta(q_0, 0) = \{q_0, q_1\}$, $\delta(q_0, 1) = \{q_0\}$, $\delta(q_1, 0) = \emptyset$, $\delta(q_1, 1) = \{q_2\}$, $\delta(q_2, 0) = \{q_0\}$, and $\delta(q_2, 1) = \{q_0\}$. The transition function can also be represented as a table and diagram as shown in Table 2 and Figure 2 respectively.

Table 2: Transition table of an NFA

$Q\backslash\Sigma$	0	1
q_0	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\emptyset$	$\{q_2\}$
q_2	$\{q_0\}$	$\{q_0\}$

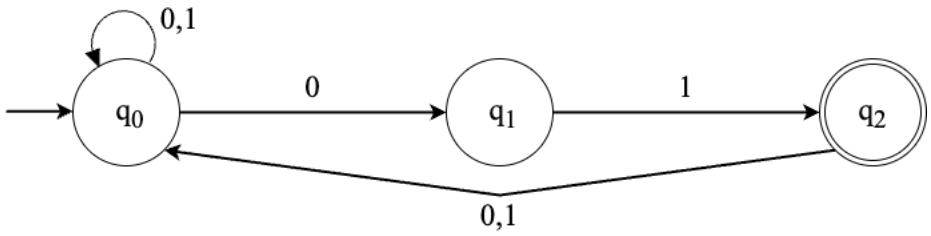

Figure 2: Transition diagram of a three states NFA

The equivalence of DFA and NFA is as a result of considering elements of 2^Q or the value of the NFA transition function to be either singleton or empty set.

For NFA, an Extended transition function $\delta^*: Q \times \Sigma^* \rightarrow 2^Q$ reads a string rather than a single symbol as the second argument to transit from one state to a set of states, similar to what was defined for DFA.

The language accepted by NFA A_2, $L(A_2)$, is the set of all strings which are accepted by the automata.

$$L(A_2) = \{w \in \Sigma^* | \delta^*(s, w) \cap F \neq \emptyset\}$$

Automata that accept the same language are said to be equivalent. The DFA A_1 represented by Figure 1, and the NFA A_2 of Figure 2 accepts exactly the same language. If the family of languages accepted by the NFA is L_{NFA} and for DFA L_{DFA} then $L_{DFA} = L_{NFA}$. (Rozenberg G., 1997)

NFA can be further generalized to have λ-transitions. These are state transitions without reading any input symbols.

We define a *nondeterministic finite automaton* with λ-transitions (λ-NFA) A_3 as a quintuple $(Q, \Sigma, \delta, s, F)$ having Q, Σ, s, F as the same as an NFA and the transition function as $\delta: Q \times (\Sigma \cup \{\lambda\}) \rightarrow 2^Q$.

Let us consider an example of λ-NFA A_3 that reads strings of $0's$ and $1's$ just like A_2, i.e. $\Sigma = \{0, 1\}$, with a set of states $Q = \{q_0, q_1, q_2\}$, if the initial state $s = q_0$, and the set of final states $F = \{q_2\}$, then, we define the transition function as $\delta(q_0, 0) = \{q_0, q_1\}$, $\delta(q_0, 1) = \{q_0\}$, $\delta(q_1, 0) = \emptyset$, $\delta(q_1, 1) = \{q_2\}$, $\delta(q_1, \lambda) = \{q_2\}$, $\delta(q_2, 0) = \emptyset$, $\delta(q_2, \lambda) = \{q_0\}$, and $\delta(q_2, 1) = \emptyset$. The transition function can also be represented as a table and diagram as shown in Table 3 and Figure 3 respectively.

Table 3: Transition table of a λ-NFA

$Q\backslash\Sigma$	0	1	λ
q_0	$\{q_0, q_1\}$	$\{q_0\}$	$\emptyset$
q_1	$\emptyset$	$\{q_2\}$	$\emptyset$
q_2	$\emptyset$	$\emptyset$	$\{q_0\}$

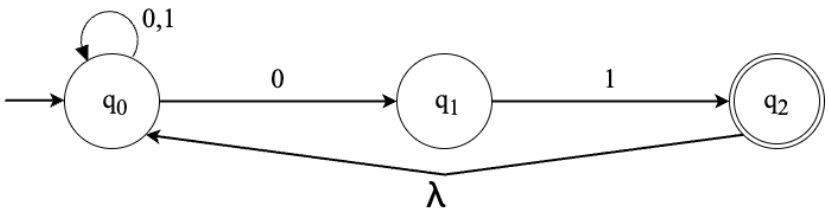

Figure 3: Transition diagram of a three states λ-NFA

The above λ-NFA accepts the same family of language as A_1 and A_2 but generally uses fewer transitions and easier to construct for users and to implement as the structure might get more complex.

The language accepted by λ-NFA A_3, $L(A_3)$, is the set of all strings which are accepted by the automata.

$$L(A_3) = \{w \in \Sigma^* | \delta^*(s, w) \cap F \neq \emptyset\}$$

2.3 Regular Expressions

Computer programs may easily implement finite automata in a number of different forms. A DFA can be implemented using a case or switch statement and a matrix can be used to express an NFA and manipulate it using matching matrix operations. However, users cannot easily specify finite automata in any of the aforementioned forms in a sequential manner. A more comprehensive expression in sequential form might be more appropriate in this situation than a specification of a

finite automaton. For example, the language accepted by DFA A_1 can be expressed as (0+1)*01, and such expressions are called regular expressions.

A *regular expression* r over an alphabet $\sum$ and the language $L(r)$ it denotes, can be define as follows:

(i) For language $L(r) = \emptyset, r = 0$.

(ii) For language $L(r) = \{\lambda\}, r = \lambda$.

(iii) For $a \in \Sigma$, if language $L(r) = \{a\}$ then $r = a$.

If r_1 and r_2 are regular expressions and $L(r_1)$ and $L(r_2)$ are the languages they represent, respectively. Then

(iv) $r = (r_1 + r_2)$ denotes the language $L(r) = L(r_1) \cup L(r_2)$.

(v) $r = (r_1 \cdot r_2)$ denotes the language $L(r) = L(r_1)L(r_2)$.

(vi) $r = r_1^*$ denotes the language $\big(L(r_1)\big)^*$.

We suppose that * has precedence over the · and +, and that · has precedence over +. When omitting a pair of parentheses won't lead to confusion, it can be done. Also, when writing regular expressions, we typically omit the · symbol.

Two regular expressions r_1 and r_2 over $\sum$ are said to be equivalent, $r_1 = r_2$, if $L(r_1) = L(r_2)$. The languages that are denoted by regular expressions are called regular languages and denote the family of regular languages as L_{REG}. (Rozenberg G., 1997)

The star operator is arguably the most important of the three regular expression operators. Without the star operator, regular expressions can only define finite languages. The number of nested stars in an expression, also known as the star height of the expression, is a natural way to gauge how complex the expression is. One of the most intriguing issues in formal language theory is the question of star height. Some

open issues continue to draw researchers. Here, we will provide the fundamental definition.

The *star height* of a regular expression r over the alphabet Σ, denoted $h(r)$, is a nonnegative integer defined recursively as follows:

(1) $h(r) = 0$, if $r = \emptyset, \lambda,$ or a for $a \in \Sigma$.

(2) $h(r) = \max\big(h(r_1), h(r_2)\big)$, if $r = (r_1 + r_2)$, if $r = (r_1 r_2)$, where r_1 and r_2 are regular expressions over Σ

(3) $h(r) = h(r_1) + 1$, if $r = r_1^*$ and r_1 is a regular expression over Σ.

The star height of a regular language L, denoted $h(L)$, is the least integer x such that $h(r) = x$ for some regular expression r denoting L. (Rozenberg G., 1997)

Regular expressions are analogous to finite automata, in the languages they define (Kleene, 1951). A regular expression can be converted into an equivalent finite automaton using a variety of procedures, and vice versa. The transformations from a regular expression to an equivalent finite automaton and from a finite automaton to an equivalent regular expression will be discussed in the next two sections.

2.3.1 Regular Expressions to Finite Automata

Regular expression can be converted into FA using the *subset* method in three basic steps (John E. Hopcroft, Rajeev Motwani, & Jeffrey D. Ullman, 2006, p. 102).

i. Construct λ-NFA transition diagram
ii. Convert λ-NFA to NFA
iii. Convert NFA to DFA

Below are some basic constructs of finite automata that reads strings of $0's$ and $1's$, $\Sigma = \{0, 1\}$ from regular expressions;

i. When a regular expression generates a single letter, $r = 1$ then

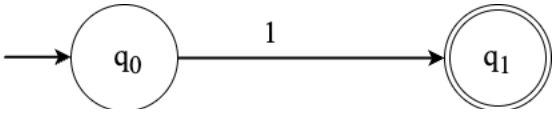

Figure 4: Transition diagram for $r = 1$

ii. When a regular expression is the concatenation of two letters, $r = 01$ then

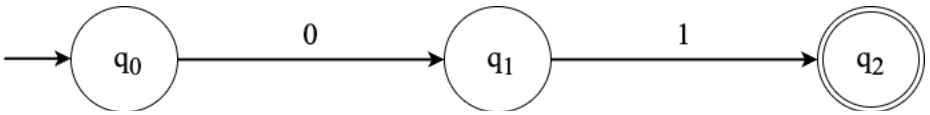

Figure 5: Transition diagram for $r = 01$

iii. When a regular expression is the union of two letters, $r = 0 + 1$ then

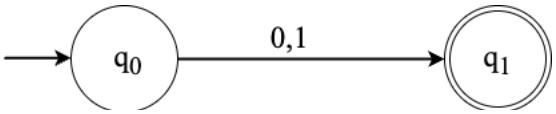

Figure 6: Transition diagram for $r = 0 + 1$

iv. When regular expression is of the form $r = (0 + 1)^*$ then

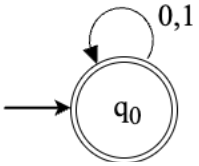

Figure 7: Transition diagram for $r = (0 + 1)^*$

As the name implies, the subset method starts by constructing λ-NFA for the innermost subset of a regular expression using the rules stated above.

2.3.2 Finite Automata to Regular Expressions

Two popular methods for converting finite automata to regular expressions are *Arden's method* (Sakarovitch, 2009) and *state elimination method* (John E. Hopcroft,

Rajeev Motwani, & Jeffrey D. Ullman, 2006, p. 98). The later will be discussed here but first we define an *extended finite automata* EFA.

Extended finite automata just like regular finite automata, is a quintuple $(Q, \Sigma, \delta, s, F)$ with the exception of the transition function defined as $\delta: Q \times Q \rightarrow \mathrm{R}_{\Sigma}$ where R_{Σ} is the set of regular expressions over Σ. EFA has a transition between pair of states labeled as a regular expression. If the transition from a state q_i to q_j is not explicitly defined then $\delta(q_i, q_j) = \emptyset$.

Using the state elimination method, finite automata are first reduced to extended finite automata by eliminating one state that is neither the starting state s nor part of the final states F. The basic idea of the process is as follows.

If o and q are two states in Q, let r_{oq} be the transition between the two states. If we are to eliminate a state $q \in Q$ such that $q \neq s$ and $q \notin F$, we denote the resulting EFA as $A' = (Q', \Sigma, \delta', s, F)$ where $Q' = Q - \{q\}$. For each pair of states $o, p \in Q'$ of an EFA

$$\delta'(o, p) = r_{op} + r_{oq} r_{qq}^{*} r_{qp}$$

This process is repeated until we are left with only two states, the initial state s and the final state f and $\delta'(s, f)$ is the regular expression equivalence of the finite automata.

The above process is governed by the following set of rules;

i. The initial state s of finite automata A must not have incoming transition. If there is a transition, we create a new state s' as the starting state and set $\delta(s', s) = \lambda$.

ii. If the automaton has more than one final state, i.e. $|F| > 1$, or an outgoing transition from the final state $f \in F$ then we add a new state f' and set it to the only final state with $\delta(f, f') = \lambda$ for each $f \in F$.

For example, we try to convert the finite automata A_1 in figure 1.2.1 to a regular expression. The initial state q_0 has self-loop, so we add a new state s' and set it as the initial state, and also the final state q_2 has an outgoing transition, so we add a new state f' and set it to the final state, as it is shown in Figure 8.

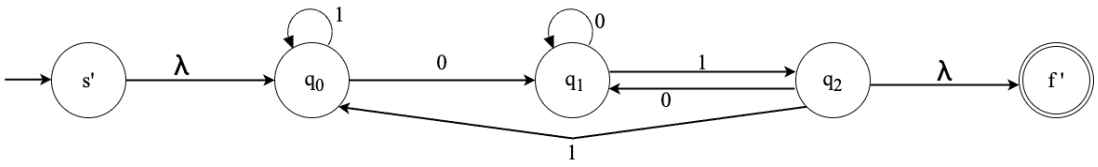

Figure 8: EFA or λ -NFA transition diagram

We choose to eliminate the state q_1 first as the elimination can be done in no particular order, followed by q_0 and finally q_2, as Figure 8 to 10 has shown.

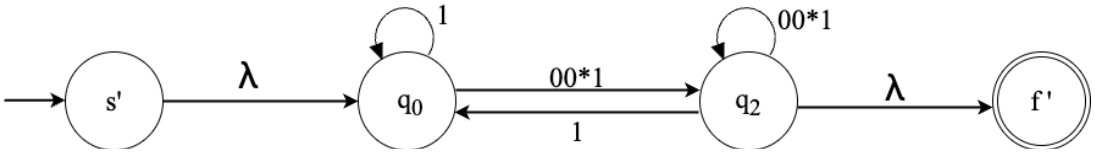

Figure 9: EFA with q_1 eliminated

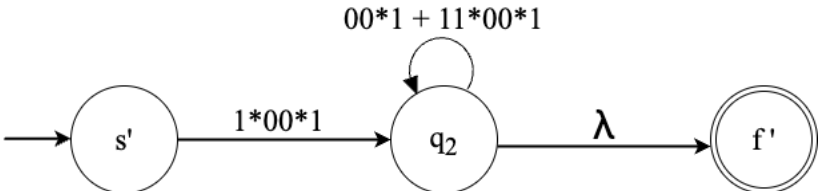

Figure 10: EFA with q_0 and q_1 eliminated

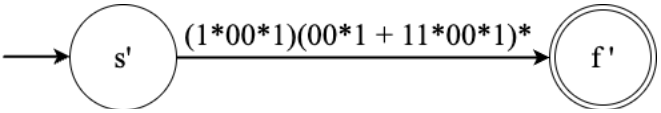

Figure 11: EFA with q_0, q_1 and q_2 eliminated

The resulting regular expression can be simplified as $(1+0)^*01$. Note that state elimination method can be used in converting DFA, NFA and λ-NFA to regular expression since they are all extended finite automata. Figure 12 shows a generic two-state automaton G^* and the regular expression that represent the language accepted by this automaton when $f \in \{q_0, q_1\}$ is given by r_0, r_1 respectively.

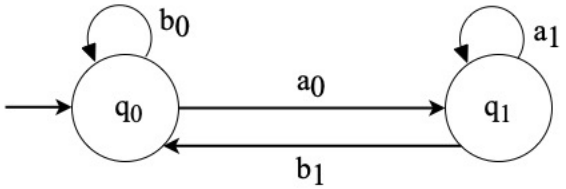

Figure 12: A generic two-state automata G^*

$$r_0 = (b_0 + \mathrm{a}_0\mathrm{a}_1^*\mathrm{b}_1)^*$$

$$r_1 = \mathrm{b}_0^*\mathrm{a}_0(a_1 + \mathrm{b}_1\mathrm{b}_0^*\mathrm{a}_0)^*$$

Since Finite automata can either be of two types: Deterministic Finite Automata (DFA) or Non-Deterministic Finite automata (NFA), and if $D = (Q_D, \Sigma, \delta_D, q_0, F_D)$ is the DFA that is constructed from NFA $N = (Q_N, \Sigma, \delta_N, \{q_0'\}, F_N)$ using the subset construction, we say $L(D) = L(N)$ and a language L is accepted by some DFA if and only if L is accepted by some NFA (John E. Hopcroft, Rajeev Motwani, & Jeffrey D. Ullman, 2006, p. 64). For every language that is defined by these regular expressions there is an automaton, and for every language that is accepted by an automaton there is also a regular expression (John E. Hopcroft, Rajeev Motwani, & Jeffrey D. Ullman, 2006, p. 92). For this reason, we consider Deterministic finite automata (DFA) in this thesis and often mean DFA when referring to Finite automata in general.

2.4 Graphs

A *graph* often called vertex-edge graph is a connection of a set of nodes that can be represented mathematically as a pair (V, ε) of a nonempty set Vcalled *vertices* (nodes or points), and a set of elements connecting the vertices ε called *edges* (arcs or lines). If $e \in \varepsilon$ is an edge connecting two vertices $v, u \in V$, then v, u are the *end vertices*, and we denote e by (v, u) or without set notation as vu. The two vertices v, u are each *incident* to e, and edge e is also incident to vertex v and u also, v and u are *adjacent* to each other since they share an edge e. Two vertices are adjacent if they are both incident to an edge and two edges are adjacent to each other if they are both incident to the same vertex. The degree of a vertex, *deg* v is the number of edges incident to the vertex which could be odd or even degrees depending on the number of edges if odd or even respectively. If the degree of a vertex is zero then the vertex is called *isolated*. Denoted as $G(V, \varepsilon)$ is a graph G with finite set of vertices V and finite set of edges ε hence G is always finite. Mostly, a graph is represented as a diagram, an example is the Figure 13.

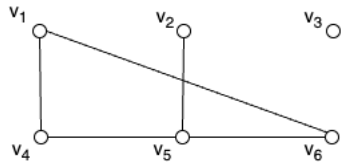

Figure 13: A graph G

The figure represents the graph $G(V, \varepsilon)$ with vertices $V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$ and edges $\varepsilon = \{v_1v_4, v_1v_6, v_2v_5, v_4v_5, v_5v_6\}$. Note that v_3 is an isolated vertex because it has no edge incident to it and hence a degree of zero. The vertices $v_1, v_2, v_3, v_4, v_5, v_6$ have a degree of 2,1,0,2,3,2, respectively. According to our definition, vertex v_1 is adjacent to v_4 and v_6, and incident to edges v_1v_4 and v_1v_6.

The above definition of a graph can be more specified as a graph that has no *multiple edges* between two vertex or a *self-loop* of a vertex to itself, hence called a *simple graph*.

A graph is referred to as *multigraph* if it has multiple edges and as *pseudograph* if it has multiple edges and/or self-loops. (Edgar G. Goodaire, & Michael M. Parmenter, 1998)

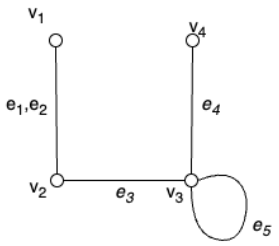

Figure 14: A psuedograph

Figure 14 is a pseudograph $G(V, \varepsilon)$ with vertices $V = \{v_1, v_2, v_3, v_4\}$ and edges $\varepsilon = \{e_1, e_2, e_3, e_4, e_5\}$. Edges e_1 and e_2 are incident to vertices v_1 and v_2 hence multiple edges with the two edges sharing a single graphical edge with multiple labeling. Edge e_5 is a loop because it is incident to only v_3. v_3 also has a degree of four, counting the self-loop as it enters and leaves the vertex.

A graph may be *directed* or *undirected*. If the edges of a graph do not have direction, then the graph is undirected, else if a graph has an arrow at one or both ends of its edges as a form of directedness, it is said to be a directed graph. Figure 13 and Figure 14 are examples of an undirected graph.

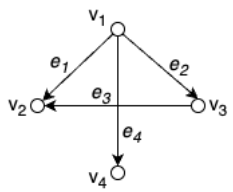

Figure 15: A directed graph

Figure 15 is an example of a directed graph $G(V, \varepsilon)$ with vertices $V = \{v_1, v_2, v_3, v_4\}$ and edges $\varepsilon = \{e_1, e_2, e_3, e_4\}$. The properties of a graph discussed up to this point apply to both directed and undirected graphs.

For a directed graph, the *indegree* of a vertex is the number of edges incident to the vertex with direction oriented towards the vertex, while the *outdegree* is the number of edges incident to the vertex with the edge direction oriented away from the vertex. Vertex v_3 in Figure 15 has an indegree of one and also an outdegree of one.

The *diameter* of a graph G is the length of the shortest path between the most distant vertices. If G can be represented as the union of two subgraphs G_1, G_2 (each having at least one edge) such that G_1, and G_2 have only one vertex v in common, then v is a *cut vertex*. If a vertex v of graph G is contained in every cycle of G, then v is called *pancyclic vertex*. (Ádám, 1973)

If H is a subgraph of a graph G, and H contains all the vertices (but possibly not all the edges) of G, then H is an *e-subgraph* of G. The subgraph H of G is called a p-subgraph if the following condition is satisfied: whenever vertex v_1 and v_2 are contained in H and the edge e of G is incident to v_1 and v_2, then e is contained in H too. (Ádám, 1970)

In a pseudograph, a *walk* is an alternating sequence of vertices and edges that starts and ends with a vertex, with each vertex except the last vertex been incident with the edge that comes after and with each vertex (except the first vertex) being incident with the edge that came before. If a walk starts with vertex u and ends with vertex v it can be denoted as a uv-walk. The number of edges in a walk determines its length. If the start and last vertices of a walk are the same, it is *closed*; otherwise, it is *open*. A *path* is a walk with distinct vertices, whereas a *trail* is a walk with distinct edges. A *circuit* is a trail that is closed. A *cycle* is a circuit in which no vertex appears more than

once but the first vertex appears exactly twice (at the beginning and the end). A cycle having n vertices is known as an n-cycle. If n is even, the cycle is even; if n is odd, then the cycle is odd. A path is always a trail because if a walk's vertices are all different, then its edges must also be unique. The opposite is not true, a trail does not have to be a path. The *number of cycles* an edge e belongs to is denoted by $Z(e)$, and $Z(v)$ for the number of cycles in which vertex v occurs. (Ádám, 1970)

In a pseudograph, an *Eulerian circuit* or *Eulerian cycle* is a circuit that includes every vertex and every edge of the graph in the circuit. In other words, it is a trail that reaches each graph edge exactly once. A pseudograph with an Eulerian circuit is known as an *Eulerian pseudograph*.

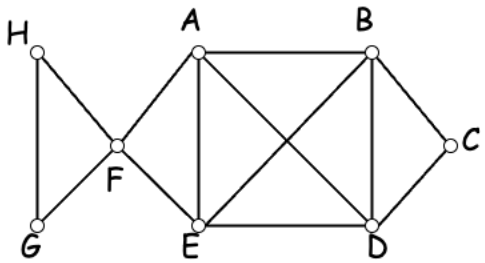

Figure 16: An eulerian graph

We use Figure 16 to illustrate the difference between a circuit and an Eulerian circuit. *A B C D E F G H F A* is a circuit but not Eulerian even though it has all vertices, four edges were left out of the circuit, while the vertices *B, C, D, E, F, G, H, F,* are referred to as *inner vertices* of the circuit. On the other hand, the circuit *A B C D E F G H F A D B E A* is Eulerian and hence the figure is an Eulerian or Euler graph.

A pseudograph is said to be *connected* if and only if there is a path between any two vertices. An example of connected pseudograph is Figure 16 that has a path with any two of its vertices as endpoints. Figure 17 is an example of a pseudograph that is connected but not Eulerian, as vertex *A* only has one edge incident to it. Any

circuit that uses the edge BA is trapped and can not continue without reusing the edge again.

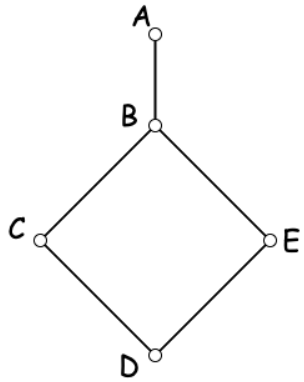

Figure 17: A graph without eulerian circuit

Thus, it is not enough for a graph to be connected for it to be Eulerian, the degree of the vertices also matters, as Figure 17 traps any circuit at vertex A because it has a degree of one. Every time we encounter a vertex (other than the one from which we started) on an Eulerian circuit, we either depart on a loop and quickly return, never crossing that loop again, or we leave on an edge different from the edge by which we entered, crossing neither edge again. Therefore, it is possible to pair any edge that is incident to a vertex in respect to the circuit flow, the edge that enters the vertex and the edge that leaves, and also we can pair the edge that leaves our circuit first vertex and the edge that enters the circuit last vertex. So, in addition to being connected, an Eulerian graph also needs to contain vertices of even degree. In contrast, a connected graph with even degree vertices must be Eulerian. We can now look at the connected graph in Figure 16 which has only even degree vertices, but the graph of Figure 17 has odd vertices as well.

A *Hamiltonian cycle* in a graph is a cycle that contains every vertex of the graph and it visits every vertex exactly once except for maybe the first and last vertex.

A *Hamiltonian graph* is one with a Hamiltonian cycle and it is synonymous to *Hamiltonian circuit*, *Hamilton cycle*, or *Hamilton circuit*.

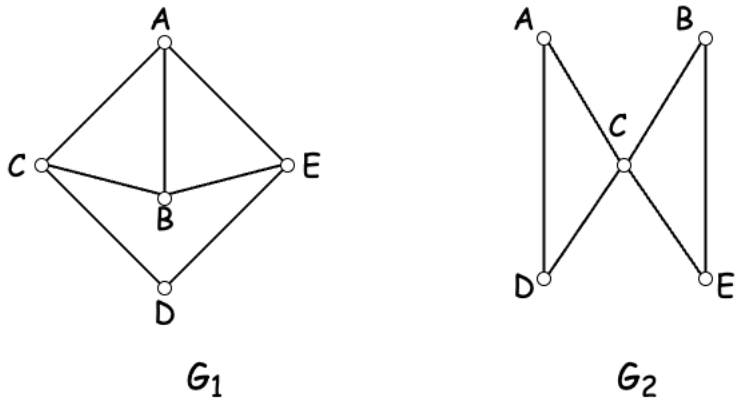

Figure 18: Hamiltonian graph and non-hamiltonian graph

In Figure 18 the graph G_1 is a Hamiltonian graph with Hamiltonian cycle *A B C D E A*. G_2 is not Hamiltonian. One way to see it is to split the graph G_2 into two subgraphs with vertices $\{A, D, C\}$ and $\{C, B, E\}$. It is impossible to have a circuit that will include all vertices from both subgraphs without passing through vertex C more than once in order to include all edges. In contrast to how easy it is to show a graph is Eulerian, there is no known theorem to show a graph is Hamiltonian, although there are properties of cycles that can be used to determine if a graph is not Hamiltonian. (Edgar G. Goodaire, & Michael M. Parmenter, 1998)

Finite automata take on different form when represented using transition graphs yielding specific property. One of such form is *Strongly Connected Automata* SCA and we define it to be the quintuple $(Q, \Sigma, \delta, q_0, F)$ such that for any pair of state $q_i, q_j \in Q$ there exists an element $w \in \Sigma^*$ such that $\delta^*(q_i, w) = q_j$ (Ito, 1978). That is, every other state can be reached from any state of Strongly connected automata. Obviously, Strongly connected automata cannot have a trap state, as every state must have an outgoing degree greater than one. Strongly connected automata has wide

variety of application especially in theoretical computing such as strongly connected synchronizing automata (Rodaro, 2018) and strongly connected Moore automata (Volkov, 2008).

2.5 New Definitions

In this subsection, we show the new concepts we use in our study. We use strongly connected automata A of the form $(Q, \Sigma, \delta, q_0, q_f)$ where the set of final states F has only one state q_f.

The **distance** $d \in \mathbb{Z}^+ \cup \{0\}$ of the strongly connected automaton A, is defined as the length of the shortest word $w \in \Sigma^*$ from q_0 to q_f such that $\delta^*(q_0, w) = q_f$ (this word represents a walk from the graph theoretical point of view). We are interested in defining the distance of finite automata because every strongly connected automaton accepts a base word with a length equal to the distance we just defined.

We define **Hamiltonian-like word** H_w of a strongly connected automata as the word accepted by automaton $A = (Q, \Sigma, \delta, q_0, q_f)$ that the walk represented by this word pass through all states Q of A, such that $\delta^*(q_0, H_w) = \bigcup_i \delta(q_i, x)$ for every $q_i \in Q$ and $H_w = xw$ for $x \in \Sigma$, and $w \in \Sigma^*$.

The **Eulerian-like word** E_w of a SCA is the word accepted by automaton $A = (Q, \Sigma, \delta, q_0, q_f)$ that the walk represented by this word pass through all states Q of A and the word contains every symbol $x \in \Sigma$, graphically all the edges in the automata.

Consider the transition graph of automaton $A = (Q, \sum, \delta, q_0, q_f)$ shown in Figure 19.

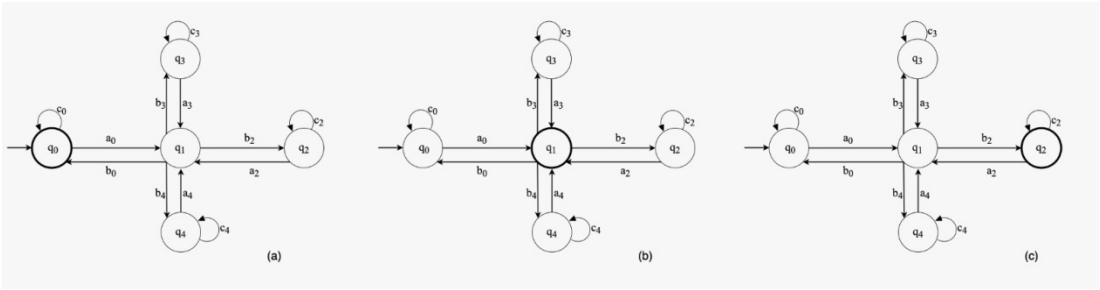

Figure 19: Automata with varying final state

The distance of (a), (b), (c) varies as we change the final states, and the graph of A for when F is $\{q_3\}$ or $\{q_4\}$ is similar to (c). According to our definition of distance, the distance for (a), (b), (c) is 0,1 and 2 respectively.

The reason for defining Hamiltonian-like word is because the shortest word accepted by some strongly connected automata might not pass through all states as seen in Figure 19(a) that accepts the empty word λ. One of the shortest Hamiltonian-like words accepted by the automata A in Figure 19 above is (a) $a_0b_3a_3b_2a_2b_4a_4b_0$ with length 8, (b) $a_0b_3a_3b_2a_2b_4a_4$ with length 7 and (c) $a_0b_3a_3b_4a_4b_2$ with length 6.

In this thesis, we explore different types of strongly connected automata and the regular language accepted by them. Even though Finite automata have been widely studied, the structural study of the transitional graph has not.

In many examples, we vary our final state F to be the $\{q\}$ for every $q \in Q$ as seen above. By varying the set of final states, we are able to consider the distance between the initial state q_0 and the final state q_f as we have defined as the distance d of the strongly connected automata S. We also relate this distance to the regular languages accepted by these automata and go further to investigate the number of Kleene stars associated with each regular expression.

Chapter 3

METHODOLOGY

3.1 Preamble

In this chapter, we explore five types of Strongly connected automata $(Q, \Sigma, \delta, q_0, F)$ such that for any pair of states $q_i, q_j \in Q$ there exits $w \in \Sigma^*$ such that $\delta(q_i, q_j) = w$.

We named these five types of automata literarily to understand the structure of such automata as:

1. Line directed,
2. Directed cycle,
3. Bidirected cycle,
4. Starred,
5. Floral.

For each type, we establish the structure of the automata and derive the regular expression r_i that generates the language these automata accept while varying the final state $F = \{q_f\}$ to be $\{q_i\}$ for $i = 0, 1, 2, .., n-1$ where n is the number of states.

Let $G(V, \Sigma)$ be a strongly connected automata graph. We associate the automaton $A = (Q, \Sigma, \delta, q_0, \{q_f\})$ such that $V = Q$, and the edges of G are given the transitions δ by their names, in this way every edge of the graph has it unique symbol in the alphabet and therefore our study really concentrate on the properties of the graph. More specifically, we defined our automata in the subsection of the next section.

We define the degree of a state as the number of edge incident to that state. The Indegree deg^+ as the number of inbound edges(arrow)s and Outdegree deg^- as the number of outbound edges(arrows).

3.2 Types of Strongly Connected Automata

3.2.1 Line Directed Automata

Line directed automata are characterized with each state q_i nested to it previous state q_{i-1} for $i = 1, 2, .., n$ and $n \geq 2$. Each state $q_i \in (Q - \{q_0, q_n\})$ has both indegree and outdegree to be 2, that is $deg^+ = deg^- = 2$, while q_0, q_n has indegree and outdegree to be 1.

We consider two forms of this Line directed automata, the simple line directed automata have define above and looped line directed automata with q_0, q_n both having a loop transition.

3.2.1.1 Simple Line Directed Automata

Let $A = (Q, \Sigma, \delta, q_0, F)$ be a Simple line directed Automaton with vertices $q_0, q_1, \dots, q_{n-1} \in Q$ having n states and $n \geq 2$. Every edge $q_i q_{i+1}$ has label a_i and oppositely oriented edge $q_{i+1} q_i$ with label b_{i+1}. Vertex q_i has degree $(2,2)$ for every $q_i \in (Q - \{q_0, q_{n-1}\})$, while q_0 and q_{n-1} has degree $(1,1)$ such that $0 \leq i \leq n-1$. $q_0, q_1, \dots, q_{n-1} \in Q$ forms a path represented by the sequence $q_0 q_1 \dots q_{n-1}$. Automaton A has a diameter of $n-1$. For $n \geq 3$, every vertex $q_i \in (Q - \{q_0, q_{n-1}\})$ is a cut vertex. Automaton A has no pancyclic vertex.

As mentioned, we are writing the regular expression for all possible choices of the final state. Let r_i denote the regular expression which describes the regular language when $F = \{q_i\}$.

Two states:

$$r_0 = (a_0 b_1)^*$$

$$r_1 = a_0(b_1a_0)^*$$

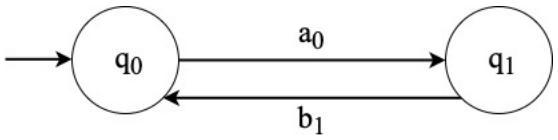

Figure 20: Two-state simple line directed automata

Three states:

$$r_0 = (a_0(a_1b_2)^*b_1)^*$$

$$r_1 = a_0(a_1b_2 + b_1a_0)^*$$

$$r_2 = (a_0b_1)^*(a_0a_1)\big((b_2a_1) + (b_2b_1)(a_0b_1)^*(a_0a_1)\big)^*$$

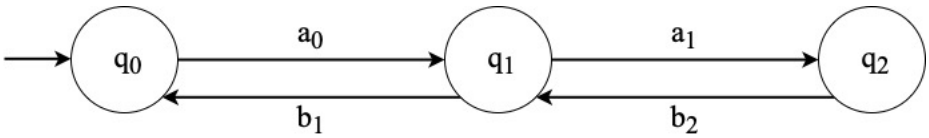

Figure 21: Three-state simple line directed automata

Four states:

$$r_0 = (a_0(a_1(a_2b_3)^*b_2)^*b_1)^*$$

$$r_1 = a_0(b_1a_0 + a_1(a_2b_3)^*b_2)^*$$

$$r_2 = (a_0b_1)^*(a_0a_1)\big((a_2b_3) + (b_2a_1) + (b_2b_1)(a_0b_1)^*(a_0a_1)\big)^*$$

$$r_3 = (a_0(a_1b_2)^*b_1)^*(a_0(a_1b_2)^*a_1a_2)\big((b_3a_2) + (b_3b_2(a_1b_2)^*a_1a_2)$$

$$+ (b_3b_2(a_1b_2)^*b_1)(a_0(a_1b_2)^*b_1)^*(a_0(a_1b_2)^*a_1a_2)\big)^*$$

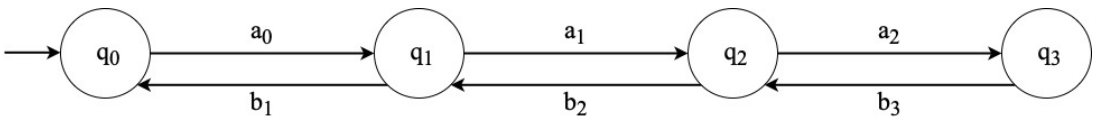

Figure 22: Four-state simple line directed automata

Five states:

$$r_0 = (a_0(a_1(a_2(a_3b_4)^*b_3)^*b_2)^*b_1)^*$$

$$r_1 = a_0(b_1a_0 + a_1(a_2(a_3b_4)^*b_3)^*b_2)^*$$

$$r_2 = (a_0b_1)^*(a_0a_1)\big((a_2(a_3b_4)^*b_3) + (b_2a_1) + (b_2b_1)(a_0b_1)^*(a_0a_1)\big)^*$$

$$r_3 = (a_0b_1 + a_0a_1(b_2a_1)^*b_2b_1)^*(a_0a_1a_2)\big((b_3(b_2a_1)^*a_2) + (a_3b_4) + (b_3b_2b_1)(a_0b_1 + a_0a_1(b_2a_1)^*b_2b_1)^*(a_0a_1a_2)\big)^*$$

$$r_4 = (a_0b_1 + a_0a_1(b_2a_1)^*b_2b_1 + a_0a_1a_2(b_3(b_2a_1)^*a_2)^*b_3b_2b_1)^*(a_0a_1a_2(b_3(b_2a_1)^*a_2)^*a_3)\big((b_4(b_3(b_2a_1)^*a_2)^*a_3) + (b_4(b_3(b_2a_1)^*a_2)^*b_3b_2b_1)(a_0b_1 + a_0a_1(b_2a_1)^*b_2b_1 + a_0a_1a_2(b_3(b_2a_1)^*a_2)^*b_3b_2b_1)^*(a_0a_1a_2(b_3(b_2a_1)^*a_2)^*a_3)\big)^*$$

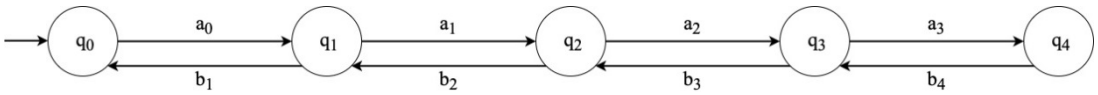

Figure 23: Five-state simple line directed automata

We will show the number of circuits contained in n −state simple line-directed automata in proposition 1 and the length of Hamiltonian-like words they accept in proposition 7.

3.2.1.2 Looped Line Direct Automata

Let $A = (Q, \Sigma, \delta, q_0, F)$ be a Simple line directed Automaton with n states and $n \geq 2$. We add two loop transitions $\delta(q_0, b_0) = q_0$, and $\delta(q_{n-1}, a_{n-1}) = q_{n-1}$. The new automaton is a Looped line direct automaton, and it has properties such as cut vertex, and diameter similar to a Simple line direct automaton of the same number of states. The language accepted by these automata can be defined as follows.

Two states:

$$r_0 = (b_0 + \mathrm{a}_0\mathrm{a}_1^*\mathrm{b}_1)^*$$

$$r_1 = b_0^* a_0 (a_1 + b_1 b_0^* a_0)^*$$

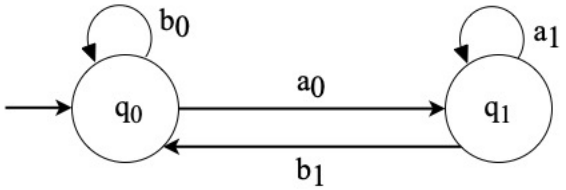

Figure 24: Two-state looped line direct automata

Three states:

$$r_0 = (b_0 + a_0 (a_1 a_2^* b_2)^* b_1)^*$$

$$r_1 = b_0^* a_0 (a_1 a_2^* b_2 + b_1 b_0^* a_0)^*$$

$$r_2 = (b_0 + a_0 b_1)^* (a_0 a_1)\big((a_2 + b_2 a_1) + b_2 b_1 (b_0 + a_0 b_1)^* (a_0 a_1)\big)^*$$

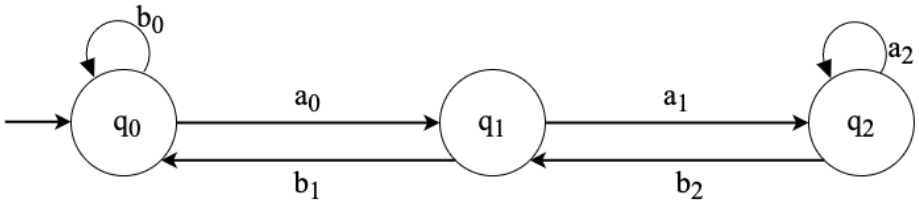

Figure 25: Three-state looped line direct automata

By reducing the graph using an extended transition graph, we have

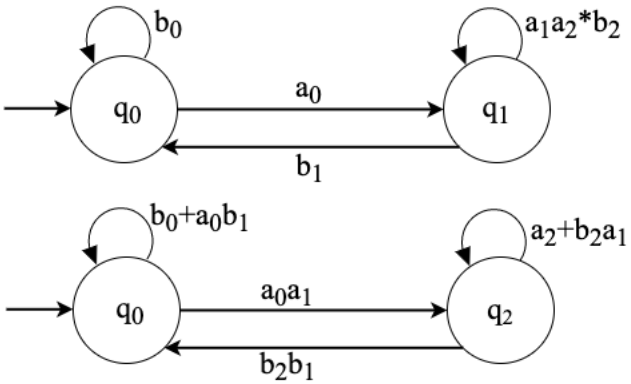

Figure 26: Two-state automata from three states looped line direct automata

Four states:

$$r_0 = (b_0 + a_0(a_1(a_2a_3^*b_3)^*b_2)^*b_1)^*$$

$$r_1 = b_0^*a_0(a_1(a_2a_3^*b_3)^*b_2 + b_1b_0^*a_0)^*$$

$$r_2 = b_0^*(a_0a_1)\big(b_2a_1 + a_2a_3^*b_3 + b_2b_1b_0^*(a_0a_1)\big)^*$$

$$r_3 = (b_0 + a_0(a_1b_2)^*b_1)^*(a_0(a_1b_2)^*a_1a_2)\big(a_3 + b_3a_2 + (b_3b_2)(a_1b_2)^*(a_1a_2)$$

$$+ (b_3b_2(a_1b_2)^*b_1)(b_0 + a_0(a_1b_2)^*b_1)^*(a_0(a_1b_2)^*a_1a_2)\big)^*$$

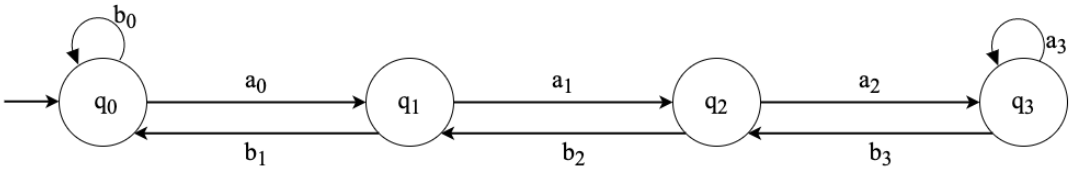

Figure 27: Four-state looped line direct automata

Reducing the graph to eliminate q_3 and q_2 separately.

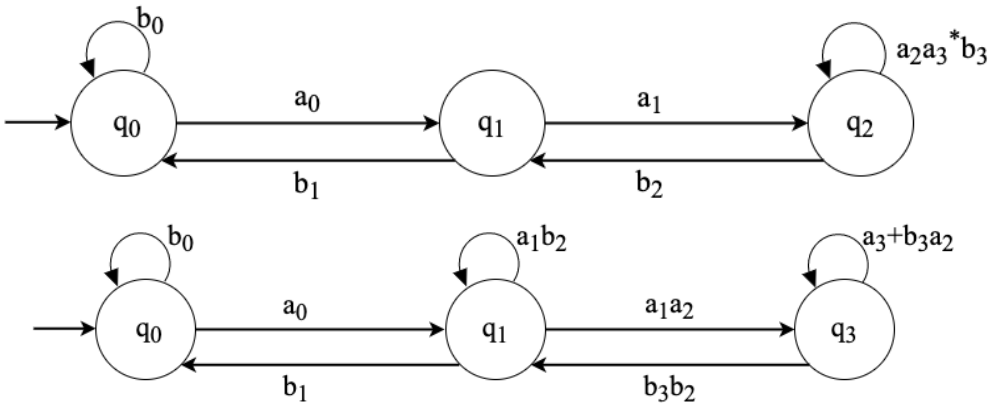

Figure 28: Three-state GTG of four states looped line direct automata

Further reduction of the above 3 states automata gives

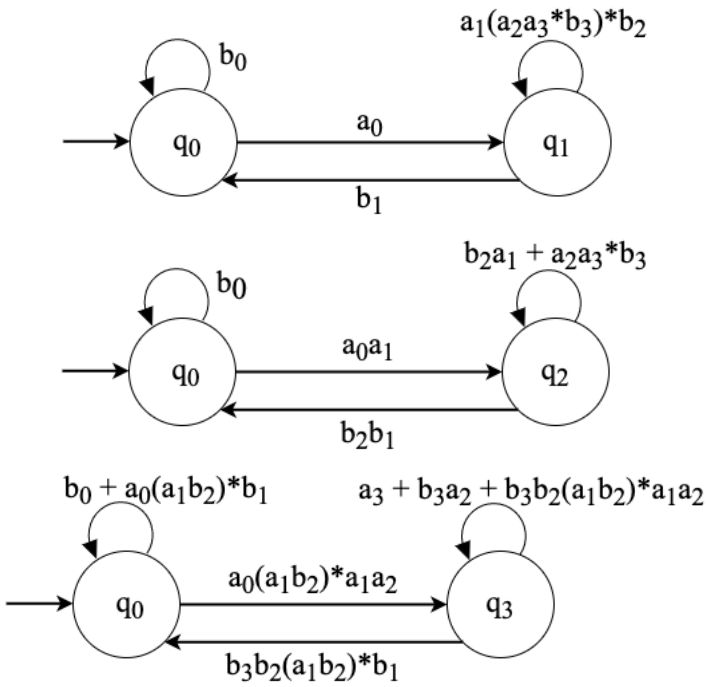

Figure 29: Two-state GTG of four states looped line direct automata

Looped line direct automata are similar to Simple line direct automata with the addition of two loops which means the number of circuit it contains is the same as that of the equal state simple line direct automaton with the addition of two, as it will be shown in proposition 2. From the above state reductions, we are able to show the maximum star height in the regular expression accepted by these automata in proposition 6. And finally, we give the length of the Hamiltonian-like word these automata accept as similar to that of Simple line direct in proposition 7.

3.2.2 Directed Cycle Automata

Directed cycle Automaton $A = (Q, \Sigma, \delta, q_0, F)$ is characterized with each state q_i incident to it previous state q_{i-1} for $0 \leq i \leq n-1$ and $n \geq 2$. Each state $q_i \in Q$ has both indegree and outdegree to be 1, that is $deg^+ = deg^- = 1$, and from graph theory called *Balanced directed cycle*. Every edge $q_i q_{i+1}$ has a label a_i and the last

edge $q_{n-1}q_0$ has a label a_{n-1}. $q_0, q_1, \ldots, q_{n-1} \in Q$ forms a path represented by the sequence $q_0q_1 \ldots q_{n-1}$ with the number of cycles containing any edge $Z(q_iq_j) = 1$. Directed cycle automata have no cut vertex and a diameter of $n - 1$.

These automata contain a cycle in which the first state as a vertex appears exactly twice (at the beginning and at the end) while the other states appear only once.

The language accepted by these automata can be defined by regular expression r_i as follows.

Two states: This automaton has the same structure as a Two-state Simple direct automaton with different edge labels.

$$r_0 = (a_0a_1)^*$$

$$r_1 = a_0(a_1a_0)^*$$

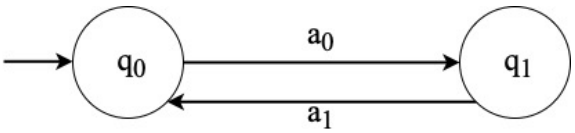

Figure 30: Two-state directed cycle automata

Three states:

$$r_0 = (a_0a_1a_2)^*$$

$$r_1 = a_0(a_1a_2a_0)^*$$

$$r_2 = a_0a_1(a_2a_0a_1)^*$$

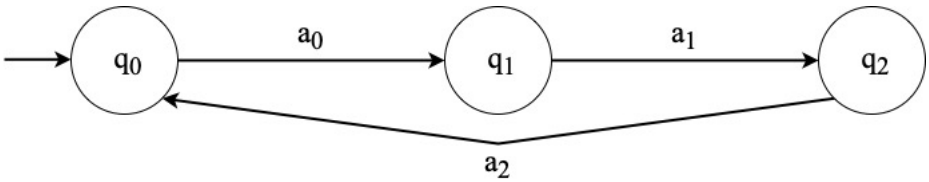

Figure 31: Three-state directed cycle automata

Four states:

$$r_0 = (a_0a_1a_2a_3)^*$$

$$r_1 = a_0(a_1a_2a_3a_0)^*$$

$$r_2 = a_0a_1(a_2a_3a_0a_1)^*$$

$$r_3 = a_0a_1a_2(a_3a_0a_1a_2)^*$$

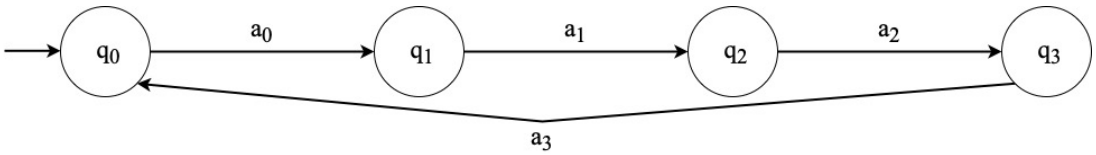

Figure 32: Four-state directed cycle automata

Five states:

$$r_0 = (a_0a_1a_2a_3a_4)^*$$

$$r_1 = a_0(a_1a_2a_3a_4a_0)^*$$

$$r_2 = a_0a_1(a_2a_3a_4a_0a_1)^*$$

$$r_3 = a_0a_1a_2(a_3a_4a_0a_1a_2)^*$$

$$r_4 = a_0a_1a_2a_3(a_4a_0a_1a_2a_3)^*$$

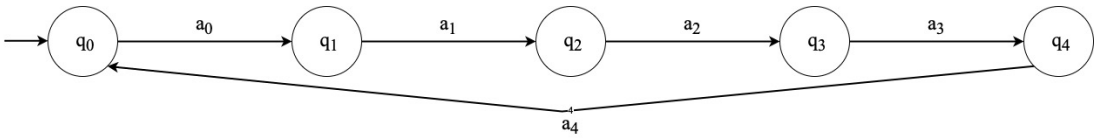

Figure 33: Five-state directed cycle automata

Like most strongly connected automata, Directed cycle automata have no cut vertex and it is trivial that the regular expression of the accepted language can be written with exactly one Kleene star. In proposition 8, we show the length of Hamiltonian-like words and Eulerian-like words accepted by these automata with the

index of the final state as a dependent variable or the distance of the automata as we have defined in the previous chapter.

3.2.3 Bidirected Cycle

Let $A = (Q, \Sigma, \delta, q_0, F)$ be a Bidirected cycle Automaton with n states and $n \geq 2$. Every edge $q_i q_{i+1}$ has a label a_{i+1} and an oppositely oriented edge $q_{i+1} q_i$ with label b_{i+1}. The first and last vertex are both incident to edge $q_0 q_{n-1}$ with label a_0 and oppositely oriented edge $q_{n-1} q_0$ with label b_0. Each vertex $q_i \in Q$ has both indegree and outdegree to be 2, that is $deg^+ = deg^- = 2$. Automata A has a diameter of $n - 2$ and no cut vertex. The language accepted by these automata can be defined as follows. Two-state Bidirected cycle automaton has the same structure as a Simple line direct automaton with two states, hence the language accepted can be represented by the same regular expression.

Three states:

$$r_0 = (a_0 b_0)^* + (a_0 b_0)^* \left((a_0 + a_1 a_2)(b_2 a_2)^* (b_0 + b_2 b_1)\right)^*$$

$$r_1 = (a_0 b_0)^* (a_1 + a_0 b_2)(a_2 b_2)^* \left((b_1 + a_2 b_0)(a_0 b_0)^* (a_1 + a_0 b_2)(a_2 b_2)^*\right)^*$$

$$r_2 = (a_1 b_1)^* (a_0 + a_1 a_2)(b_2 a_2)^* \left((b_0 + b_2 b_1)(a_1 b_1)^* (a_0 + a_1 a_2)(b_2 a_2)^*\right)^*$$

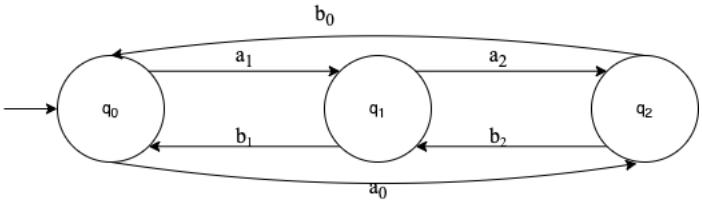

Figure 34: Three-state bidirected cycle automata

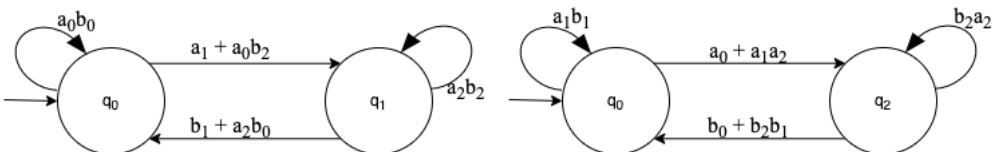

Figure 35: Two-state GTG of bidirected cycle automata of three states

Four states:

$$r_0 = (a_0b_0 + a_1b_1)^*$$
$$+ (a_0b_0 + a_1b_1)^*\big((a_1a_2 + a_0b_3)(a_3b_3 + b_2a_2)^*(b_2b_1 + a_3b_0)\big)^*$$

$$r_1 = (a_0(b_3a_3)^*b_0)^*(a_1$$
$$+ a_0(b_3a_3)^*b_3b_2)(a_2b_2$$
$$+ a_2a_3(b_3a_3)^*b_3b_2)^*\big((b_1 + a_2a_3(b_3a_3)^*b_0)(a_0(b_3a_3)^*b_0)^*(a_1$$
$$+ a_0(b_3a_3)^*b_3b_2)(a_2b_2 + a_2a_3(b_3a_3)^*b_3b_2)^*\big)^*$$

$$r_2 = (a_0b_0 + a_1b_1)^*(a_1a_2$$
$$+ a_0b_3)(a_3b_3$$
$$+ b_2a_2)^*\big((a_3b_0 + b_2b_1)(a_0b_0 + a_1b_1)^*(a_1a_2$$
$$+ a_0b_3)(a_3b_3 + b_2a_2)^*\big)^*$$

$$r_3 = (a_1(a_2b_2)^*b_1)^*(a_0$$
$$+ a_1(a_2b_2)^*a_2a_3)(b_3a_3$$
$$+ b_3b_2(a_2b_2)^*a_2a_3)^*\big((b_0 + b_3b_2(a_2b_2)^*b_1)(a_1(a_2b_2)^*b_1)^*(a_0$$
$$+ a_1(a_2b_2)^*a_2a_3)(b_3a_3 + b_3b_2(a_2b_2)^*a_2a_3)^*\big)^*$$

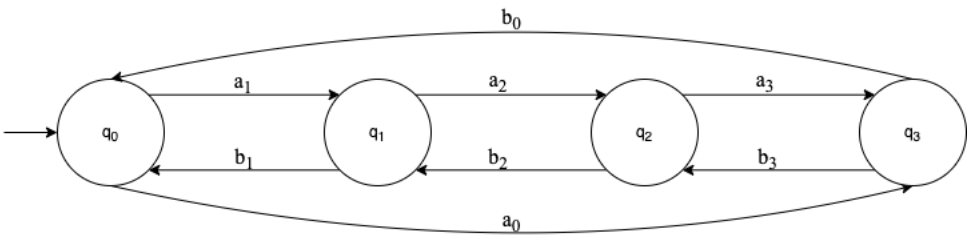

Figure 36: Four-state bidirected cycle automata

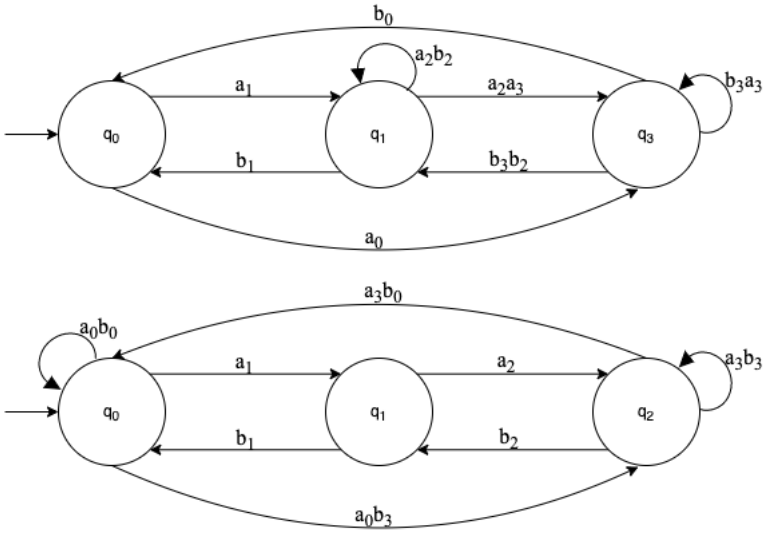

Figure 37: Three-state GTG of four states bidirected cycle automata

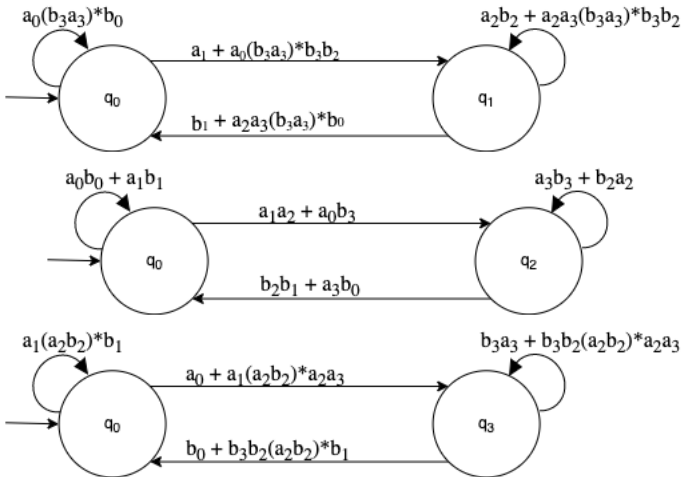

Figure 38: Two-state GTG of four states bidirected cycle automata

Five states:

$$r_0 = (a_0b_0 + a_1b_1 + a_0b_4(a_4b_4)^*a_4b_0)^*$$
$$+ (a_0b_0 + a_1b_1$$
$$+ a_0b_4(a_4b_4)^*a_4b_0)^*\big((a_1a_2$$
$$+ a_0b_4(a_4b_4)^*b_3)(a_3(a_4b_4)^*b_3 + b_2a_2)^*(b_2b_1 + a_3(a_4b_4)^*a_4b_0)\big)^*$$

$$r_1 = (a_0b_0 + a_0b_4(a_4b_4 + b_3a_3)^*a_4b_0)^*(a_1 + a_0b_4(a_4b_4 + b_3a_3)^*b_3b_2)(a_2b_2 + a_2a_3(a_4b_4 + b_3a_3)^*b_3b_2)^*\big((b_1 + a_2a_3(a_4b_4 + b_3a_3)^*a_4b_0)(a_0b_0 + a_0b_4(a_4b_4 + b_3a_3)^*a_4b_0)^*(a_1 + a_0b_4(a_4b_4 + b_3a_3)^*b_3b_2)(a_2b_2 + a_2a_3(a_4b_4 + b_3a_3)^*b_3b_2)^*\big)^*$$

$$r_2 = (a_0b_0 + a_1b_1 + a_0b_4(a_4b_4)^*a_4b_0)^*(a_1a_2 + a_0b_4(a_4b_4)^*b_3)(a_3(a_4b_4)^*b_3 + b_2a_2)^*\big((a_3(a_4b_4)^*a_4b_0 + b_2b_1)(a_0b_0 + a_1b_1 + a_0b_4(a_4b_4)^*a_4b_0)^*(a_1a_2 + a_0b_4(a_4b_4)^*b_3)(a_3(a_4b_4)^*b_3 + b_2a_2)^*\big)^*$$

$$r_3 = (a_0b_0 + a_1(a_2b_2)^*b_1)^*(a_0b_4 + a_1(a_2b_2)^*a_2a_3)(a_4b_4 + b_3a_3 + b_3b_2(a_2b_2)^*a_2a_3)^*\big((a_4b_0 + b_3b_2(a_2b_2)^*b_1)(a_0b_0 + a_1(a_2b_2)^*b_1)^*(a_0b_4 + a_1(a_2b_2)^*a_2a_3)(a_4b_4 + b_3a_3 + b_3b_2(a_2b_2)^*a_2a_3)^*\big)^*$$

$$r_4 = (a_1b_1 + a_1a_2(b_2a_2 + a_3b_3)^*b_2b_1)^*(a_0 + a_1a_2(b_2a_2 + a_3b_3)^*a_3a_4)(b_4a_4 + b_4b_3(b_2a_2 + a_3b_3)^*a_3a_4)^*\big((b_0 + b_4b_3(b_2a_2 + a_3b_3)^*b_2b_1)(a_1b_1 + a_1a_2(b_2a_2 + a_3b_3)^*b_2b_1)^*(a_0 + a_1a_2(b_2a_2 + a_3b_3)^*a_3a_4)(b_4a_4 + b_4b_3(b_2a_2 + a_3b_3)^*a_3a_4)^*\big)^*$$

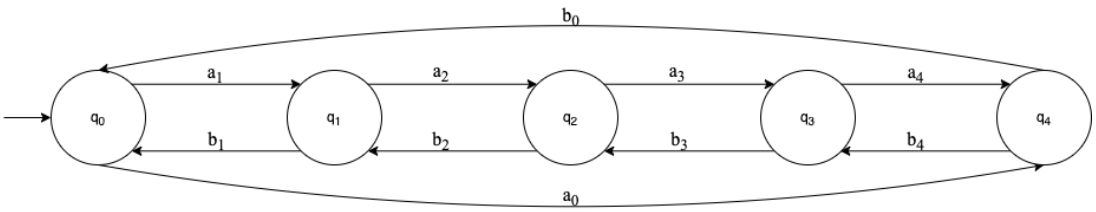

Figure 39: Five-state bidirected cycle automata

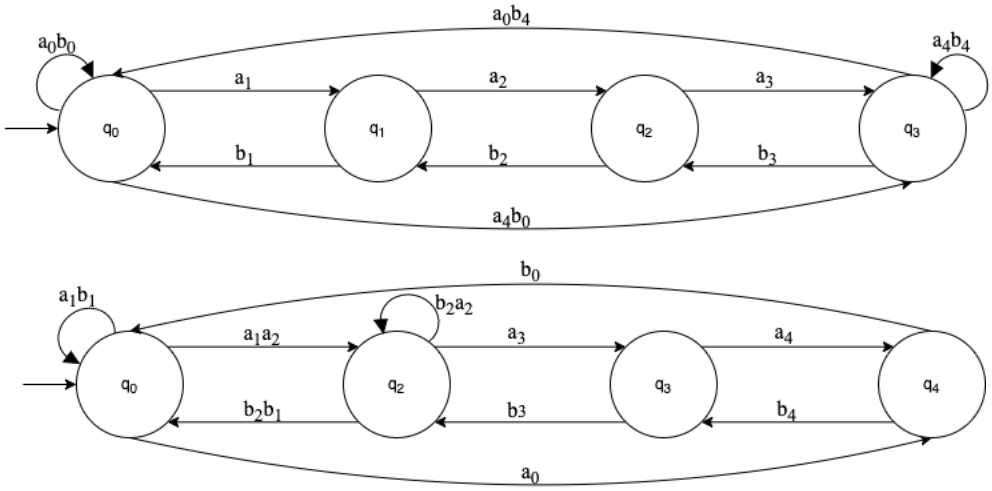

Figure 40: Four-state GTG of five states bidirected cycle automata

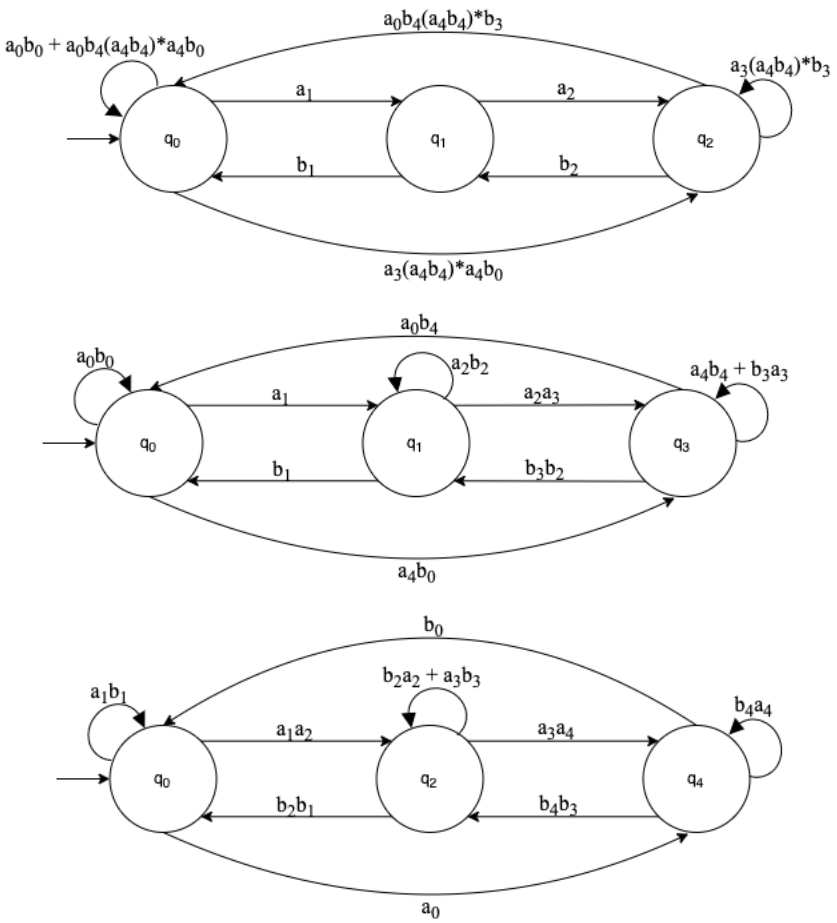

Figure 41: Three-state GTG of five states bidirected cycle automata

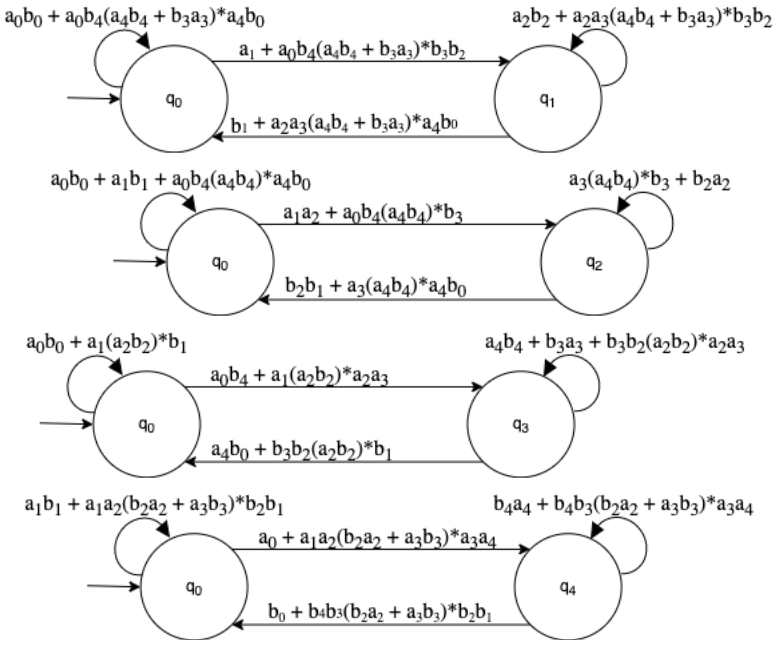

Figure 42: Two-state GTG of five states bidirected cycle automata

For n −states Bidirected cycle automata, we show the number of circuits and cycles it contains in proposition 3 and its regular expression minimum and maximum star height in proposition 4 and 6, respectively as being similar to the same number of state loop line direct automata. Finally, on these automata, we show the length of the shortest Hamiltonian-like and Eulerian-like words for odd and even numbers of states in proposition 9.

3.2.4 Starred

Starred automaton $A = (Q, \Sigma, \delta, q_0, F)$ is characterized as a complete bipartite graph $\mathcal{K}_{1,n-1}$ for n states in Q and $n \geq 2$. It is an automaton with one internal state as node and $n - 1$ leaves. Each state $q_i \in (Q - \{q_0\})$ has both indegree and outdegree to be 1, that is $deg^+ = deg^- = 1$ while q_0 has $deg^+ = deg^- = n - 1$. Every edge $q_0 q_i$ has a label a_i while the oppositely oriented edge $q_i q_0$ has a label b_i. It has a diameter of 2 and a single cut vertex q_0 for $n \geq 3$.

The language accepted by these automata can be defined as follows.

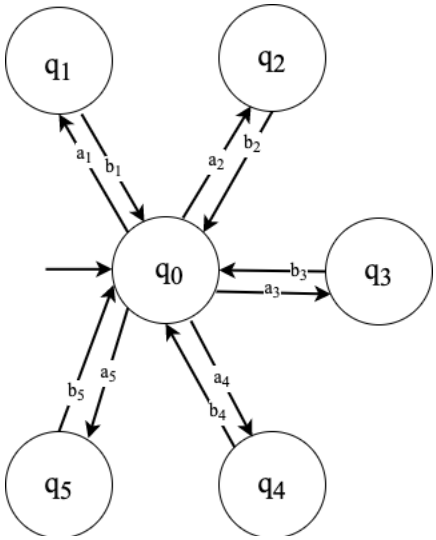

Figure 43: Six-state starred automata

Two-state Starred automaton is structurally similar to Simple line direct or Directed cycle automata of two states with different edge labeling.

$$r_0 = (a_1 b_1)^*$$

$$r_1 = a_1 (b_1 a_1)^*$$

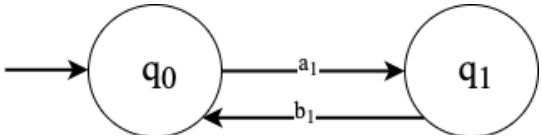

Figure 44: Two-state starred automata

Three-state Starred automaton is similar in structure to a Simple line direct of three states with the initial state in the middle.

$$r_0 = (a_1 b_1 + a_2 b_2)^*$$

$$r_1 = (a_1 b_1 + a_2 b_2)^* a_1$$

$$r_2 = (a_1 b_1 + a_2 b_2)^* a_2$$

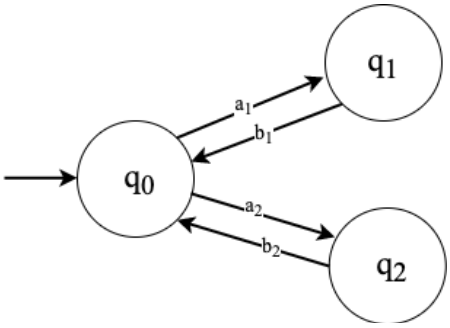

Figure 45: Three-state starred automata

Four states:

$$r_0 = (a_1 b_1 + a_2 b_2 + a_3 b_3)^*$$

$$r_1 = (a_1 b_1 + a_2 b_2 + a_3 b_3)^* a_1$$

$$r_2 = (a_1 b_1 + a_2 b_2 + a_3 b_3)^* a_2$$

$$r_3 = (a_1 b_1 + a_2 b_2 + a_3 b_3)^* a_3$$

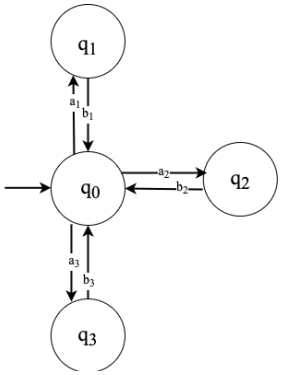

Figure 46: Four-state starred automata

Five states:

$$r_0 = (a_1b_1 + a_2b_2 + a_3b_3 + a_4b_4)^*$$

$$r_1 = (a_1b_1 + a_2b_2 + a_3b_3 + a_4b_4)^*a_1$$

$$r_2 = (a_1b_1 + a_2b_2 + a_3b_3 + a_4b_4)^*a_2$$

$$r_3 = (a_1b_1 + a_2b_2 + a_3b_3 + a_4b_4)^*a_3$$

$$r_4 = (a_1b_1 + a_2b_2 + a_3b_3 + a_4b_4)^*a_4$$

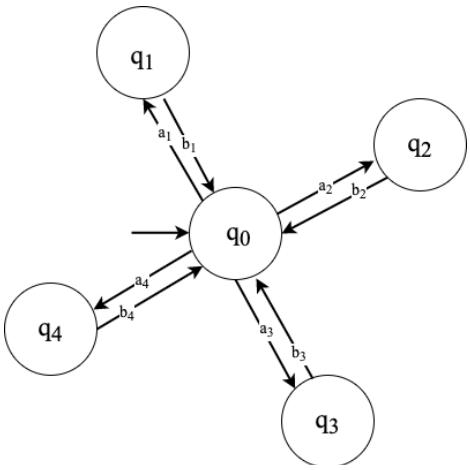

Figure 47: Five-state starred automata

3.2.5 Floral

The Floral automata are just like the Starred automata with a centered state q_0 acting as a node to other states in a more complex connection. Let $A = (Q, \Sigma, \delta, q_0, F)$ be a Floral automaton, every state $q_i \in (Q - \{q_0\})$ is incident to q_0. We define the smallest structure to be a Petal consisting of four states, as shown in Figure 48. For i number of petals in A, every edge q_0q_i has a label a_i and for $j \in \{1,2\}$, the edge q_iq_{ij} has a label a_{ij}, while edge $q_{ij}q_0$ has a label b_{ij}. It has a diameter of 4 and a single cut vertex q_0 for $n \geq 4$ and $n \; mod \; 3 = 1$. The language accepted by these automata can be defined as follows.

One petal:

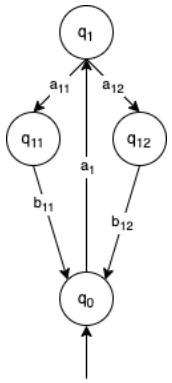

Figure 48: One-petal floral automata with four states

$$r_0 = (a_1a_{11}b_{11} + a_1a_{12}b_{12})^*$$

$$r_1 = r_0a_1 = (a_1a_{11}b_{11} + a_1a_{12}b_{12})^*a_1$$

$$r_{11} = r_0a_1a_{11} = (a_1a_{11}b_{11} + a_1a_{12}b_{12})^*a_1a_{11}$$

$$r_{12} = r_0a_1a_{12} = (a_1a_{11}b_{11} + a_1a_{12}b_{12})^*a_1a_{12}$$

Two petals:

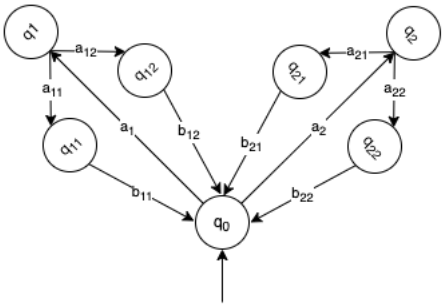

Figure 49: Two-petal floral automata with seven states

$$r_0 = (a_1a_{11}b_{11} + a_1a_{12}b_{12} + a_2a_{21}b_{21} + a_2a_{22}b_{22})^*$$

$$r_1 = r_0a_1 = (a_1a_{11}b_{11} + a_1a_{12}b_{12} + a_2a_{21}b_{21} + a_2a_{22}b_{22})^*a_1$$

$$r_2 = r_0a_2 = (a_1a_{11}b_{11} + a_1a_{12}b_{12} + a_2a_{21}b_{21} + a_2a_{22}b_{22})^*a_2$$

$$r_{11} = r_0a_1a_{11} = (a_1a_{11}b_{11} + a_1a_{12}b_{12} + a_2a_{21}b_{21} + a_2a_{22}b_{22})^*a_1a_{11}$$

$$r_{12} = r_0a_1a_{12} = (a_1a_{11}b_{11} + a_1a_{12}b_{12} + a_2a_{21}b_{21} + a_2a_{22}b_{22})^*a_1a_{12}$$

$$r_{21} = r_0a_2a_{21} = (a_1a_{11}b_{11} + a_1a_{12}b_{12} + a_2a_{21}b_{21} + a_2a_{22}b_{22})^*a_2a_{21}$$

$$r_{22} = r_0a_2a_{22} = (a_1a_{11}b_{11} + a_1a_{12}b_{12} + a_2a_{21}b_{21} + a_2a_{22}b_{22})^*a_2a_{22}$$

Three petals:

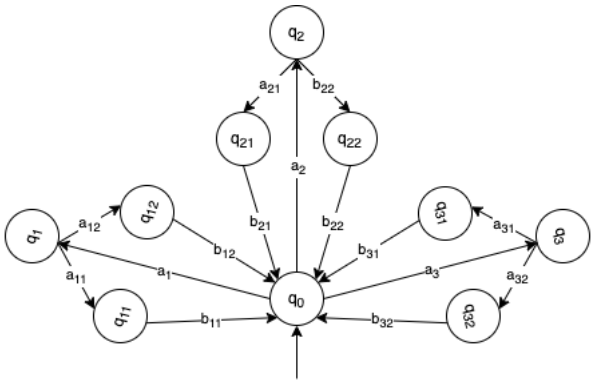

Figure 50: Three-petal floral automata with ten states

$$r_0 = (a_1a_{11}b_{11} + a_1a_{12}b_{12} + a_2a_{21}b_{21} + a_2a_{22}b_{22} + a_3a_{31}b_{31} + a_3a_{32}b_{32})^*$$

$$r_1 = r_0 a_1 = (a_1 a_{11} b_{11} + a_1 a_{12} b_{12} + a_2 a_{21} b_{21} + a_2 a_{22} b_{22} + a_3 a_{31} b_{31} + a_3 a_{32} b_{32})^* a_1$$

$$r_2 = r_0 a_2 = (a_1 a_{11} b_{11} + a_1 a_{12} b_{12} + a_2 a_{21} b_{21} + a_2 a_{22} b_{22} + a_3 a_{31} b_{31} + a_3 a_{32} b_{32})^* a_2$$

$$r_3 = r_0 a_3 = (a_1 a_{11} b_{11} + a_1 a_{12} b_{12} + a_2 a_{21} b_{21} + a_2 a_{22} b_{22} + a_3 a_{31} b_{31} + a_3 a_{32} b_{32})^* a_3$$

$$r_{11} = r_0 a_1 a_{11} = (a_1 a_{11} b_{11} + a_1 a_{12} b_{12} + a_2 a_{21} b_{21} + a_2 a_{22} b_{22} + a_3 a_{31} b_{31} + a_3 a_{32} b_{32})^* a_1 a_{11}$$

$$r_{12} = r_0 a_1 a_{12} = (a_1 a_{11} b_{11} + a_1 a_{12} b_{12} + a_2 a_{21} b_{21} + a_2 a_{22} b_{22} + a_3 a_{31} b_{31} + a_3 a_{32} b_{32})^* a_1 a_{12}$$

$$r_{21} = r_0 a_2 a_{21} = (a_1 a_{11} b_{11} + a_1 a_{12} b_{12} + a_2 a_{21} b_{21} + a_2 a_{22} b_{22} + a_3 a_{31} b_{31} + a_3 a_{32} b_{32})^* a_2 a_{21}$$

$$r_{22} = r_0 a_2 a_{22} = (a_1 a_{11} b_{11} + a_1 a_{12} b_{12} + a_2 a_{21} b_{21} + a_2 a_{22} b_{22} + a_3 a_{31} b_{31} + a_3 a_{32} b_{32})^* a_2 a_{22}$$

$$r_{31} = r_0 a_3 a_{31} = (a_1 a_{11} b_{11} + a_1 a_{12} b_{12} + a_2 a_{21} b_{21} + a_2 a_{22} b_{22} + a_3 a_{31} b_{31} + a_3 a_{32} b_{32})^* a_3 a_{31}$$

$$r_{32} = r_0 a_3 a_{32} = (a_1 a_{11} b_{11} + a_1 a_{12} b_{12} + a_2 a_{21} b_{21} + a_2 a_{22} b_{22} + a_3 a_{31} b_{31} + a_3 a_{32} b_{32})^* a_3 a_{32}$$

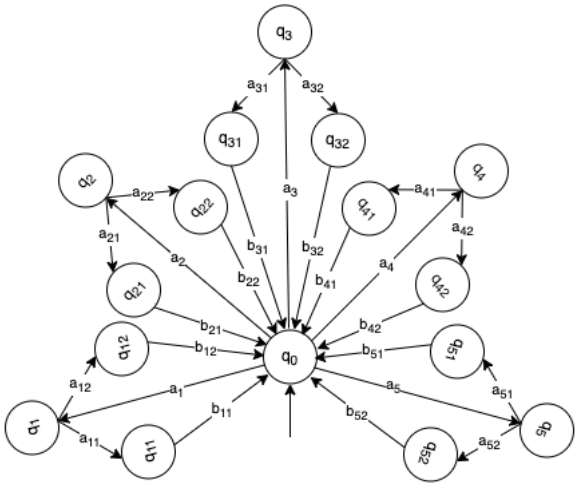

Figure 51: Five-petal floral automata with sixteen states

Both starred and floral automata can be said to have a graph G with a pancyclic vertex $q_0 \in Q$ with a finite number of p-subgraph H incident to q_0. A p-subgraph H of graph G has all the edges incident to its set of vertices in its edge set. While p-subgraph H of starred automata has two vertices, that of floral has four vertices. We will also show in Proposition 10 and 11, the length of Hamiltonian-like and Eulerian-like words accepted by these two automata types.

Chapter 4

ANALYSIS

In this chapter, we give and prove some theoretical results about the Strongly connected automata classes we consider. We start with general bounds.

The length of the Hamiltonian-like word H_w and Eulerian-like word E_w of a strongly connected automaton $A = (Q, \Sigma, \delta, q_0, q_f)$ is at least the number of states, and the distance d of the automaton is at most the number of states, $d \leq |Q| \leq |H_w|$.

These can be seen as follow. The part of the argument $d \leq |Q|$, is trivial. Since d is the length of the first word accepted by automata A**,** the word either passes through all states or fewer states. The second part $|Q| \leq |H_w|$ is a direct interpretation of the definition of Hamiltonian-like word as the word that passes through all states of the automata, hence cannot be less than $|Q|$.

It is important to know that a walk which is an alternating sequence of vertices and edges, in our case can be written as a sequence of vertices or edges as all edges has a unique label, and the edge incident to two vertices is also unique in orientation.

Proposition 1

If $A = (Q, \Sigma, \delta, q_0, q_f)$ is a Simple line directed automata with $|Q| = n$, then A has $\frac{n(n-1)}{2}$ circuits.

Proof: Let $G(Q, \Sigma)$ be a graph of A with n vertices and Z number of circuits. The number of circuits an edge $e \in \Sigma$ belongs to, is greater than 1, $Z(e) \geq 1$. All circuits of G are symmetric with length $2k$ for $1 \leq k \leq n-1$. For n states, G has $(n-k)$

circuits of length $2k$ for $1 \leq k \leq n-1$. As we vary the value of k, $Z = \sum_{k=1}^{n-1} k$ can be expressed as $1 + 2 + \cdots + (n-1)$. We have $2Z = n(n-1)$ and hence the prove. ■

Proposition 2

If $A = (Q, \Sigma, \delta, q_0, q_f)$ is a Loop line directed automata with $|Q| = n$, then A has $\frac{n(n-1)}{2} + 2$ circuits.

Proof: Let $G(Q, \Sigma)$ be Simple line directed graph with n vertices and Z number of circuits, and $G'(Q, \Sigma \cup \{b_0, a_{n-1}\})$ be Loop line directed graph with n vertices and Z' number of circuits. It is trivial that G is e-subgraph of G' and $Z' \geq Z$. All the circuits in G are also circuits in G' with the addition of two circuits b_0, a_{n-1} which denote the self-loops $q_0q_0, q_{n-1}q_{n-1}$ respectively. Hence the proof, following from proposition 1. ■

Proposition 3

If $A = (Q, \Sigma, \delta, q_0, q_f)$ is a Bidirected cycle automata with $|Q| = n$, then A has $n^2 + 2$ circuits for $n \geq 3$, and $n + 2$ cycles.

Proof: Let $G(Q, \Sigma)$ be the graph of Bidirected cycle automata with n vertices and Z number of circuits. The uniqueness of a circuit depends on the length of the circuit and the order of edges in the circuit.

There are always two n-circuit of the form $a_1a_2 \ldots a_{n-1}b_0$, $a_0b_{n-1} \ldots b_2b_1$, additionally, for each state of the n states, there are n circuit that starts with the state and of varying length $2k$ for $1 \leq k \leq n$. Thus, we have $n^2 + 2$ circuits for $n \geq 3$. Alternatively, if n is odd, for $1 \leq k \leq n$, A has n $2k$-circuit for each value of k, and two n-circuit then $Z_n = n^2 + 2$.

If n is even, for $1 \leq k \leq n$, A has $(n-1)$ $2k$-circuit for each value of k, and $(n+2)$ n-circuit then $Z_n = n(n-1) + (n+2) = n^2 + 2$.

For the number of cycles, there are always n 2-cycle in A and two n-cycle (cycle with length n) of the form $q_0 q_1 \dots q_{n-1} q_0$ and $q_0 q_{n-1} \dots q_1 q_0$, which sums the number of cycles to $n + 2$. ■

A two state automata graph that is Loop line directed is called generic two-state automaton (John E. Hopcroft, Rajeev Motwani, & Jeffrey D. Ullman, 2006) and we represent it as G^* as shown in Figure 12. The automata graph of Simple line direct, Looped line direct and Bidirected cycle can be reduced to G^* using Generalized Transition Graph (GTG) method.

Proposition 4

The regular expression accepted by Loop line directed and Bidirected cycle automata has a minimum star height of two.

Proof: Loop line directed and Bidirected cycle automata can be reduced using the GTG elimination method to a generic two-state automaton, as shown in Figure 12. The regular expression that is accepted by this automata graph r_0 and r_1 for when the final state is q_0 and q_1, respectively, has a star height of two. From the expression, the star height of both Loop line directed and Bidirected cycle automata is a minimum of two and hence the proof. ■

Proposition 5

The graph of Directed and Bidirected cycle automata has no cut vertex.

Proof: Bidirected cycle automata has two Hamiltonian cycle and Directed cycle has one Hamiltonian cycle as it only cycle, hence removing one vertex will not split the component of the two automata graph. ■

Proposition 6

The regular expression describing the language accepted by Loop line directed or Bidirected cycle automata $A = (Q, \Sigma, \delta, q_0, q_f)$ with $|Q| = n$, can be written with the maximum star height of $\left\lfloor \frac{n-1}{2} \right\rfloor + 2$.

Proof: For Loop line directed or Bidirected cycle graph with n vertices, let k be the index representing the final state of the automata for $2 \leq k \leq n-1$. In deriving the regular expression that these automata accept we will eliminate all state except for the initial state represented as q_0 and the final state which is q_k. Eliminating a 2-cycle results in the incident vertex having a loop while eliminating a 4-cycle results in the incident vertex having a Kleene star in the regular expression associated with it edges. $\left\lfloor \frac{n-k+1}{2} \right\rfloor$ is the number of nested star that edges incident to q_k can have by eliminating vertex q_{k+1} to q_{n-1}. $\left\lfloor \frac{k}{2} \right\rfloor$ is the number of nested stars that edges incident to q_0 can have by eliminating vertex q_1 to q_{k-1}. The resulting automata graph is a generic two-state automata that accepts regular expression with star height of two independent of which of the two state is the final state. Hence $\left\lfloor \frac{n-k-1}{2} \right\rfloor + \left\lfloor \frac{k}{2} \right\rfloor + 2$ represent the star height of Loop line directed or Bidirected cycle. ■

Proposition 7

Line directed automata $A = (Q, \Sigma, \delta, q_0, q_f)$ with $|Q| = n$ and $q_f = q_k$ accepts Hamiltonian-like word of length $2n - k - 2$ and Eulerian-like word of length $2n + k - 2$ for Simple line directed and $2n + k$ for Loop line directed automata.

Proof: Let $G(Q, \Sigma)$ be graph of Line directed automaton with n vertices, then every Hamiltonian-like word will always contain the walk $q_0 q_1 \dots q_{n-1}$ of length $n - 1$ for $q_0, q_1, \dots, q_{n-1} \in Q$. If vertex q_k is the vertex equivalent to the final state, then there

is $n-k-1$ steps from the last state to the accepting state. Then the Hamiltonian-like word will have a length of $(n-1)+(n-k-1)$ which is $2n-k-2$.

For Eulerian-like word, it takes $2n-2$ steps form the initial state back to initial state to cover all edges and an additional k steps to reach the final state which adds up to $2n-2+k$ and for Loop line directed we have additional two steps for the two loops. ■

Proposition 8

Directed cycle automata $A=(Q,\Sigma,\delta,q_0,q_f)$ with $|Q|=n$ and $q_f=q_k$, accept Hamiltonian-like word and Eulerian-like word of length $(n-1)$ when $k=n-1$ and $(n+k)$ for other values of k.

Proof: Let $G(Q,\Sigma)$ be Directed cycle graph with n vertices, then every Hamiltonian-like word will contain the walk $q_0q_1\dots q_{n-1}$ of length $n-1$ for $q_0,q_1,\dots,q_{n-1}\in Q$. When k is the index of the last state q_{n-1}, then the length of the Hamiltonian-like word is $n-1$. For any other states, the walk will return back to q_0 with n steps and for the k steps to reach q_k. Hence, the length of the Hamiltonian-like word will be $n+k$. ■

Proposition 9

Bidirected cycle automata $A=(Q,\Sigma,\delta,q_0,q_f)$ with $|Q|=n$ and index of $q_f=q_k$, accepts Eulerian-like word of $2n+k$ and Hamiltonian-like word of length $|H_w|_n^k$ which is n when $k=0$ and

When n is odd, median $m_1=\frac{n+1}{2}$ and $m_2=\frac{n+1}{2}+1$,

$$|H_w|_n^k=\begin{cases} n+k, & 1\le k\le m_1 \\ 2n-k-2, & m_2\le k\le n-1 \end{cases}$$

When n is even, median $m=\frac{n+2}{2}$.

$$|H_w|_n^k=\begin{cases} n+k, & 1\le k\le m \\ 2n-k-2, & m<k\le n-1 \end{cases}$$

Proposition 10

Starred automata $A = (Q, \Sigma, \delta, q_0, q_f)$ with $|Q| = n + 1$ and $q_f = q_k$, with the pancyclic vertex having index 0 has Hamiltonian-like word and Eulerian-like word of length $2n$ when $k = 0$. For other values of k, Hamiltonian-like word of length $2n - 1$ and Eulerian-like word of length $2n + 1$.

Proof: The statement is trivial from the structure of starred automata. For example, from Figure 43 of Six-state starred automata, $|Q| = 6$ which means $n = 5$. When $q_f = q_0$, $H_w = a_1b_1a_2b_2a_3b_3a_4b_4a_5b_5$ with $|H_w|_5^0 = 10 = 2n$, and for other values of k, for example $q_f = q_5$, $H_w = a_1b_1a_2b_2a_3b_3a_4b_4a_5$ with $|H_w|_5^5 = 9 = 2n - 1$. ■

Proposition 11

Floral automata graph with subgraphs H_n having three vertices each all incident to the pancyclic vertex 0 where n is the number of subgraphs, with degree order of $(1,1)$, $(1,2)$ and $(1,1)$ for the vertex index of $n1$, n and $n2$ respectively. The length of the Hamiltonian-like word $|H_w|$ and Eulerian-like word $|E_w|$ accepted by these automata when $k = 0$ is $6n$, $6n + 1$ when $k = n$. When k is $n1$ or $n2$, $|H_w|$ is $6n - 1$, and $|E_w|$ is $6n + 2$.

Proof: Using induction, the base case is Figure 48, One-petal floral automata with $n = 1$. $|H_w|_1^0 = 6$ has $H_w = a_1a_{11}b_{11}a_1a_{12}b_{12}$, $|H_w|_1^1 = 7$ has $H_w = a_1a_{11}b_{11}a_1a_{12}b_{12}a_1$, $|H_w|_1^{11} = 5$ has $H_w = a_1a_{12}b_{12}a_1a_{11}$, $|H_w|_1^{12} = 5$ has $H_w = a_1a_{11}b_{11}a_1a_{12}$. The Hamiltonian-like words cannot be shorter than these as the walks are unique to the structure.

If we assume true for n. Then, $|H_w|_n^0 = 6n$, $|H_w|_n^k = 6n + 1$, $|H_w|_n^{k1} = |H_w|_n^{k2} = 6n - 1$.

For $n + 1$ petals, $|H_w|_{n+1}^0 = |H_w|_n^0 + |H_w|_1^0 = 6n + 6 = 6(n + 1)$.

$|H_w|_{n+1}^k = |H_w|_n^0 + |H_w|_1^1 = 6n + 7 = 6(n + 1) + 1$.

$|H_w|_{n+1}^{k1,k2} = |H_w|_n^0 + |H_w|_1^{11,12} = 6n + 5 = 6(n+1) - 1.$

The Eulerian-like word accepted by the automata is only different from the Hamiltonian-like word when k is $n1$ or $n2$ because Eulerian-like word has to traverse all edges, $6n$, before reaching $q_f = q_k$ in two steps, hence the plus two. ■

Chapter 5

CONCLUSION

Studying cycles in graph theory is of significant importance, offering valuable insights into the structure and behavior of interconnected systems like the five structures of some strongly connected automata that we have analyzed. Cycles and circuits play a vital role in understanding connectivity, network properties, and developing efficient algorithms, which is crucial in fields like computer networks, transportation systems, and social networks, where the flow of information and resources relies on connectivity. Our study of cycles helps unravel some algebraic properties of these automata, with connections to linear algebra and algebraic graph theory

The length of the Hamiltonian-like word or the Eulerian-like word accepted by strongly connected automata can give insight into identifying areas where the system may be unnecessarily complex or redundant and help simplify the design of the automaton to improve its performance.

REFERENCES

Ádám, A. (1970). On some generalization of cyclic networks. 105-119.

Ádám, A. (1973). On some open problems of applied automaton theory and graph theory. In *The art of counting* (pp. 574—617). M. I. T. Press, Cambridge (Mass.).

Ádám, A. (1975). On graphs satisfying some conditions for cycles, I. 3-13.

Ádám, A. (1983). On certain partitions of finite directed graphs and of finite automata. *Acta Cybernetics VI/4*, 331-346.

Edgar G. Goodaire, & Michael M. Parmenter. (1998). *Discrete Mathematics with Graph Theory.* Prentice-Hall, Inc.

Horváth, G., & Nagy, B. (2014). *Formal Languages and Automata Theory.* Budapest: Typotex.

Ito, M. (1978). A representation of strongly connected automata and its applications. *Journal of Computer and System Sciences*(17), 65-80.

John E. Hopcroft, Rajeev Motwani, & Jeffrey D. Ullman. (2006). *Introduction to Automata Theory, Languages, and Computations.* Prentice Hall.

Kleene, S. C. (1951). Representation of events in nerve nets and finite automata. In *Automata Studies* (pp. 2-42). N.J.: Princeton Univ. Press, Princeton.

Linz, P. (2011). *An introduction to formal languages and automata* (Vol. 5th). Cathleen Sether.

Nagy, B. (2006). Union-free languages and 1-cycle-free-path-automata. *Publicationes Mathematicae Debrecen 68*, 183-197.

Rodaro, E. (2018). Strongly connected synchronizing automata and the language of minimal reset words. *Advances in Applied Mathematics*, 158-173.

Rozenberg G., &. S. (1997). Handbook of Formal Languages. Berlin: Springer.

Sakarovitch, J. (2009). *Elements of Automata Theory.* Cambridge University Press.

Sudkamp, T. A. (2005). Languages and Machines: An Introduction to the Theory of Computer Science.

Volkov, M. V. (2008). Synchronizing automata and the Cerny conjecture. *Language and Automata Theory and Applications, Lecture Notes in Computer Science*, *5196*, pp. 11-27.