Algebraic Characterization of Star-free Languages and Their Subclasses

Esther(Xinning) Fang and Vinty(Liwen) Guo
Advised by Professor Steven Lindell

Submitted in Partial Fulfillment of the Requirements of the BA in Computer Science
Bryn Mawr College
Spring 2020

Abstract

This paper focuses on the connection between formal language theory and algebraic automata theory. In particular, the paper starts with the theorem by Marcel-Paul Schützenberger, who stated the algebraic characterization of star-free languages as a whole. Looking into details, the paper will look at two subclasses of star-free languages, piecewise testable languages and locally testable languages with their relationship between algebra. In the end, subclasses of piecewise testable languages and locally testable languages, strictly piecewise testable languages and strictly local languages will be examined, and their algebraic characterizations will be provided as well. Combining and summarizing the results and proofs provided by various papers, this paper aims to offer an understandable and clear collection of algebraic characterizations of star-free languages and their subclasses while giving algorithms for testing whether a given language belongs to any type of the languages mentioned.

Contents

1 Introduction **3**

2 Background **4**

2.1 Formal Language Theory 4

2.2 Semigroup Theory 6

2.3 Star-free Languages 8

2.4 Aperiodic Monoids 8

2.5 Schützenberger's Theorem 11

3 Literature Review **12**

3.1 Schützenberger's Theorem 12

3.2 Piecewise Testable Languages and Strictly Piecewise Languages 12

3.3 Strictly Local Languages and Locally Testable Languages 13

4 Methodology **15**

4.1 Generating Finite Semigroup/Monoid Using Finite Automata 15

4.2 Piecewise Testable Languages and Strictly Piecewise Languages 17

4.3 Strictly Local Languages and Locally Testable Languages 21

5 Results & Analysis **27**

5.1 Piecewise Testable Languages 27

5.2 Strictly Piecewise Languages 27

5.3 Strictly Local Languages 28

5.4 Locally Testable Languages 28

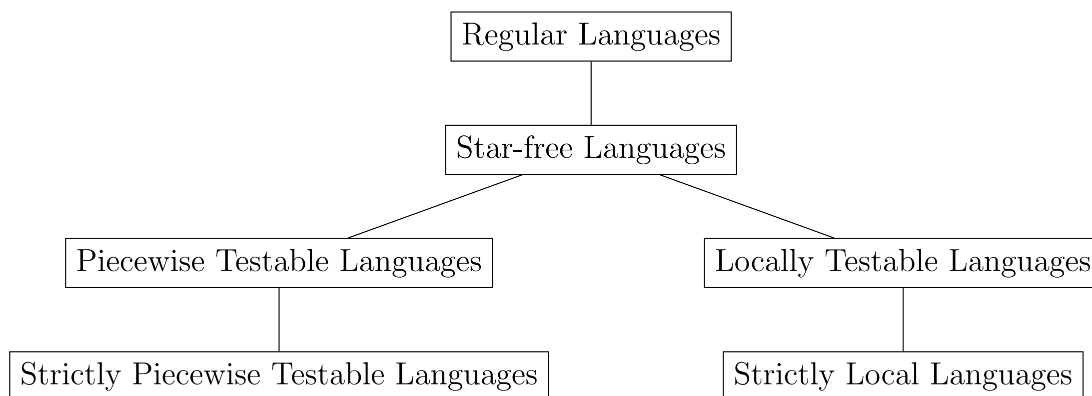
6 Future Work **29**

7 Summary **30**

1 Introduction

Formal languages are a concept commonly used in Computer Science. Programming languages are counted as formal languages. What's more, the concept of formal languages is not only being used in Computer Science, but also being studied in Mathematics and Linguistics. This paper is motivated by the close connection between theoretical computer science and algebra. We aim to present how subclasses of regular languages can be represented by semigroups and monoids in terms of algebra, while introducing algebraic automata theory to both computer scientists and mathematicians.

This paper concentrates on the algebraic characterizations of star-free languages as well their subclasses. The hierarchy of the languages we will cover is like the graph below:



To display such a connection, this paper will be divided into these sections:

1. Background on both Formal Language Theory and Semigroup Theory
2. Schützenberger's Theorem (algebraic characterization of star-free languages)
3. Piecewise Testable Languages, Strictly Piecewise Testable Languages, Locally Testable Languages, and Strictly Local Languages

With all background concepts being set up, we will first present Schützenberger's Theorem, which states the relationship between star-free languages and aperiodic monoids. Then we will move to the four subclasses of star-free languages: piecewise testable languages, strictly piecewise testable languages, locally testable languages, and strictly local languages. With the given theorems for each language, we will develop algorithms to test whether the given language belongs to the particular class with the help of semigroups and monoids in Mathematics. Examples will be given to show how the algorithms work.

In conclusion, this paper will examine star-free languages with their subclasses. Algebraic characterizations for each class of language will be developed with a clear explanation of how to distinguish the given language using the combination of finite automaton and

semigroups/monoids. Through these ways, the connections between languages and semigroups/monoids are built.

2 Background

This section will introduce basic notations, definitions, theorems which will be used to establish algebraic characterizations of star-free languages, piecewise languages, and locally languages. The section will be divided into two subsections, one focusing on Group Theory and one on Formal Language Theory.

2.1 Formal Language Theory

Definition 1. [3] Given a set V, define

$$V^0 = \{\varepsilon\}(\text{the language consisting only of the empty string}),$$

$$V^1 = V$$

and define recursively the set

$$V^{i+1} = \{wv : w \in V^i, v \in V\}\text{for each } i > 0.$$

If V is a formal language, then V^i, the i-th power of the set V, is a shorthand for the concatenation of set V with itself i times. The definition of ***Kleene star (*) on*** V is

$$V^* = \bigcup_{i \geq 0} V^i = V^0 \cup V^1 \cup V^2 \cup V^3 \cup V^4 \ldots$$

Notice that $(V^*)^* = V^*$.

Now let Σ be a finite, nonempty alphabet, Σ^* be the set of all words over Σ and Σ^+ be the set of all nonnull words over Σ. We denote elements within the alphabet Σ using small latin letters $(a, b, c, etc.)$, which are called symbols. A word over Σ is a finite concatenation of symbols from Σ.

Example 1. $1^* = \{\varepsilon, 1, 11, 111, etc.,\}$

Example 2. $10^* = \{\varepsilon, 10, 1010, 101010, etc.\}$

Definition 2. Given an alphabet Σ, ***regular languages*** are defined by the following:

1. $\emptyset$ is a regular language

2. For any string $s \in \Sigma^*$, $\{s\}$ is a regular language

3. If two languages A and B are regular languages, then $A \cup B$ is also a regular language
4. If two languages A and B are regular languages, then AB is also a regular language
5. if a language A is a regular languages, then A^* is also a regular language
6. Nothing else forms a regular language

Example 3. Let $\Sigma = \{0, 1\}$. Then according to rules in **Definition 2**:

- By rule 2 $\{11\}$ is a regular language because $11 \in \Sigma^*$.
- Let $A = \{0\}$ and $B = \{1\}$. By rule 2 they are both regular languages. Then by rule 3

The class of regular languages could be recognized using a finite automata, which can recognize patterns by taking an input that contains a string of symbols and either accept or reject it.

Definition 3. [6] A ***finite state automata*** is a 5-element tuple $(\Sigma, Q, \delta, q_0, F)$ where:

- Σ is an alphabet
- Q is a finite number of states
- δ is a transition function from $Q \times \Sigma$ to Q
- $q_0 \in Q$ is the initial state, and
- $F \subseteq Q$ is the set of final states

Example 4. Let the finite automata A have $\Sigma = \{0, 1\}$, $Q = \{q_1, q_2\}$, $q_0 = q_1$, $F = q_1$, and δ be

	0	1
δ	0	1
q_1	q_1	q_2
q_2	q_2	q_1

Then this automata has two states in total, q_1 and q_2, where q_1 serves as both the start state and the accepting state. The transition function δ indicates that when the machine is curently at q_1 and reads symbol 0, it will stay at q_1. When it reads 1 instead, it will reach to state q_2. Similarly, when the machine is at q_2 and reads the symbol 0, it will stay at q_2 while reading 1 will let it proceed to q_1.

When using graphs with arrows to represent this automata A, the so-called state diagram looks like this:

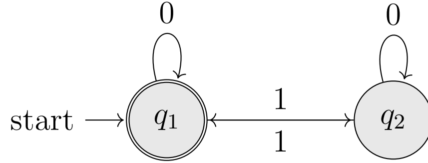


Fig 1: State diagram of A

Consider string "100". Starting with q_1, the machine will read "1" and proceed to q_2. Then it will remain at q_2 after reading "0" according to the transition function δ. To read another "0", the machine will still be at q_2, but it is not at the accepting state as all the symbols in the given string is given. Thus this input is rejected. To generalize, the language this finite automata accepts is the language that has even number of 1's because once a "1" is being read, another "1" has to be read in order to reach to the accepting state, namely q_1.

Theorem 1. A language L is regular if and only if it is a finite automaton language, *i.e.*, there is a finite automata A such that $L = L(A)$.

In **Example 4**, it is clear that we have a finite automata that accepts the language, and according to **Theorem 1**, this language that the finite automata A accepts is a regular language.

2.2 Semigroup Theory

To connect Formal Language Theories with Algebra, we need concepts from Group Theory, or Semigroup Theory in particular. This subsection will focus on semigroups and monoids, which would be useful when developing algebraic characterizations of languages.

Definition 4. [6] Let S be a nonempty set, a ***binary operation*** on S is defined by a function $S \times S \to S$.

Definition 5. [6] A ***semigroup*** is a nonempty set S with a binary operation on it that is associative:

$$\text{For all } a, b, c \in S,\ a(bc) = (ab)c.$$

Example 5. Let the set be $\mathbb{N} = \{1, 2, 3, ...\}$ and let the binary operation be addition.

The given set is a semigroup as the binary operation allows the sum of two natural numbers to also be a natural number. What's more, it is trivial that addition is associative. Similarly, the set is also a semigroup if the binary operation is multiplication instead of addition.

Definition 6. [6] A ***monoid*** is a semigroup S with an identity element $e \in S$ such that

$$se = es = s \text{ for all } s \in S.$$

Homomorphism is a mathematical term that acts as a tool to show the relevance of two different structures. In our cases, we shall use homomorphism to show the similarity of regular languages and monoids as well as the relation between them.

Definition 7. ***(Homomorphism)***

1. Given two semigroups S_1, S_2, a ***Homomorphism*** between S_1 and S_2 is a function $h : S_1 \to S_2$ such that for all $a, b \in S_1$, $h(ab) = h(a)h(b)$.

2. Given an alphabet Σ and a monoid M, a ***Homomorphism*** between Σ^* and M is a function $h : \Sigma^* \to M$ such that for all $a, b \in \Sigma^*$, $h(ab) = h(a)h(b)$ where ab is the concatenation of a and b.

Note that in the definition of the homomorphism from an alphabet Σ to a monoid M above, ab is the concatenation of two elements a, b in Σ^*, and $h(a)h(b)$ is the product of $h(a)$ and $h(b)$ in M since h is a function mapping from Σ^* to M. What's more, $h(\varepsilon)$ is the identity element of M.

Definition 8. A ***syntactic semigroup*** is defined as the smallest semigroup S such that there exists a homomorphism $h : \Sigma^+ \to S$ and $L \subseteq \Sigma^+$ such that $L = h^{-1}(F)$ for some $F \subseteq S$.

We can use the graph below to illustrate the relation described in **Definition 8**:

$$\begin{array}{ccc} L & \xleftarrow[h^{-1}]{} & F \\ \subseteq & & \subseteq \\ \Sigma^+ & \xrightarrow{h} & S \end{array}$$

This graph shows the relationship between subsets and languages that was given in the previous definition. Note that if S is finite, then the language L got will be regular.

The following three definitions on monoids are the key terms in algebraic characterizations of piecewise testable languages as well as in the algorithms for verifying piecewise testable languages.

Definition 9. Given a language L, the ***syntactic monoid*** of L is the smallest monoid M such that there's a homomorphism $h : \Sigma^* \to M$ where $L = h^{-1}(F)$ for some $F \subseteq M$.

Definition 10. [7] The set of strings that contain w as a subsequence is the (principal) ***shuffle ideal*** of w:

$$SI[w] = \{v \in \Sigma^* | w \sqsubseteq v\}$$

Definition 11. [4] A syntactic monoid M is ***J-trivial*** if and only if for all $a, b \in M$, if $MaM = MbM$, then $a = b$ (i.e. M's principal shuffle ideals are distinct).

$$\forall a, b \in M, MaM = MbM \rightarrow a = b \iff M \text{ is } \textbf{\textit{J-trivial}}$$

The following two definitions are the key terms in algebraic characterizations of strictly local languages and locally testable languages as well as in the algorithms for checking which language the given one belongs to.

Definition 12. [10] Let S be a semigroup. An element of $e \in S$ is called ***idempotent*** if an only if $e^2 = e$.

Example 6. 0 is always an idempotent element in semigroups because $0 \cdot 0 = 0$.

Definition 13. Let S be a semigroup. eSe is called a ***local subsemigroup*** when $e \in S$ is an idempotent element.

2.3 Star-free Languages

Here we introduce the terminology called ***Star-Free Languages*** which will be discussed and used more later in this paper.

Definition 14. [2] A regular language is ***star-free*** if it can be described by a regular expression constructed from the letters of the alphabet, $\emptyset$, and all boolean operators including complementation and concatenation (no Kleene star *).

Example 7. A regular language 1*0*, which represents all strings constructed with the alphabet set {0,1} except the strings that have 0 before 1 (all strings without a "01"), is star-free because 1*0* can also be expressed as $\overline{\overline{\emptyset}01\overline{\emptyset}}$, an regular expression without Kleene star *.

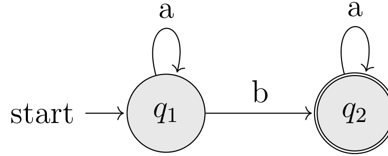
2.4 Aperiodic Monoids

Now we shall introduce the terminology called ***Aperiodic Monoids*** which will be discussed and used in the next section. Below is the informal definition of an aperiodic monoid based on ***monoids multiplication table*** followed by an example.

Definition 15. A monoid is ***aperiodic*** if there is no identity e in its **monoids multiplication table** other than $e \cdot e$.

Example 8. Consider the language $L = a^*ba^*$ over an alphabet $\Sigma = \{a, b\}$ all strings that contains exactly 1 "b". We shall show that the monoid of the language L is aperiodic.

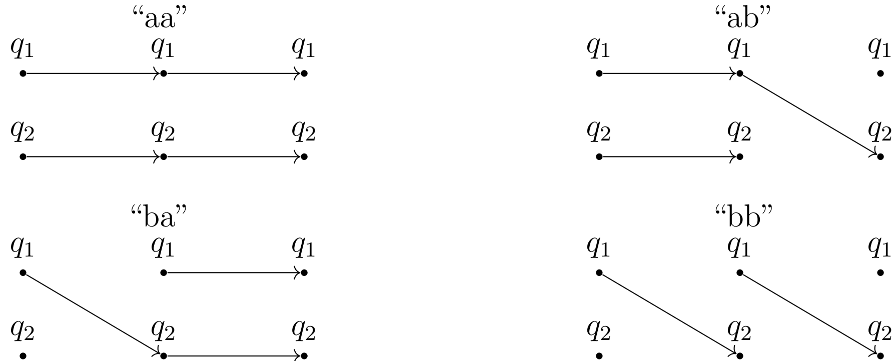
To construct a monoid, our first step is to construct a finite automata for L. The corresponding finite automata looks like:



We could see from this finite automata that in order to end at the accepting state, the input string always has to go from state q_1 to state q_2 exactly once. A string in this language L may have zero or more "a"'s in front of and after the middle "b". Any other possibilities are excluded. Therefore, this finite automata only accepts language $L = a^*ba^*$.

With the finite automata, we can find all the partial functions that are distinct. Starting with strings with length 1, which are "a" and and "b", the partial functions we have are:

Next we will find the partial functions for strings with length of 2, which are "aa", "ab", "ba", and "bb". The functions are:



Looking at the partial function for "bb", we can see that there is no connection in the middle state, which means that the string "bb" would not be accepted. Thus, we can induce that $b \cdot b = 0$. Also, we can see that the partial function for "aa" is the same as the one for "a", mapping each state to itself. Thus, we have conclude that $a = e$, which is the identity. In addition, we can find that the partial functions for "ab" and "ba" are connected by one middle state (respectively q_1 and q_2). Therefore, we can simplify these two partial functions into:

From above, we can see that "*ab*" = "*ba*" since they have the same pattern for their partial function mappings. So far we already have all the distinct partial functions and there is no new partial functions that can be developed. Then we can create the *monoid multiplication table* by using the these partial functions:

$\cdot$	a/e	b
a/e	a/e	b
b	b	0

We also include 0 in the monoid multiplication table because "*bb*" is not any element in Σ. Now by **Definition 15**, we can conclude that the monoid of this language $L = a^*ba^*$ is **aperiodic** and this L is an aperiodic language.

As we have shown earlier with examples that **Monoids** are actually just another representation of finite automata and thus of regular languages as well, we now can introduce the following equivalent conditions:

Theorem 2. Given a finite monid M, the following statements are equivalent:

1. M contains no non-trivial groups, i.e., $\forall G \subseteq M$, if G is a group, then $G = \{e\}$, meaning that G is trivial.
2. $\forall a \in M, \exists n \in \mathbb{Z}^+$ such that $a^n = a^{n+1}$.
3. No non-trivial element of M is invertible, i.e., $\nexists m \neq e$ such that m^{-1} exists. This means the only invertible element in M is trivial, which is the identity element e.

We are going to prove part of this theorem by providing the proof of the following directions: Statement 2 implies statement 3, statement 2 implies statement 1, and statement 3 implies statement 1. Note that this series of proof is only the partial proof of this intermediate theorem.

Proof. $2 \rightarrow 3$

If M is aperiodic, then no non-trivial element of M is invertible. Suppose we have an aperiodic monoid M, meaning that $\forall m \in M, \exists n \in \mathbb{Z}^+$ s.t. $m^n = m^{n+1}$. Suppose we have an arbitrary invertible element m, we need to prove that m is the identity element. Since m is invertible, m^{-1} exists in M. Since M is aperiodic, for some $n \in \mathbb{Z}^+$,

$$m^n = m^{n+1}$$

$$m^n \cdot (m^{-1})^n = m^{n+1} \cdot (m^{-1})^n$$

$$m^n \cdot m^{-n} = m^{n+1} \cdot m^{-n}$$

$$e = m^{n+1-n} = m$$

Thus, m is equal to e. □

Proof. $2 \rightarrow 1$

If M is aperiodic, then M contains no non-trivial groups, meaning that every group in M is trivial (the group only has identity element e). By applying the same method used in proof (2 implies 3), we can easily prove this direction of the proof. □

Proof. $3 \rightarrow 1$

If the only invertible element in M is the identity element e, then any group in M is trivial. This direction of proof can proved by contradiction. Suppose we have statement 3, which is that the only invertible element in M is the identity element. And suppose the negation of statement 1, which is that there is a group g in M that is non-trivial. Then, this means g has an non-trivial element a where its inverse element a^{-1} is also in g. However, this contradicts statement 3 since the non-trivial element a in M is invertible. Thus, the negation of statement 1 is false once we suppose statement 3. So 3 implies 1. □

2.5 Schützenberger's Theorem

For this subsection, we will introduce the *Schützenberger's Theorem* based on the algebraic characterization talked about earlier.

Definition 16. A **regular language** L is ***aperiodic*** if there is a finite aperiodic monoid M and a homomorphism h: $\Sigma^* \rightarrow M$ such that $L = h^{-1}(F)$ for some $F \subseteq M$.

Finally, we can introduce THE statement of the *Schützenberger's Theorem*:

Theorem 3. A language is star-free if and only if its syntactic monoid is aperodic.

3 Literature Review

The readings below provide general ideas of and formal definitions of the terms, and also introduce some theorems used in the classes of languages we will be talking about and drawing examples upon. In order to conform with the flow of this paper, we divided the materials into the following sections.

3.1 Schützenberger's Theorem

In order to analyze PT, LT, SP, and SL languages, there are some background knowledge we need to acquire first. The paper *A Proof of Schützenberger' Thoroem by Alejandro Mallea*[6] contains the proof of Schützenberger's Theorem which we worked on before this paper, along with some basic knowledge needed before the proof in both Formal Language Theory and Algebra. The definitions contained in this source are clear and well-organized in terms of wording, which can be used for formal definitions of concepts such as regular languages, finite automata, semigroups, monoids, aperiodic, etc.. Definitions referenced from this paper are necessary for the eventual proof of Schützenberger's Theorem. With these definitions, we could come up with examples to help with the clarity of the essential terms.

3.2 Piecewise Testable Languages and Strictly Piecewise Languages

The main source we use for reference for Piecewise Testable languages is *On Languages Piecewise Testable in the Strict Sense*[7] by Rogers and Heinz. In general, this paper mainly provides reviews on Piecewise Testable languages where most of their results are collected from the "folklore", explores Strictly Piecewise Testable languages by defining the language and properties and providing some abstract characterizations of the class, and also presents algorithms related to SP_k (k-strictly piecewise languages). For our thesis, we mainly focus on its reviews on Piecewise Testable languages, and also on its exploration of Strictly Piecewise languages.

The authors state in the paper that the relationship between the Strictly Piecewise (SP) languages and the Piecewise Testable (PT) languages is precisely analogous to the the relationship between the Strictly Local (SL) languages and the Locally Testable (LT) languages, meaning that SP languages and SL languages are in the same language hierarchy in a way that their super classes are PT languages and LT languages respectively which are also in the same language hierarchy. In addition, LT languages and PT languages share a lot of similarities and one way to differ these two classes of languages is by "how they determine an expression's well-formedness". For example, for LT languages, the set of *continuous* subsequences in the expression determines the expression's well-formedness. The set of *continuous*

subsequences up to length k is known as *k-factors*. And for PT languages, it is only the set of subsequences that determine the expression's well-formedness. The paper also states that SP languages is the class of languages that are closed under subsequence.

This paper formally defines the concept of *(principal) shuffle ideal* which will be talked about more and given example upon in the next section. Also, the paper suggests that the class of "PT languages is the smallest class of languages including principal ideals of w for all $w \in \Sigma^*$ and is closed under Boolean operations" while the "the class of k-Piecewise Testable (PT_k) languages is the smallest class of languages including principal ideals of w for all $w \in \Sigma^{\leq k}$ and closed under Boolean operations." In addition, this paper extends the idea of *shuffle ideal* to languages which will be used in a theorem that verifies a PT language.

Despite all the well-formed definitions and theorems mentioned in Rogers' and Heinz' paper, there are some equivalent/alternative definitions of PT languages introduced in *An Algebraic Characterization of Strictly Piecewise Languages*[4] by Fu, Heinz, and Tanner, and the definitions in this paper will be used in our examples in the next section. Fu's and Heinz's paper[4] contains reviews of fundamental concepts, defines PT, SP, and SL classes, and presents an algebraic characterization of the SP (Strictly Piecewise) class. Later on in this paper, we will also use the their definitions and theorems on SP languages.

In addition to the theorems and definitions mentioned above in the first source, this paper mentions a relation called ***J relation*** defined on semigroups, and also the concept of ***J-trivial***. The algebraic characterization of PT languages using ***J-trivial*** on their syntactic monoids was proved to be practical in verifying PT languages by Imre Simon in 1975. And then, this paper provides an example of a PT language with its canonical acceptor, which we will examine more closely in the next section.

Additionally, regarding SP languages, they introduce the ideas of *wholly nonzero* and *right annihilator* which they use to define SP languages. They also introduce a theorem on *wholly nonzero* which we will try to provide alternative proof in the next section if time permits. There is, however, a tiny mistake (we believe) in its section after the proof of the theorem just mentioned, and we will try to address that in the next section as well.

3.3 Strictly Local Languages and Locally Testable Languages

Generally, local languages are defined based on the notion of substring, so we dedicated to finding definitions that conform to this fundamental idea. Though there weren't many resources available, we still found a variety of definitions for strictly local languages and locally testable languages. It is crucial to sort the content to find definitions and theorems that are useful when talking about the algebraic characterizations.

For each language, we have a definition in terms of the language itself and a definition for the semigroup, which sets the basis for the theorem that we will further establish for

algebraic characterizations.

First we will talk about the two papers we picked with the definitions needed for strictly local languages. One paper is *Aural Pattern Recognition Experiments and the Subregular Hierarchy* by Rogers and Pullum [8]. Though Rogers and Pullum focused on aural pattern recognition in their paper, there are materials that set basis for strictly local languages. Among them suffix substitution closure gives us a clear understanding of strictly local languages regarding formal language theory. For the mathematical definition involving semigroups, we chose the paper by Luca and Restivo. As the name of the paper, *A Characterization of strictly Locally Testable Languages and Its Application to Subsemigroups of a Free Semigroup* [5], indicates, Luca and Restivo concentrated on the characterization of strictly local languages. In this paper we found the equivalent conditions of a semigroup being strictly local and a property about all idempotents of the semigroup unambiguous, which is perfectly suitable when we build the algorithm to test whether a language is strictly local.

Following strictly local languages, we will study locally testable languages. As we mentioned, the class of strictly local languages is subclass of the class of locally testable languages. Among numerous definitions of locally testable languages, we found a sentence by Zalcstein in his *Locally Testable Languages* concise and understandable. He stated that the boolean closure of strictly locally languages is the collection of locally testable languages [10]. This definition let us introduce strictly local languages before locally testable languages with the help of definitions for strictly local languages we adopted in this paper. What's more, Zalcstein also gave a clear defintion of idempotents, which is the central concept used in both strictly local and locally testable languages when approaching their algebraic characterizations.

Lastly, we found another resource that is relative to algebraic characterization of locally testable languages. In *Characterizations of Locally Testable Events* by J.A.Brzozowski and Imre Simon, **Definition 3.2** [1] gives us a not only simple but also computable definition of a semigroup being locally testable. With this definition, we can build the algorithm to check if a language is locally testable or not with the algebraic characterization established.

In one word, the concepts in the four resources we chose to include define the languages either in formal language theory or in semigroup theory. Using the definitions from these papers, we are able to find the algebraic characterizations of strictly local and locally testable languages while coming up with algorithms that are easy to understand.

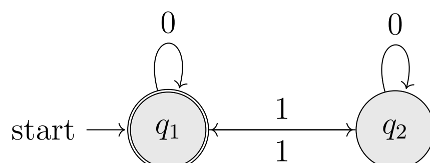
4 Methodology

In this section, we will focus on the algebraic characterizations of subclasses of star-free languages: piecewise testable languages and locally testable languages with their subclasses. This section consists of three parts. The first one will introduce a method to construct finite semigroups and monoids, which would be useful when we reach to the algebraic characterization of each language. This method will be helpful when we are testing whether the a language belongs to any class. Following that section come the two sections disscussing piecewise testable and locally testable languages. Besides algebraic characterizations, Algorithms will be provided to distinguish whether a given language belongs to the class of languages in the discussion.

4.1 Generating Finite Semigroup/Monoid Using Finite Automata

The first step to establish relation between regular languages and Algebra is to produce corresponding monoids of the languages. This section will use an example to show the process of producing a monoid from the given regular language.

Recall **Example 4** in the previous (sub)section, where we have an alphabet $\Sigma = \{0, 1\}$ and the language L over Σ is the language containing even number of 1's. Having the definition of L, we could build the finite automata, which was already given as follows:



With the finite automata, we could come up with partial functions representing the relationship between two states for different symbols accepted until no distinct partial functions are produced. Starting with the two shortest symbols, "0" and "1", we can produce the following partial functions:

The partial function on the left means that there are two ways to get a "0": starting from q_1 and staying at q_1 or starting from q_2 and staying at q_2, which are shown by the loop back to q_1 and q_2 themselves in the finite automata. Similarly, there are two ways to obtain a "1": starting from q_1 and reaching q_2 or starting from q_2 and reaching q_1.

Going further, we could consider building upon the two partial functions produced above, which will give us "00", "01", "10", and "11". We have:

For both partial functions above, it could be seen that if the machine starts from q_1, it will stay at q_1 after reading the input; if it starts from q_2, it will stay at q_2 after reading the input, and the results are the same as the result got when only reading "0". Similarly, when "01" and "10" are read respectively, the partial functions are:

We could find that both the results are same as the result of reading in "1".

Therefore, we have $0 \cdot 0 = 1 \cdot 1 = 0$ and $0 \cdot 1 = 1 \cdot 0 = 1$. from these states, no matter how only the input string is, we will always have the partial functions that are either the same as the one for "0" or the same as the one for "1". Since no more distinct partial functions will be produced, we can now set up the multiplication table:

$\cdot$	0	1
0	0	1
1	1	0

From the multiplication table we generated above, we can see that 0 serves as the identity element since for all $a \in \Sigma$, $0 \cdot a = a \cdot 0 = a$ because $0 \cdot 0 = 0$ and $1 \cdot 0 = 0 \cdot 1 = 1$. According to the multiplication table, it is clear that the set has binary operation, *i.e.,* $\Sigma \times \Sigma \to \Sigma$, since only 0 and 1 are produced where $0, 1 \in \Sigma$. By **Definition 5**, we have established a finite semigroup of the language L. If the semigroup we construct has an identity element, then we also obtain a finite monoid for L.

To summarize, when given a language L, we need three steps to find the corresponding monoid:

1. Building a finite automata that represents L,

2. Finding all distinct partial functions, and

3. Creating a multiplication table according to the partial functions.

Given such a process to generate a semigroup or a monoid of a language, we can move on to the algebraic characterizations of piecewise testable, locally testable languages and their subclasses. We will first introduce the class of piecewise testable languages along with its subclass, strictly piecewise languages, followed by the inspection on strictly local languages and locally testable languages.

4.2 Piecewise Testable Languages and Strictly Piecewise Languages

Before diving into PT languages, we first introduce a concept of *Subsequence* used in later sections.

Definition 17. A subsequence is a sequence developed from another sequence by deleting some or no elements(letters) without changing the order of the remaining elements(letters).

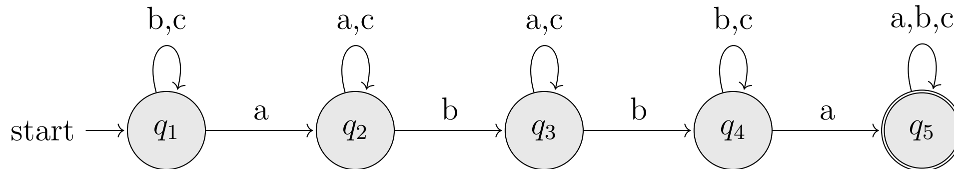
If sequence B is a subsequence of sequence A, we write $B \sqsubseteq A$.

Example 9. Let $A =$ "*kababcabcabcq*" and let $B = kq$. We say $B \sqsubseteq A$ because we can get B from A by deleting all the elements except the first one and the last one from the sequence without changing the order of the remaining elements, which are "*kq*".

The class of Piecewise Testable languages was introduced by Imre Simon[9]. Following him and some other scholars, we introduce the concept of *(principal) shuffle ideal*, which is used in the definition of Piecewise Testable languages.

Regarding **Definition 10** of *Shuffle Ideal*, we present the following example in order to help better understand this basic concept.

Example 10. The below is a DFA that recognizes the shuffle idea of $w = abba$ over alphabet $\Sigma = \{a, b, c\}$.



As we can see from this DFA, this DFA accepts precisely those strings that contain $w = abba$ as a subsequence. Take the string "*baabbaa*" as an example. *abba* clearly is a subsequence of the string "*baabbaa*" (since *abba* can be drived from "*baabbaa*" by deleting some letters while the order of the remaining letters is preserved), meaning that the shuffle ideal of *abba* contains the string "*baabbaa*". And this DFA accepts the string "*baabbaa*", which can be verified by checking every letter of the string in order on this automata.

Now it would be easier to understand when we introduce the **class** of Piecewise Testable (PT) languages using the idea of shuffle ideal.

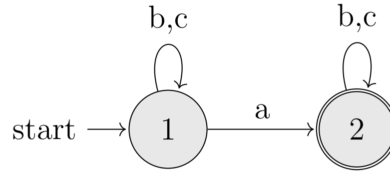
Definition 18. [7] The **class** of **Piecewise Testable** languages is the the smallest class of languages including SI(w) for all $w \in \Sigma^*$ and closed under Boolean operations $(\cdot, +, ^{-})$.

A chracterization of Piecewise Testable languages was introduced by Imre Simon in 1972 and it uses the concepts of ***J-trivial*** and *syntactic monoids*. Now we have introduced the definitions of *syntactic monoids* and ***J-trivial*** in **Definition 9** and **Definition 11** in the background, we can now look at Simon's Theorem of Piecewise Testable languages using those concepts more closely.

Theorem 4. (Simon 1975) A language is Piecewise Testable iff is syntactic monoid is ***J-trivial***.

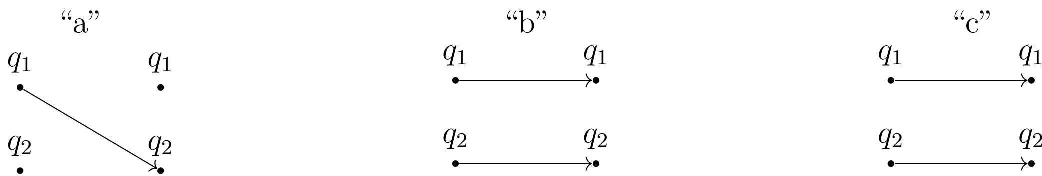
Now we will examine an language as an example to see if it is Piecewise Testable by the above theorem.

Example 11. Consider the language $L = (b^*c^*)a(b^*c^*)$ over an alphabet $\Sigma = \{a, b, c\}$ having all words with exactly one a [4]. And NFA that represents this language is shown below:



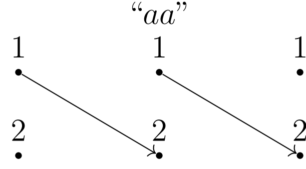
We can see from this NFA that it is impossible to not have exactly one "a" because in order to reach to the accepting state from the starting state, the machine has to go through the "a" transition exactly once. Therefore, this automata accepts the language L.

with the finite automata, we can find all the partial functions that are distinct. Starting with strings with length 1, which are "a", "b", and "c", the partial functions we obtain are:



As we can see, the partial function for the string "b" and the one for "c" are identical and they behave like identity 1 in multiplication. Thus, we use 1 to represent "b" and "c".

Next, we are going to obtain the partial functions for strings with length 2, namely "aa", "ab", "ac", "bc", "ba", "ca", and "cb". Since we have concluded that "b" and "c" are identity, meaning that every element multiplied by them equals that element itself, we only have "aa"" to construct partial functions for.



As we can see from the partial function for "*aa*", there is no connection in the middle, which means the string "*aa*" would not be accepted by the automata. Thus, we can include that $a \cdot a = 0$. Then, we can create the *monoid multiplication table* by using the above partial functions where b, c are represented by identity 1:

·	1	a	0
1	1	a	0
a	a	0	0
0	0	0	0

Now we shall check if its syntactic monoid M is ***J****-trivial*. Recall that in order for a syntactic monoid to be ***J****-trivial*, we need for all $a, b \in M$, if $M\text{a}M = M\text{b}M$, then $a = b$. In this example, we have

$$MaM = \{0, a\}, \text{ since no matter what M's are, } MaM \text{can only be either 0 or a}$$

$$M1M = \{a, 1, 0\} = M$$

$$M0M = \{0\}$$

As we can see, all of the three two-sided principal ideals are distinct, meaning that its syntactic monoid is ***J-trivial*** since all shuffle principal ideals in this monoid M are distinct. Besides, this is also an *aperiodic monoid/language* by **Definition 15**.

Looking more closely into the PT languages, we now introduce a even smaller class of languages under PT languages: Strictly Piecewise (SP) Languages. Below is the general definition of SP languages.

Definition 19. A language L is **Strictly Piecewise** iff it is closed under subsequences:

$$L \text{ is SP iff } \bigcap_{i=1}^{n} \overline{SI(w_i)} = L$$

Strictly Piecewise Languages fall into subclasses of Piecewise Testable Languages, and Piecewise Testable Languages are Star-Free Languages, which means SP is also a subset of SF ($SP \subseteq PT \subseteq SF$). The proof of this statement could be sketched by the following: Since SP is closed under subsequences, then by *Higman's Lemma*,

$$L \in SP \Rightarrow L = \{y : x_1, x_2, x_3..., x_k \sqsubseteq y\} = \bigcap_{1 \leq i \leq k} \{y : x_i \sqsubseteq y\} \in PT \subseteq SF$$

Thus, any SP language is a PT language, which is a SF language, meaning that $SP \subseteq PT \subseteq SF$.

Before we talk more about the theorem on Strictly Piecewise (SP) languages, we need to introduce the notions of *wholly nonzero* and *annihilating* as follows:

Definition 20. [4] A language is **wholly nonzero** if the image of its complement under the fundamental homomorphism is zero. In other words, Language L is wholly nonzero iff $\overline{L} = [0]$, meaning every word x not in the language L is zero.

Now we introduce the notion of *annihilating* languages.

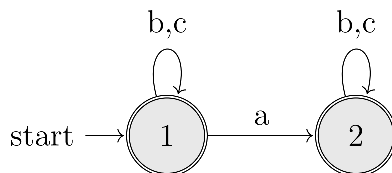
Definition 21. A language is **annihilating** if its syntactic monoid satisfies: $xz = 0 \Rightarrow xyz = 0$.

With the above background, we now introduce the the theorem that verifies SP language by the following:

Theorem 5. [4] A language L is **Strictly Piecewise** (SP) iff L is wholly nonzero and annihilating.

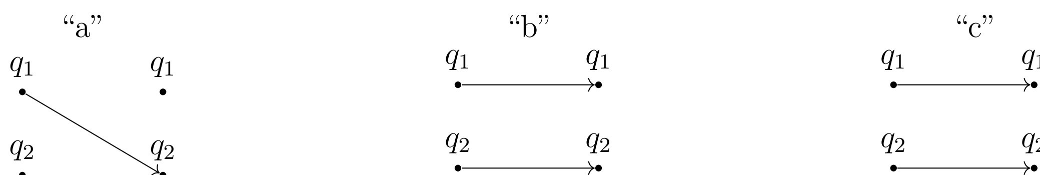
Example 12. Let $\Sigma = \{a, b, c\}$ and let the language $L = (b^*c^*) \cup (b^*c^*)a(b^*c^*)$ having no more than one a. We shall show that L is strictly piecewise testable language.

To construct a syntactic monoid, our first step is to construct a finite automata for L. The corresponding finite automata is the following:



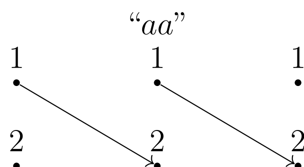
We can see from this finite automata that it is impossible to have more than one "a" because there's only one one-way "a" transition. The machine will either stop with or without going through that transition, meaning it will either proceed "a" once or not proceed it at all. Therefore, this finite automata will reject any string on $\Sigma = \{a, b, c\}$ that has more than one "a".

With the finite automata, we can find all the distinct partial functions. Starting with strings with length 1, which are "a", "b", and "c", we obtain the following partial functions:



As we can see, the partial function for the string "*b*" and the one for "*c*" are identical and they behave like identity 1 in multiplication. Thus, we use 1 to represent "*b*" and "*c*".

Next, we are going to obtain the partial functions for strings with length 2, namely "*aa*", "*ab*", "*ac*", "*bc*", "*ba*", "*ca*", and "*cb*". Since we have concluded that "*b*" and "*c*" are identity, meaning that every element multiplied by them equals that element itself, we only have "*aa*"" to construct partial functions for.



As we can see from the partial function for "*aa*", there is no connection in the middle, which means the string "*aa*" would not be accepted by the automata. Thus, we can include that $a \cdot a = 0$. Then, we can create the *monoid multiplication table* by using the above partial functions where b, c are represented by identity 1:

·	1	a	0
1	1	a	0
a	a	0	0
0	0	0	0

With the above *monoid multiplication table*, we can check whether L is wholly non-zero. As we can see, the only word that is zero is the word that contains more than one "*a*", namely "*aa*", "*abab*" = "*ab* · *ab*" = "*a* · *a*" = 0, and so on. Thus, every word not in this language L, which is the word that has more than one "*a*", is zero. Then, by **Definition 20**, this language L is **wholly non-zero**.

Also from the *monoid multiplication table*, let $y \in \Sigma^*$, "*aa*" = 0 ⇒ "*aya*" = 0 as well. This is because in this alphabet, there is only one "deciding" element "*a*", the rest of them ("*b*" and "*c*") are all identities, which means the product will always follow the other element multiplied. Thus, "*y*" will either be 1 or "*a*". Either way, "*aya*" = 0. Therefore, this language L is **annihilating** by **Definition 21**, and finally we have shown that this language L is Strictly Piecewise.

4.3 Strictly Local Languages and Locally Testable Languages

In this section, we will start with the smaller class, strictly local languages, which can be distinguished on the basis of substrings relying on the symbols that are adjacent to each other. With necessary definitions and theorems, we will introduce the theorem that states the relation between strictly local languages and semigroups with an example showing how

to use the theorem to test whether a language is strictly local languages. Afterwards, we will look at a broader class of languages, locally testable languages. Similarly, the theorem will be stated in terms of the algebraic characterization of locally testable languages along with an example.

Before getting into definitions of strictly local languages, we are going to introduce the concept of k-factors, which can be defined as a string drawn from an alphabet with the length k. The formal definition of k-factors is:

Definition 22. [8]

$$F_k(w) = \begin{cases} \{y \mid w = x \cdot y \cdot z,\ x, y, z \in \Sigma^*,\ |y| = k\}, & \text{if } |w| > k, \\ \{w\} & \text{otherwise.} \end{cases}$$

Example 13. Let $\Sigma = \{a, b\}$. Consider $w =$ "$aabaa$" and $k = 2$.

The set of 2-factors of w is $\{\triangleright a, aa, ab, ba, a\triangleleft\}$, where $\triangleright$ is the initial marker and $\triangleleft$ is the final market of a string. Note that initial and final markers are not counted in the alphabet.

A finite string w is strictly k-local if $F_k(\triangleright \cdot w \cdot \triangleleft) \subseteq F_k(\{\triangleright\} \cdot \Sigma^* \cdot \{\triangleleft\})$. A language is strictly local if and only if it is strictly k-local for some k. In other words, a strictly local language is the union of strictly k-local languages for all finite k in the given alphabet.

Another way to define strictly local languages is to use Suffix Substitution Closure [8], which is defined as:

Definition 23. A language L is strictly local language if and only if there is some k such that for any string x whose length is $k-1$ and strings u_1, v_1, u_2, v_2 such that $u_1xv_1 \in L$ and $u_2xv_2 \in L$, then $u_1xv_2 \in L$.

This tells that if there are two strings in the set with the same substring, then if we substitute suffix of one string with the suffix of the other, than the resulting string is still in the language. The x that is remained the same is also called a ***constant*** [5]. We can also interpret strictly local languages as excluding certain substring in the given alphabet. Let the ignored substring be a concatenation of letters $x_1, x_2, ..., x_k$, then a language that is strictly local can be shown as $\{y \mid x_1, x_2, ..., x_k \not\subseteq y\} = \bigcap_{i=1}^{k}\{y \mid x_i \not\subseteq y\}$. Since this notation could be rewritten in a star-free way, which is $\bigcap_{i=1}^{k}\{\overline{\overline{\emptyset}x_i\overline{\emptyset}}\}$, we can easily verify that strictly local languages are also star-free languages.

Now we shall introduce the related mathematical concepts needed to set up the algebraic characterization of strictly local languages. We will start with syntactic semigroups. Instead of using syntactic monoids, we will examine the relationship between syntactic semigroups and strictly local and locally testable languages. Recall from **Definition 5** that a semigroup

is a nonempty set with a binary operation on it that is associative. Now we are going to define syntactic semigroup, which is also a key concept that will be used in the algebraic cahracterization of strictly local and locally testable languages.

The idea of idempotent of a semigroup is important in testing whether a language is a strictly local language after the language being converted into a syntactic semigroup. It is defined as the following.

The following theorem introduced by Luca and Restivo connects strictly local languages with a certain property of semigroup, which turns out to be a possible algebraic characterization of strictly local languages.

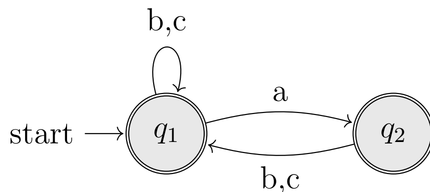
Theorem 6. [5] A language is strictly local if and only if in its syntactic semigroup S, every local subsemigroup $eSe \subseteq \{e, 0\}$.

With this clear theorem, we are able to test if a language L is strictly local using the following steps:

1. Find syntactic semigroup S for L by generating multiplication table
2. From the multiplication table, find all idempotents
3. For each idempotent e, compute eSe
4. Check if $eSe \subseteq \{e, 0\}$

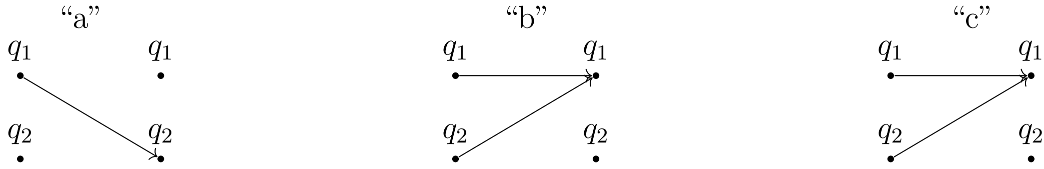
Example 14. Let $\Sigma = \{a, b, c\}$ and L be the language without two consecutive "a"s. We shall show that L is a strictly local language.

First of all we need to construct a syntactic semigroup so that we can use **Theorem 6** to check whether L is strictly local. We need to build a corresponding finite automata representing L, which looks like the graph below:



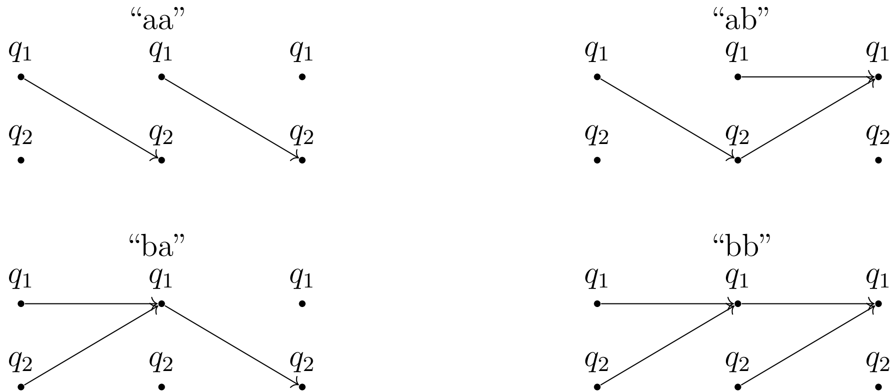
We could see from this finite automata that it is impossible to have consecutive two "a"s because after reading an "a", the machine will either stop at q_2 with nothing following the "a" that is read or continue reading "b" or "c" and procceed to q_1. Therefore double "a"s will never appear in the strings accepted by this finite automata.

With the finite automata, we can find all the partial functions that are different from each other. Starting with strings with length 1, which are "a", "b", and "c", the partial functions we get are:



It is obvious that the partial function for the string “b” and the partial function for “c” are the same. Since we are building a syntactic semigroup, we could ignore either symbol. Here we are going to ignore “c” and use “b” to represent both symbols.

Next we will get the partial functions for strings with length of 2, namely “aa”, “ab”, “ba” and “bb”. The functions look like:



Looking at the partial function for “aa”, we can see that there is no connection at the second state. This means that the string “aa” will not be accepted. Hence we could induce that $a \cdot a = 0$. Also, we can find that the partial function for “bb” can be converted to the same partial function of “b” because the only ways for “bb” to be recognized is starting from q_1 or q_2 and terminate at q_1. Then we could state that $b \cdot b = b$. Using the similar way, when only taking the starting states and the final states into consideration, we can convert the partial functions of “ab” and “ba” into:

To see whether there are more distinct partial functions, we should go further and look at the strings with length of 3: “aaa”, “aab”, “aba”, “abb”,“bbb”, “bab”, “baa”, “bba”. Since we just concluded that $a \cdot a = 0$, $aaa = 0$, $aab = 0$, and $baa = 0$. Similar, since $bb = b$, $abb = ab$, $bbb = bb = b$, and $bba = ba$. The only two partial functions we need to construct at this point are for “aba” and “bab”, and we could use the simplified partial functions for “ab” and “ba” from above:

After simplifying these two partial functions, we could see that $aba = a$ and $bab = b$.

At this point, there is no new partial functions being developed, and we have constructed all partial functions for this language. Using these partial functions, we will create a multiplication table as follows:

$\cdot$	a	b	ab	ba	0
a	0	ab	0	a	0
b	ba	b	b	ba	0
ab	a	ab	ab	a	0
ba	0	b	0	ba	0
0	0	0	0	0	0

We also include 0 in the multiplication table because $a^2 = 0$, which is not any element in Σ. To show that L is locally testable, we need to first find the idempotent in the syntactic semigroup. $I \subseteq S'$. To do that, we could simply look at the multiplication table and fine all $e \in S$ such that $e^2 = e$. In this example, $I = \{b, ab, ba, 0\}$. With the set of the idempotents, we can find $eS'e$ for each $e \in I$ as follows:

$$bSb = \{b, 0\}$$

$$abSab = \{ab, 0\}$$

$$baSba = \{ba, 0\}$$

$$0S0 = \{0\}$$

To find each eSe, we need to consider all elements in S' and do the multiplication. For instance, when we are getting $bS'b$, we can substitute S' by all elements $s \in S'$. We get $bab = (ba)b = b$, $bbb = b$, $babb = (ba)(bb) = (ba)b = b$, $bbab = (bb)(ab) = b(ab) = b$, and $b0b = 0b = 0$.

It is clear that for each idempotenet e except for 0, $eS'e$ only contains itself and 0, which indeed is a subset of $\{e, 0\}$. In the case of $0S'0$, 0 is the only element, a proper subset of $\{e, 0\}$. Hence by **Theorem 6**, the language L with no two consecutive "a"s is a strictly local language.

To conclude, when we are given a language L. the first thing is to construct the multiplication table by finding the finite automata and partial functions. With the multiplication table, we can check if for all idempotents e, the resulting sets $eSe \subseteq \{e, 0\}$.

We have introduced the algebraic characterization for strictly local languages, including an algorithm of testing whether a language is strictly local. Now we are going to take a look at the class of locally testable languages.

The class of strictly local languages is a subclass of locally testable languages. There is a restriction added to the idempotent of a semigroup, which will be stated in later definitions and theorems.

Definition 24. [10] The class of ***locally testable languages*** is the collection of taking closure of boolean operations of strictly locally languages.

As Zalcstein indicated, the strictly local languages are not closed under boolean operations, while one can obtain locally testable languages by simply taking their boolean closure, which would give us some clue on why the class of strictly local languages is the subclass of locally testable languages. As strictly local languages are star-free, locally testable languages are star-free as well.

Theorem 7. [1] A language is locally testable if all the local subsemigroups of its syntactic semigroup are idempotent commutative monoids.

In other words, for each idempotent $e \in S$, eSe is a idempotent commutative monoid.

Following **Theorem 7**, we can develop an algorithm to check if the given language is locally testable like what we did for strictly local languages. We will continue using the example for strictly local languages, as the language being used is also locally testable.

Example 15. In the previous example, we had got all the idempotents of the syntactic semigroup S and calculated eSe where e denotes idempotent, which are:

$$bSb = \{b, 0\}$$

$$abSab = \{ab, 0\}$$

$$baSba = \{ba, 0\}$$

$$0S0 = \{0\}$$

With 0 in all generated subsemigroup, we could tell that for every idempotent $e \in S$, the subsemigroup eSe is commutative since for all $s \in eSe$, $s \cdot 0 = 0 \cdot s = 0$. Also, all elements in each eSe are idempotents, as 0 is an idempotent and e itself is also idempotent. Thus by **Theorem 7** the language with no consecutive two "a"s is a locally testable language.

In general, to tell whether a language L is a locally testable language after getting the syntactic semigroup S, we can find all the idempotents of $e \in S$ and get eSe for each e to see whether the eSe got is commutative.

The reason why the collection of strictly local languages is a subclass of the collection of locally testable languages can be seen when we compare **Theorem 6** to **Theorem 7**. If eSe for all e being idempotents of the semigroup S contains only itself and 0, then it is commutative as $e \cdot 0 = 0 \cdot e$. In the below example we will show that the language given is a strictly local language.

5 Results & Analysis

This paper gives introduction to the connection between semigroup theory in Algebra and formal language theory in Theoretical Computer Science. Starting with Schützenberger's Theorem, the paper looks at algebraic characterizations of star-free languages and their subclasses, which are: strictly piecewise testable languages, piecewise testable languages, strictly local languages, and locally testable languages. Showing the processes of converting a language into a algebraic monoid or semigroup, we are able to construct algorithms for checking whether a language belongs to any of the four classes. To summarize the main results, we could split them into four parts:

1. Piecewise Testable Languages
2. Strictly Piecewise Languages
3. Strictly Local Languages
4. Locally Testable Languages

5.1 Piecewise Testable Languages

Piecewise Testable Languages use syntactic monoids and principal ideals to identify/verify whether a given language is PT. After following section 4.1 to obtain the the corresponding monoid and going through the definition of *J-trivial syntactic monoid*, we will be able to list all the principal ideals of the syntactic monoids: MxM for x being all generators. With these results, we can check if MxM for all x are distinct sets. If they are all distinct, then the language is piecewise testable. Otherwise, the language is not piecewise testable.

5.2 Strictly Piecewise Languages

Strictly Piecewise Languages use two criteria testify whether a language is SP: *wholly non-zero + annihilating = SP*. Similarly, first we need to go through the process of obtaining its syntactic monoid and the *monoid multiplication table*. Then, we can identify whether every

word not in this language is zero, and whether it is annihilating by the definition. If the language satisfies both, then it is a SP language. Otherwise, it is not a SP language.

5.3 Strictly Local Languages

Locally languages use syntactic semigroups instead of monoids. After obtaining the syntactic semigroup of the given language, we should find all idempotents using the multiplication table. For each idempotent e we get, we can compute eSe given that S is the resulting syntactic semigroup. With the results, we can check if the elements in each eSe is a subset of $\{e, 0\}$. If any of them doesn't follow this rule, then the language is not strictly testable language. If all idempotents appear to follow the rule, then the language is strictly local.

5.4 Locally Testable Languages

The process is similar to the one testing whether a language is strictly local. After having the syntactic semigroup S of the given language L, we should also compute eSe for each idempotent e. Then we can test if all sets of eSe is commutative and idempotent, and this will tell whether L is locally testable or not.

Except for the algebraic characterization we described above, we also gave sketches of proofs of showing the relationship between their subclasses and with star-free languages.

Throughout the process of looking for useful resources, we had encountered many different definitions of the same concepts. It was time-consuming for us to sort out clear ones and also appropriate ones. It was important for us to find ones that combine algebra and computer science. After figuring out that it was hard for us to come up with full proofs for the theorems, we dedicated in presenting easy ways to use the algebraic characterizations we got to test whether a language belongs to any class of languages we have studied. Therefore, compared to other papers that talked about piecewise or locally languages, we were lack of full proofs of theorems, but we were more clear and succinct on the algebraic characterization part.

6 Future Work

At the very beginning, we were aiming at showing the algebraic characterizations of locally and piecewise languages while giving proofs of the theorems. However, due to lack of time, we weren't not able to include proofs of the theorems of a language being piecewise testable/ strictly piecewise testable/ locally testable/ strictly local. Also we included a brief sketch showing that piecewise testable/ strictly piecewise testable/ locally testable/ strictly local are star-free as well as that strictly piecewise testable languages are subclasses of piecewise testable languages and that strictly local languages are subclasses of locally testable languages. With more time, we might be able to look deeper into those proofs and came up with full proofs. To conclude, if we are granted more time, we would mainly take a look at full proofs, try to see if we can develop more understandable versions with less background knowledge required, and look for more resources though there weren't many available.

7 Summary

This paper looks into the algebraic characterizations of PT, SP, SL, and LT languages, which are all subclasses of Star-Free languages, provides detailed examples, walks the verification of each, and talks about the hierarchy of these languages. We think this will provide a good resource for students who need help with some detailed steps which are usually not provided by more authoritarian resources. Also, the study of formal languages is becoming more and more popular and it is even more important in the field of linguistics. This topic of the paper might also be useful for students in that area. While composing this paper, we mainly exercised the following steps: 1. Doing literature review in order to be familiarize ourselves with the area and the materials, including all the background knowledge in Algebra and Logic, the terminologies, the proofs, and how they are used. 2. Searching for examples and also generating our own examples to walk through the steps to see if the materials/logic check out. 3. Collecting and organizing the sources to put them in the order that flows more naturally for students like us. The main results would be the four detailed series of steps to follow in order to verify those four languages. If time permits, we would want to dig more into the proofs and try to improve them by cutting out unnecessary steps or offering an alternative proof that requires less understanding in the background knowledge.

References

[1] J. Brzozowski and I. Simon. Characterizations of locally testable events. *Discrete Mathematics*, 4:243–271, 1973.

[2] V. Diekert and P. Gastin. First-order definable languages. pages 261–306, 01 2008.

[3] H. Ebbinghaus, J. Flum, and W. Thomas. *Mathematical logic (2. ed.)*.

[4] J. Fu, J. Heinz, and H. G. Tanner. An algebraic characterization of strictly piecewise languages. In *Theory and Applications of Models of Computation*, pages 252–263, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[5] D. A. Luca and A. Restivo. A characterization of strictly local languages and its application to subsemigroups of a free semigroup. *Information and Control*, 44:300–319, 1980.

[6] A. Mallea. *A Proof of Schützenberger's Theorem*. Pontificia Universidad Católica de Chile, Santiago, Chile, 2012.

[7] J. Rogers, J. Heinz, G. Bailey, M. Edlefsen, M. Visscher, D. Wellcome, and S. Wibel. On languages piecewise testable in the strict sense. pages 255–265, 01 2009.

[8] J. Rogers and G. Pullum. 1 introduction aural pattern recognition experiments and the subregular hierarchy. *Journal of Logic, Language and Information*, 20:329–342, 07 2011.

[9] I. Simon. Piecewise testable events. In *Automata Theory and Formal Languages*, pages 214–222, Berlin, Heidelberg, 1975. Springe, Berlin, Heidelberg.

[10] Y. Zalcstein. Locally testable languages. *Journal of Computer and System Science*, 6(2):151–167, 1972.