Journal of Multidisciplinary Engineering Science and Technology (JMEST)
ISSN: 2458-9403
Vol. 12 Issue 7, July - 2025

Formal Language and Finite Automata

Dr. Hieu D. Vu
Fort Hays State University
600 Park Street
Hays, KS. 67601
hdvu@fhsu.edu

***Abstract*—The subject of computer programming languages has been a very important topic in computer science since the development of computing machines. The Language, human (natural) language or programming language in theory is sets of strings that understandable, acceptable by human or accepted and be able to be processed by a machine (computer). The accepting of a language is an important question such as solving a problem. From the beginning of computing theory, the concept of automata and formal languages theory contribute greatly to the foundation of programming language and compiler designs.**

Keywords—Computer Science, Finite Automata, Language Theory, Natural Language, Programming Language, Formal Language, Deterministic / Non-deterministic Automata, Language Processors, Compilers.

I. INTRODUCTION

In 1979, the topic of automata and language theory was little known to scientific community, was still an area of research and study. The topic was taught mainly at the graduate level across the universities. Today, the subject of automata is included in undergraduate curriculum. Technology improved rapidly that changed many things: the way we learn, the way we works, and even entertaining. Along with the demand in computer and high technology, the environment related to computer science has been growing fast to an unimaginable degree, and the number of courses in computer science curriculum has been also expanding in the past three decades.

Today, we still believe automata theory is very important tools for computer scientists in various new disciplines under the umbrella of computer science (CS) such as: Management Information Systems (MIS), Information Technology (IT), and Telecommunication and Networking (TELCOM). This paper will review the topic "Formal Language and Finite Automata" and illustrate some of its applications. [1]

II. LANGUAGES

II.1. Language Definition.

A language may contain an infinite number of strings defined on a finite set of symbols or alphabets ∑. Let ∑* be the set of all finite strings of symbols in ∑, including string of length zero (ϵ). So a language is a subset of ∑* for some alphabet ∑. This makes natural languages and programming languages included in this formal definition.

The types or classes of finite will provide useful properties of the language defined. If we call C is the class of the language defined specially with certain type of description, so we can answer whether or not membership in class C is preserved under various operations. Other answer is if a language in class C could be recognized simply and quickly, so we can develop a compiler for that particular language in class C. We also want to develop algorithms to process a given string s, to answer the question "Is string s in language L?" [2]

II.2. Formal Language Definition.

A formal language is an abstraction of general characteristics of programming languages that are processed by a computer. It is consisted with a set of symbols and some rules of formation or combination into strings named entities called words or program variables, and these entities can be grouped together called sentences or programming statements. So, formal language is a set of acceptable strings by the rules of formation. Any programming languages developed should have the same essential features as formal language.

II.3. Regular Language and Regular Grammar

II.3.1. Regular Expressions

One way to describe regular language is using the notation of regular expression. This notation includes a combination of symbols from the alphabet ∑, operators +, **.**, and *, and parentheses for grouping. For example, the language L_1 = {a} is denoted by the regular expression a. Another language L_2 = {a, b, c} and operator + as union (∪), we have another regular expression a + b + c. Similarly, we use operator (**.**) as concatenation, and * for star-closure then language L_3 = (a + (b**.**c))* means the star-closure of {a} ∪ {bc} Language L_3 can be expressed as: {λ, a, bc, aa, abc, bca, bcbc, aaa, aabc,.}.

II.3.2. Formal Definition of a Regular Expression

We construct regular expression by repeating applying certain recursive rules in a similar arithmetic expressions.

Definition:Let alphabet ∑ be given, then

1. Ø, λ and a ϵ ∑ are regular expressions and called primitive regular expressions.

2. If r_1 and r_2 are regular expressions then r_1+r_2, $r_1.r_2$, r_1*, and (r_1) are also regular expressions.
3. A string s is a regular expression if and only if it can be derived from the primitive regular expressions by applying a finite number of the rules in (number 2 above).

Example: Let ∑ = {a, b, c}, then the string s = (a+b+c)*.(c+Ø) is a regular expression, because it is constructed by the above rules.

Let r_1 = c, r_2 = Ø, we can see that c + Ø and (c + Ø) are also regular expressions.

a+b+c is a regular expression → (a+b+c) is a regular expression → so as (a+b+c)*

→ and finally (a+b+c)*.(c+Ø).[3]

II.3 3. Regular Grammars

In the context of natural language, the grammar is a set of rules for constructing or validating sentences of the language. As an example, consider the English sentence (statement).

The Students study automata theory

Sentence ==>Subject Verb Object (general form of a Sentence)

==> Article Noun Verb Object(replacing Subject)

==> The Noun Verb Object (replace Article by value The)

==> The students Verb Object(replace Noun by value students)

==> The students study Object(replace Verb by value study)

==> The students study Noun(replace Object)

==> The students study automata theory (replace Object by values)

Figure 1. Derivation of an English Sentence

II.3.3.1. Context-Free Grammars

A context free grammar should have the following sets.

.A set of non-terminal symbols (S, S_1, S_2,.)

. A set of terminal symbols (from the alphabet ∑)

.A set of rules

The function of grammar is to construct or to validate sentences of a language (is the sentence belong to the language or not?), in which many sentences can be generated or validated.

II.3.3.2. Definition

A grammar can be represented by a quadruple

$G = (N, \sum, P, S)$

Where:

1. N is a finite set of non-terminals.

2. ∑ is a finite set of terminals

3. $S \epsilon N$ is the start symbol

4. P is a finite subset of N x V* called the set of production rules.

$V = N \cup \sum$.

Example:

Let P = {*S* --> ab, *S* --> bb, *S* --> aba, *S* --> aab}, ∑ = {a, b}, *N* ={*S*}, Then grammar

$G = (N, \sum, P, S)$ is a context free grammar. Each production rule has the left hand side the start symbol *S* and on the other side is a terminal string.

1. S ==> ab
2. S ==> bb
3. S ==> aba
4. S ==> aab

Therefore, the language generated by *G*.

$L(G)$ = {ab, bb, aba, aab}. [4]

III. AUTOMATA

III.1. Automata Theory

In theoretical computer science, automata theory is the study of abstract machines and the computational problems that can be solved by these abstract machines. The word automata (plural form of automaton) comes from the Greek word "αὐτόματα" that means "Self acting". An automaton is a self acting computing device that follows a predefined sequence of operational instructions automatically.

Example of automaton.

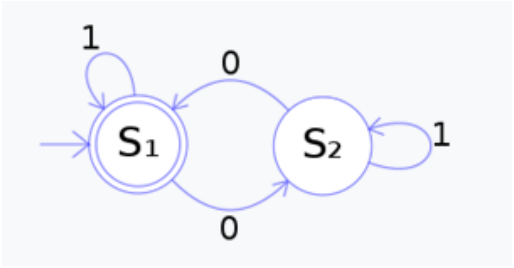

Figure 2

Figure 2 describes an automation in a finite state machine which consists of states (represented by circles) and transitions (arrows). When the automation reads (sees) an input symbol, it will make a change (transition) to another state, according to the transition functions which takes the previous state and the input symbol as arguments.

The automaton in figure 2 above described by its state diagram, starts in state S_1, then changes states to S_2 then back to S_1 following arrows labeled 0, or 1, according to the input symbols received. The double circle around S_1 indicates it is an accepting state. An automaton is an abstract self computing machine following a predefined sequence of instructions automatically. An automaton with a finite number of states is called a Finite Automation (FA) or Finite

State Machine (FSM). Automata are classified by the class of formal languages that they can recognize, and are useful in many applications: theory of computing, compilers design, artificial intelligence, parsing, and formal verifying. [5]

III.2. Finite Automata

Any regular languages such as natural or human languages and computer programming languages are strictly controlled, governed by regular grammars. We might say grammars as devices to generate regular expressions, and finite automata as accepting devices to process the languages according to some rules of grammar structures.

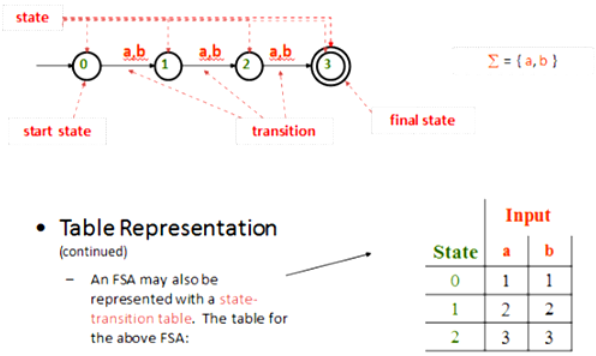

Figure 3

A Finite Automata (FA) as in figure 3 above is defined as:

- Finite number of states, one state must be initial state, and more than one or none can be the final states.
- Distinguishing rule. At each state, there must be an outgoing transition (to another state) for each input symbol in (symbols or alphabets) ∑.

As in figure 2 above, ∑ = {a, b}, and

states = 0, 1, 2, 3 where 0 is the initial state, 1 and 2 are states, and 3 is the final state.

Transitions:

At state 0 (initial state): reading a or b, go to state 1

At state 1: reading a or b, go to state 2

At state 2: reading a or b, go to state 3

At state 3: (final state – double circle)

Given an input string from the alphabet ∑, a finite automaton will accept or reject the input string based on the following:

- If the final state (state 3 as in the example above) is reached after reading the entire string, the finite automaton will accept the input string.
- If the final state is not reachable after reading the entire string, then the finite automaton will reject the string. [6]

IV. APPLICATIONS OF AUTOMATA THEORY AND FORMAL LANGUAGE

IV.1. Language Recognition / Processing

Automata theory is the fundamental basis for the formal languages. For any formal languages that include human languages and computer programming languages, there are some basic elements:

- A symbol is a character that is an abstract representation, meaningless by itself.
- An alphabet is a finite collection (a set) of symbols.
- A word is a finite character string of symbols from a given alphabet.
- A language is a set of words or character strings from a given alphabet.

The set of words of a language is infinite or normally unlimited, but it might also be finite or empty set ({ } or Ø). Formal language “sets” are considered as mathematical sets, so we can apply mathematical operations on sets theory such as Intersection (overlapping of sets) and Union (combination of sets but not repeating of the same elements). One important fact that sets operation (operation on languages) always produces a new language or set. Finally, sets or languages are defined and classified using techniques in automata theory.

Formal languages can be defined in one of three forms which can be described or recognized by automata theory: (1) regular expression. (2) standard automata. (3) or a formal grammar system.

- Regular expression samples

Alphabet A_1 = {a, b}

Alphabet A_2 = {1, 2}

Language L_1 = the set of all words

over A_1 = {a, ab, aab, abb,.}

Language L_2 = the set of all words

over A_2 = {1, 12, 112. 122,.}

Language $L_3 = L_1 \cup L_2$ = {a, ab,

aab, abb,., 1, 12, 112, 122,.}

Language L_4 = {a^n | n is even

number} = {aa, aaaa, aaaaaa,

aaaaaaaaa,.}

Language L_5 = {$a^n b^n$ | n is natural

number} = {ab, aabb, aaabbb,.}

- Standard automata.

Languages can be defined by standard automata (plural of automaton). Any automata or machine M operate on alphabet ∑ can generate a valid language L. Noam Chomsky, a computer scientist extended automata theory which led to the concept of formal grammar to define formal languages. The parameters of formal grammar are included:

1. A set of non-terminal symbols *N*
2. A set of terminal symbols ∑ (alphabet that constructs the language)
3. A set of production rules *P*
4. A start symbol *S*

G = (*N*, ∑, *P*, *S*)

- Grammar samples

Given:

Start symbol = *S*

Non-terminals = {*S*}

Terminals = {a, b}

Production rules: *S* --> a*S*b (rule 1),

S --> ba (rule 2)

S --> a*S*b --> abab (replace *S*

by rule 2)

S --> a*S*b --> aa*S*bb (replace *S*

by rule 1) --> aababb

(replace *S* by rule 2)

From the sample above, we can see mathematical automata can generate wide variety of complex languages with only few symbols and production rules. [7]

IV.2. Language Processors

IV.2.1. Compilers

Before a computer can execute (run) a program, it must be translated into a form that is executable by a computer. The program that does the translation is called a compiler. By definition, a compiler is a computer program that can read another program in one language (the source language or source code) then translate it into an equivalent program in another language (target language or object code or byte code in Java program). An important task of the compiler during the translation process, it will generate a list of errors it encountered to help the programmers in debugging errors in the source program (a program can be executed only when it is free of errors.)

IV.2.2. Interpreters

An interpreter is another type of language processor such as the old BASIC language created from Dartmouth College in the late 60s or early 70s. An interpreter will execute statements directly from a computer program, not generates target program like compilers. Because interpreter will take input then produces output immediately so it usually provide the feedback quickly including errors diagnosis for the programmer to fix.

IV.2.3. Hybrid Compiler

Java is the current most widespread and popular programming language. Java compiler combines compiling step and interpretation step to generate a special form called bytecodes. Then Java Virtual Machine (JVM) will interpret the bytecodes and produces the output (result). So Java programs can be run on any machines (multiple platforms). We might classify “Java types” compilers as hybrid compilers.

.

IV.2.4. A Language Processing System

Beside a compiler, several other programs components maybe needed to generate an executable target program (.exe file in machine language – binary). A source program might be stored in separated files. The task to collect these files is done by a program component called a preprocessor, and it also expanded the macros instructions in the source program.

Next step, the modified source program will be fetched into a compiler and it might produce an assembly language program as input to an assembler to generate a relocatable machine code. Another program component called the linker links all pieces of source program together (resolve external memory addresses). Finally, the loader will put together all executable object files in the memory for execution.

VI. CONCLUSION

In the theory of automata, formal languages are seen as a system for expression of certain ideas, facts, and concepts. This formalization covers varieties of human languages as well as computer programming languages. Generally, a language is a collection of sentences (or statements in programming languages), a sentence is a sequence of words, and a word is compounded of symbols of its alphabet.

The theory of each automaton will determine to accept a language or not, and this concept is very important in the designing of grammars that generate input strings belong to the languagein lexical, syntactic analysis phases of a compiler.

Theoretical computer science has many “big ideas” but it is sometime difficult to comprehend, and to grasp the subject matter fully can be tiresome. Learning a new subject such as “Automata” is a hard work, but when we get through, the subject matter becomes easier to understand and enjoyable [9]. This research paper is a stepping stone for students, readers to continue exploration into Automata and Theory of Computing world.

References:

[1]. Hopcroft, Motwani, Ullman. “Introduction to Automata Theory, Languages and

Computation”.2e, Addition-Wesley 2001. Pages: iii.

[2]. Alfred V Aho, Jeffrey D. Ullman. “The Theory of Languages”. Bell telephone Laboratories,

Inc. Murray Hill, New Jersey. Page: 97.

[3]. Peter Linz. “An Introduction to Formal Language and Automata, 5e, Jones and Bartlett

Publisher, Sudbury, MA 01776 (2012). Page 13. 78.

[4]. D. Goswami, K.V. Krishna, “Formal Language and Automata Theory”, Indian Institute of

Technology Guwahati (IIT Guwahati), November 5th, 2010, Pages: 18-23.

[5]. “Automata Theory”. https://en.wikipedia.org/wiki/Automata_theory.

Downloaded from the Internet, 06/01/2021 at 11:38am.

[6]. Dr. MM Alam, “Automata Theory” Electronic Government Research Center (EGRC),

COMSATS Institute of Information Technology, Islamabad, Pakistan. Pages 32-37

[7]. “Applications of Automata Theory”. https://cs.stanford.edu/people/eroberts/courses/soco/projects/2004-05/automata-theory/apps.html

Downloaded from the Internet 6/23/2025 at 12:48pm.

[8]. Aho, Lam, Sethi, Ullman, “Compilers: Principle, Techniques, and Tools”, 2e, Pearson

Addison Wesley (2007), pp. 1-4 83-87, 128-130, 159-173.

[9]. Michael Sipser, “Introduction to the Theory of Computation” 2e, 2006, Thomson Course

Technology, 25 Thomson Place, Boston MA. 02210. Pages: xi