Automata Theory - Lecture 10

Alphabets and Formal Language Definition

Lecturer: Martha Gichuki

Lecture learning outcomes

At the end of the lecture you will be able to:

(i) Describe Alphabets and Languages

(ii) Build formal language from alphabets and symbols

(iii) Describe various Regular operations

Introduction:

Automata theory is closely related to **formal language theory** because automatons are often classified by the class of formal languages, they are able to recognize.

An automaton is always anchored on the basic concepts of **symbols, words alphabets and strings.** These are defined briefly as follows: -

Symbol – A character or letter (An arbitrary datum) that has some meaning to or effect on the machine. Symbols are sometimes called letters.

Word – A finite string formed by the concatenation of a number of symbols.

Alphabet – A finite set of symbols. It is frequently denoted by Σ, which is the set of letters in an alphabet.

Language - refers to a set of words formed by symbols in a given alphabet.

Kleene Closure – A language may be thought of as a sub-set of possible words. The set of all possible words may in turn be thought of as the set of all possible concatenations of strings. Formally, this set of all possible strings is called a free monoid. It is denoted as * and the super script $^{(*)}$ is called the **Kleene Star**.

Alphabets & Languages

A language is simply a set of strings involving symbols from some alphabets. This general definition allows languages to be random assortments of unrelated strings as well as coherent highly structured languages such as natural languages or high-level programming languages.

In the case of high-level programming languages such as C or Pascal, the most reasonable way to think of a language as a set of strings is perhaps to take the strings to be complete legal programs in the language. We normally ask a compiler to check the syntax of our programs and sub-programs (debugging) rather than individual statements. If a language incorporates the rules or grammar that characterizes the language then the compiler reports no errors.

Therefore, an alphabet is a finite set of symbols. In the case of a common language like English, the alphabet contains 26 letters; UPPER CASE and lower case as well as blanks and various punctuation symbols. In the case of a programming language, ten numeric digits are added.

If $\sum$ is an alphabet, a string over $\sum$ is simply some number possibly zero, of elements of alphabets placed in order.

If $\sum$ = {a, b} is an alphabet, some of the strings over the alphabet are: -

(i) a

(ii) aba

(iii) aabba.

(iv) The null string denoted by $\wedge$ is the string with no symbols.

If string X is a string of an alphabet, the length of X is the number of symbols in the string and we denote this number by $|\mathbf{X}|$

Because languages are sets of strings, new languages can be constructed using set operations. For any two languages of an alphabet, their union, intersection and difference are also the languages of the alphabet.

Regular Expressions:

Regular expressions were designed to represent regular languages with a mathematical tool. This representation involves a combination of strings of symbols from some alphabet ∑. Recall that *a language is **regular** if some Finite Automaton recognizes it.*

Regular Operations on Regular Languages

In the previous sections we introduced and defined finite automata and regular languages. We now begin to investigate their properties. This will help us develop a tool box of techniques to use when designing automata to recognize particular languages. The toolbox also will include ways of proving that certain other languages are non-regular (i.e. beyond the capability of finite automata).

In arithmetic, the basic objects are numbers and the tools are the operations needed to manipulate them, e.g. the arithmetic expression: -

2 + 6 = 8 (2 & 6 are objects, while + is the tool that manipulates the two objects).

In the theory of computation, the objects are languages and the tools include operations specifically designed for manipulating them. We define three operations on languages called the regular operations and we use them to study properties of regular languages.

Let A and B be languages. We define the regular operations union, concatenation and star as follows: -

1. Union Operation

The operation simply takes all the strings in both A and B and lumps them together into one language.

$A \cup B = \{x \mid x \in A \text{ or } x \in B\}$

2. Concatenation Operation

The operation is a little trickier. It attaches a string from A in front of a string from B in all possible ways to get the strings in the new language.

$A \circ B = \{x\,y \mid x \in A \text{ and } y \in B\}$

3. Star Operation

The operation is a bit different from the other two because it applies to a single language rather than two. That is, the star operation is a unary operation instead of a binary operation. It works by attaching any number of strings in A together to get a string in the new language.

$A^* = \{x_1x_2x_3\ldots x_k \mid k \geq 0 \text{ and } x_i \in A\}$

Because "any number" includes 0 as a possibility, the empty string $\wedge$ is always a member of A*, no matter what A is.

Example:

Let the alphabet ∑ be the standard 26 letters {a, b, c…z}.

If A= {good, bad} and B= {boy, girl}, then: -

(i) Union Operation:

A∪B = {good, bad, boy, girl}

(ii) Concatenation Operation

A o B = {goodboy, goodgirl, badboy, badgirl} and

B o A = {boygood, girlgood, boybad, girlbad}

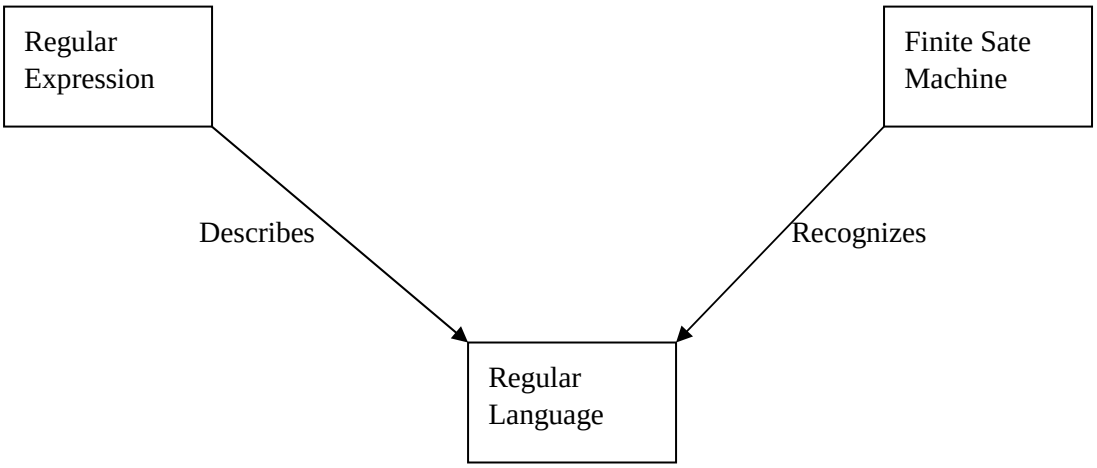
(iii) Star Operation

A* = {∧, good, bad, goodgood, goodbad, badgood, badbad, goodgoodgood, goodgoodbad, goodbadgood, goodbadbad…}

B* = {∧, boy, girl, boyboy, girlgirl, boygirl, girlboy, boyboyboy …}

Equivalence with Finite Automata

Regular expressions and finite automata are equivalent in their descriptive power. Any regular expression can be converted into a finite automaton that recognizes the language it describes and vice versa. A language is regular if and only if some regular expression describes it.



Proofing that a language is regular

In order to proof that a language is regular, we need to construct a Deterministic Finite Automata (DFA) for it.

Both of these automata can accept the same language. We can always construct some DFA M that accepts the same language as a given Non-deterministic Finite Automata (NFA) N.

A language is simply a set of strings that are made up of symbols from some alphabet. When we talk of the language of a machine, we mean that the machine recognizes that particular language.

Previously, we have seen that a Finite State Machine (FSM) either accepts or rejects strings. Therefore, a Finite State Machine defines a language. We say that a certain machine M recognizes language A if A contains all the strings that machine M accepts.

Definition of a language M

Let L(M) = be "Language L accepted by Machine M"

$\forall$ w, w$\in$L (M) then M accepts w

A language is regular if some DFA recognizes it in other words a DFA recognizes regular languages.

A regular language is one that can be constructed from the ***Big Three Operations*** i.e.

(i) Union

(ii) Concatenation

(iii) Star / Kleene Closure

L ⊆ ∑* is regular if and only if (iff) there exists ∃ a DFA M such that L= L(M). L ⊆ ∑* is regular iff ∃ | L= L(M)

Regular expressions are used to design regular languages e.g. L = {a, b} is a regular language over ∑ = {a, b} because both a and b are regular languages and they are expressions in L. Expressions are like the grammar of the language.

We can use regular operations (***union, concatenation and star***) to build expressions that describe languages. For example: -

Given (0∪1) * = {{0}, {1}}, we can perform the following operations on this language as follows: -

(i) *Concatenating* these we get {∈, 0, 1, 01, 10, 11, …} describes a certain NFA

(ii)∑*1 – Defines all strings ending with a 1

(iii) 0∑* - Defines all strings beginning with a 0 iv). {∑* 1∪ 0∑*} – Defines a union of the above two

References

1 Rowan G. & John T., (2009), *Discrete Mathematics: Proofs, Structures and Applications*, CRC Press, ISBN: 9781439812808.

2 W. D. Wallis (2003), *A Beginners Guide to Discrete Mathematics*, Springer Science & Business Media, ISBN: 978-0817642693.

3 Introduction to the theory of computation (3rd ed.), Michael, S. Boston, Cengage Learning. ISBN-13: 978-1133187790, (2012).

4 Introduction to languages and the theory of computation (3rd ed.), Martin, J., New York: McGraw-Hill. ISBN-13: 978-0072322002, (2002)