MODULE 4: THREE BASIC CONCEPTS OF AUTOMATA
WEEK 4

Learning Outcomes:

After completing this course you are expected to demonstrate the following:

1. Discuss and understand the three basic concepts: Language, Grammars and Automata

A. Engage

Recall the previous module to remind you about the topic. Ask some question that are related to the previous topic.

Example questions:

1. What are the four fundamental proof techniques?
2. What are the different between Definition into Proof?
3. What are the different steps for Proof of Contradiction?

B. Explore

Video Title: **Introduction to Grammars**
YouTube Link: **https://www.youtube.com/watch?v=sD9eIQRn6mk**
Module Video Title: **Week 4 - Introduction to Grammars**

C. Explain

Three fundamental ideas are the major themes of this module: languages, grammars and automata. In the course of our study we will explore many results about these concepts and about their relationships to each other. First, we must understand the meaning of the terms.

D. Elaborate

LANGUAGES, GRAMMARS, AUTOMATA: BASIC CONCEPTS

1. LANGUAGES

Strings: We already have the concept of a (finite) string over an alphabet A. We will always assume that the set A is finite, and that we are only concerned with strings of finite length. The alphabet is sometimes called V.

The fundamental operation by which strings are formed is the ***binary operation of concatenation***, which is just juxtaposition. If we concatenate the string abc with the string bbb, in that order, the result is the string abcbbb. Concatenation is sometimes written with a special symbol ∩ and sometimes written with no operation symbol at all, just using juxtaposition to represent itself.

Concatenation is **associative**. It is not in general commutative. (That is, it does not obey the commutative law, or commutativity axiom, although in certain examples it might "accidentally" happen that $\alpha\beta = \beta\alpha$.) The empty string is written e, and it forms an identity element for concatenation.

Given an alphabet A, the set of all strings over A (including e) is called A*. A*, with the operation of concatenation, is a monoid. (Why? Review the axioms for monoids!).

A handy aside (which occurs in various examples): the unary operation "reversal" on strings: x R. Informally, we can say that xR is just x written backwards. So, for instance, (abccd)R = dccba. The reversal of e is just e, and the reversal of any one-symbol string such as b is just b.

Given an alphabet A:

(1) If x is a string of length 0, then xR = x. (i.e. eR = e.)

(2) If x is a string of length k+1, then it is of the form wa, where w ∈A and a ∈A; then x R= (wa)R = awR.*

A good induction exercise: prove by mathematical induction on the length of strings that for all strings x and y, (x ∩ y)R = yR ∩ xR . This is question 3 of Homework 24 for next time.

Working with strings, we also make use of the notion of substring. Every string is trivially a substring of itself -- non-identical substrings are called proper substrings. An initial substring is sometimes called a prefix and a final substring a suffix

Def. (Used in "formal language theory") A language (over a vocabulary A) is any subset of A*

What is the cardinality of A*? What is the cardinality of the set of all languages over A? The set of formal devices we will consider for characterizing languages, namely formal grammars and automata, form denumerably infinite classes. So, there are languages without grammars. (Why does that follow?)

2. <u>GRAMMARS</u>

A ***formal grammar*** (and this is what we will normally mean by "grammar") is very much like a system of axioms and rules of inference, except that since a formal grammar is intended to simply generate a set of strings (a "language"), the axioms are not "interpreted" and we do not have any semantic side of the formal system. Nevertheless, we can think of a formal grammar very much as if it were a deductive system, and the sentences of the generated language as its "theorems".

Typically a formal grammar contains just one axiom, the string consisting of the initial symbol (usually S), and a finite number of rules of the form ψ → ω, where ψ and ω are strings, and the interpretation of a rule is the following: wherever ψ occurs as a substring of any given string, it may be replaced by ω to yield a new string.

Example: suppose a grammar contains the rule DP → Det NP. Then we could derive from the string DP V DP any of the following (by one or two applications of the rule): DP V Det NP; Det NP V DP; Det NP V Det NP.

Grammars use two alphabets, a ***terminal alphabet*** and a ***non-terminal alphabet***, which are normally required to be disjoint. The "output" of the grammar, which is the language it generates, is a set of strings of (over) the terminal alphabet. Intermediate strings that occur in derivations ("proofs") may contain symbols from both sets. The initial symbol or start symbol S is a non-terminal symbol.

Example:

(16-4) VT (the terminal alphabet) = {a,b}

VN (the non-terminal alphabet) = {S, A, B}

Initial symbol: S

R (the set of rules):

$S \rightarrow ABS$

$S \rightarrow e$

$AB \rightarrow BA$

$BA \rightarrow AB$

$A \rightarrow a$

$B \rightarrow b$

This sample grammar follows the common notational convention of using lower case letters for terminal symbols and upper case for non-terminals.

A derivation (one of many) of the string abba by this grammar:

S

ABS

ABABS

ABAB (because this is the same as ABABe)

ABBA

ABbA

aBbA

abbA

abba

Another way of writing a derivation is to write $S \Rightarrow ABS \Rightarrow ABABS \Rightarrow$….

This sequence is called a derivation (of abba from S), and the string abba is said to be generated by the grammar.

The set of all strings generated by a grammar G is the language generated by the grammar G.

AUTOMATA

An automaton is an abstract model of a digital computer. As such, every automaton includes some essential features. It has a mechanism for reading input. It will be assumed that the input is a string over a given alphabet, written on an input file, which the automaton can read but not change. The input file is divided into cells, each of which can hold one symbol. The input mechanism can read the input file from left to right, one symbol at a time. The input mechanism can also detect the end of the input string (by sensing an end-of-file condition). The automaton can produce output of some form. It may have a temporary storage device, consisting of an unlimited number of cells, each capable of holding a single symbol from an alphabet (not necessarily the same one as the input alphabet). The automaton can read and change the contents of the storage cells. Finally, the automaton has a control unit, which can be in any one of a finite number of internal states, and which can change state in some defined manner. Figure 4.1 shows a schematic representation of a general automation.

An automaton is assumed to operate in a discrete timeframe. At any given time, the control unit is in some internal state, and the input mechanism is scanning a particular symbol on the input file. The internal state of the control unit at the next time step is determined by the next-state or transition function. This transition function gives the next state in terms of the current state, the current input symbol, and the information currently in the temporary storage. During the transition from one-time interval to the next, output may be produced or the information in the temporary storage changed. The term configuration will be used to refer to a particular state of the control unit, input file, and temporary storage. The transition of the automaton from one configuration to the next will be called a move.

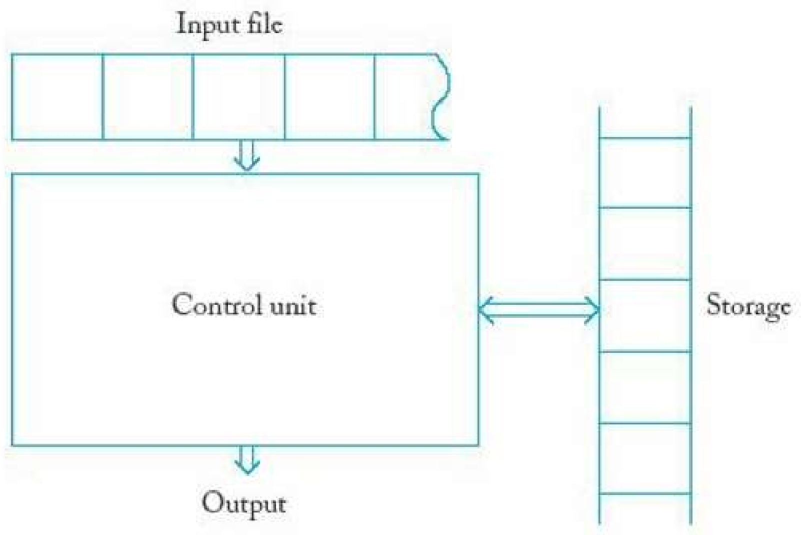

Figure 4.1

This general model covers all the automata we will discuss in this book. A finite-state control will be common to all specific cases, but differences will arise from the way in which the output can be produced and the nature of the temporary storage. As we will see, the nature of the temporary storage governs the power of different types of automata.

For subsequent discussions, it will be necessary to distinguish between ***deterministic automata*** and ***nondeterministic automata***. A deterministic automaton is one in which each move is uniquely determined by the current configuration. If we know the internal state, the input, and the contents of the temporary storage, we can predict the future behavior of the automaton exactly. In a nondeterministic automaton, this is not so. At each point, a nondeterministic automaton may have several possible moves, so we can only predict a set of possible actions. The relation between deterministic and nondeterministic automata of various types will play a significant role in our study.

An automaton whose output response is limited to a simple "yes" or "no" is called an ***accepter***. Presented with an input string, an accepter either accepts the string or rejects it. A more general automation, capable of producing strings of symbols as output, is called a ***transducer***.

E. Evaluate

ASSESSMENT:
Instructions: You may write your answer on the Activity Sheet (ACTS) provided in this module.

CONTENT FOR ASSESSMENT:
Activity Number 3 (20-points).

1. Justify if the language is acceptable or not. Provide the justification needed.

2. Let ∑ = {a, b}. Then

$$\Sigma^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}.$$

   The set

$$\{a, aa, aab\}$$

   is a language on ∑.

3. Let L = {ab, aa, baa}. Which of the following strings are in L*: abaabaaabaa, aaaabaaaa, baaaaabaaaab, baaaaabaa? Which strings are in L4?

References

1. *Automata, Grammars, and Languages: Introduction and Basic Concepts, Ling 409 Lecture Note, Partee, Lecture 24, Novemeber 23, 2005*
2. *An Introduction to Formal Languages and Automata, 5th edition, Peter Linz*

Facilitated By:		
Name	:	
MS Teams Account (email)	:	
Smart Phone Number	:	