# THEORY OF COMPUTATION, AUTOMATA AND LANGUAGES*

**R.O. AKINYEDE+ and O. FAJUYIGBE**
Department of Computer Science, Federal University of Technology, Akure, Nigeria

(*Recieved*: August 2007; *Accepted*: February 2008)

**Abstract**

Automata theory is the study of abstract machines and problems they could solve. History had it that A. Turing studied an abstract machine that possessed the capabilities of computers even before the advent of the first computer machine in 1930s. In 1940s, another machine known as finite automata, which would model human brain function, was proposed. In the same manner, in late 1950s other areas such as formal grammars, which have closed relationships to abstract automata, were studied. This research work critically studied the theory of computation, automata and languages. It also looked at problems that could be solved by computer and those that could in principle be solved.

**Key words**: Automata theory, deterministic and non-deterministic automata, pushdown automata, turing machine.

## 1. Background

According to Song (1998), automata are abstract (mathematical) machines that can read information from input and write information to output. Its finite state control unit controls this input/output process, Figure 1.

Automata theory, which is a mathematical discipline, is concerned with invention and study of mathematically abstract and idealized machines called automata. The theory proposes that human physical functions and behavior can be simulated by a mechanical or computer-controlled device. However, history had it that A. Turing studied an abstract machine that possessed the capabilities of computers even before the advent of the first computer machine in 1930s. In 1940s, another machine known as finite automata, which would model human brain function, was proposed. In the same manner, in late 1950s other areas such as formal grammars, which have closed relationships to abstract automata, were studied (Hopcroft *et al.*, 2001)

According to (Encarta, 2006), applications of automata theory have included imitating human comprehension and reasoning skills using computer programs, duplicating the function of muscles and tendons by hydraulic systems or electric motors, and reproducing sensory organs by electronic sensors such as smoke detectors. It also reported that the concept of automata, or manlike machines, has historically been associated with any self-operating machine, such as watches or clockwork songbirds driven by tiny springs and gears. But in the late 20th century, the science of robotics (the development of computer-controlled devices that move and manipulate objects) has replaced automata as it relates to replicating motion based on human anatomy. Modern theories of automata currently focus on reproducing human thought patterns and problem-solving abilities using artificial intelligence and other advanced computer-science techniques. Automata helps in the computation of partial functions from $P^*$ to $R^*$ for some alphabets P and R. Automata usually get inputs on a linear tape. In this paper, areas of formal grammars, which have closed relationships to abstract automata, were studied. The research critically studied the theory of computation, automata and languages. It also looked at the problems that could be solved by computer and those that could in principle be solved.

## 2. Deterministic Finite Automata (DFA)

Deterministic Finite Automaton (DFA), also known as Finite automaton, is a computing device that accepts regular languages and do not allow two-way operation of the tape head. The automaton, which is a deterministic state device equipped with a read head attached to a single input tape, operates by reading symbol, transfer to new instruction and advance the tape head one square to the right. This computing device reads its input tape and processes it. At times the input can be rejected depending upon the instructions being executed. Finite automaton can be represented in two ways, namely:- a transition table and transition diagram

(a). A Transition Table

+ *corresponding author* (email: femi_akinyede@yahoo.com)

* Presented in part at the First Faculty of Science Conference, Obafemi Awolowo University, Ile-Ife, July 3-5, 2007

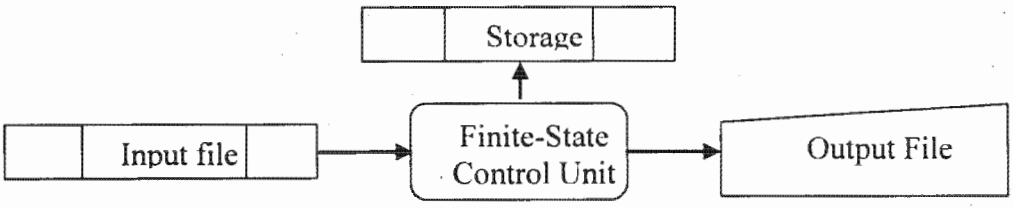


**Fig. 1**: A General Automaton

| State | 0 | 1 | Accept? |
|---|---|---|---|
| → $q_0$ | $q_0$ | $q_1$ | No |
| $q_1$ | $q_0$ | $q_2$ | Yes |
| $q_2$ | $q_2$ | $q_2$ | Yes |

**Table 1**: Transition Table

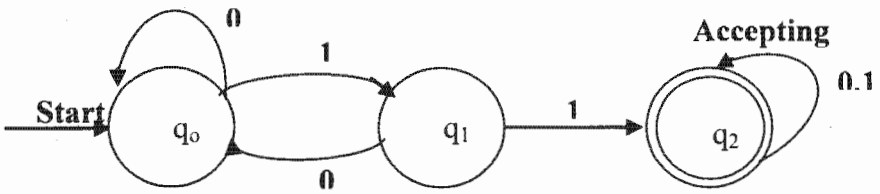


**Fig. 2**: Transition Graph

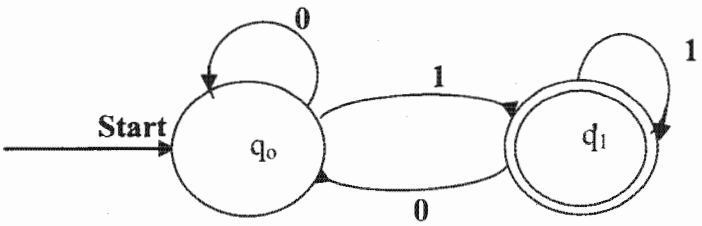


**Fig. 3**: Transition Diagram for DFA accepting all strings with a substring 01

| States | Inputs | |
|---|---|---|
| | 0 | 1 |
| →$q_0$ | $q_0$ | $q_1$ |
| $q_1$ | $q_0$ | $q_1$ |

**Table 2**: Transition table for the DFA of figure 3

A transition table is the table that shows the path as the automaton transits from state to state, initially from the start to the accepting state.

In the example given in Table 1, the machine will stay on instruction one until it reads a one and it goes on like that until it gets to instruction 3.

(b). A Transition Diagram or State Graph

Transition diagram or state graph is another method to describe finite automata. The states of the finite automaton appear as vertices of the graph while the transitions from state to state under inputs are the graph edges. The flow diagram shows the different finite states, the start state, which usually preceded by arrow, and the accepting state, which usually indicated by double circle. Once again, the number inside each circle is a name for the state it represents. Here is an example in Figure 2.

## 3. Formal Description of Automata

There are some concepts that are important to building automata theory and they are as follows:-

i. Symbol: is the input (datum) that is being read into the machine.

ii. Word: is the finite string formed by the combination of a number symbols.

iii. Language: is the finite word formed by the combination of number alphabets.

iv. Alphabet: is a finite set of symbols.

Finite automaton can further be described as a tuple $M = (Q, \Sigma, ä, q_0, F)$ where: Σ

$Q$ is a finite set (of states)

$\Sigma$ is a finite set of input symbols

ä next state or transition function,

that is $ä : Q \times \Sigma \rightarrow Q$

$q_0 \in Q$ is the start state, that is, the state in which the automaton is when no input has been processed.

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

(the accepting or final states).

Following the above definition, we can further define the entire machine by presenting a state graph. In the above example, we have $M = (\{q_1, q_2, q_3\}, \{0,1\}, ä, q_1, \{q_3\})$, where the transition function ä, is defined explicitly by a state graph.

For example, consider the transition diagram of the DFA as shown in Figure 3 below, draw the transition table.

In figure, we have that the start state is marked $q_0$ and the accepting state is marked $q_1$. The transition table is given in Table 2.

## 4. Non-Deterministic Finite Automaton (NFA)

As the name implies, it is state where the next state is not completely determined by the current state. This is contrary to the deterministic finite automaton, where every step taken has been exactly determined by the state of the machine and the symbol read. Figure 4, below provides the state graph of such a machine. Illustrating the Figure 4, if the inputs to this machine begin with the string 01, the movement will be from $q_0$ to $q_1$ to $q_2$. However, if symbol 0 is inputted, it will either move back to $q_1$ or to $q_4$. Finally, it will move to the state, which will eventually take it into a final state if possible (i.e. $q_1$ or $q_4$). From the above, we can definite NFA formally in the following way:-

Definition: A nondeterministic finite automaton is the quintuple

$A = (Q, \Sigma, ä, q_0, F)$ where Q, Σ, and F are as before but:

$Q_0$ is a set starting states, and

$ä\ (q, a) \subseteq ?S$ for each $q \in ?\ Q$ and $a \in ?\ \Sigma$.

From the above definition, we have that the NFA, instead of having a starting state and a transition function; we have a starting state set and a set of transition states. Meanwhile, the only difference between the DFA and the NFA is in the type of ä. For example, ä is a function that takes a state and input symbol as arguments in the NFA, but return a set of zero, one or more states while the DFA will take the a state and input symbol as arguments return only one state.

Example 2: Consider the Figure 5, where the NFA will accept all the strings of 0's and 1's that end in 01.

The above NFA can be specified as follows:-

$$M = (\{q_0, q_1, q_2\}, \{0,1\}, ä, q_0, \{q_2\})$$

| States | Inputs | |
|---|---|---|
| | 0 | 1 |
| ®$q_0$ | $\{q_0, q_1\}$ | $\{q_0\}$ |

Finally, non-deterministic finite automaton is allows us to define very simple machines, which perform certain operations.

For a given NFA G, there is a DFA $G^1$ which accepts the same language that the NFA accepts.

## 5. Regular Language and Regular Expression

(a). Regular Language

Song, (1998) defined regular language in the following ways:

i. $\phi$ is a regular language

ii. { } is a regular language

iii. {x} is a regular language if $x \in$

iv. $L_1 \cup L_2$ is a regular language if $L_1$ and $L_2$ are regular languages.

v. $L_1L_2$ is a regular language if $L_1$ and $L_2$ are regular languages.

vi. $L^*$ is a regular language if L is a regular language.

Example 3: Let = {a, b}. Then the following regular expressions represent the indicated sets of strings.

i. a: represents the set {a},

ii. $a^*$: represents the set $\{a\}^*$, = { λ , a, aa, aaa, ...}

iii. b: represents the set {b},

iv. ab: represents the set {a}{b} = {ab},

v. a ∪ b: represents the set {a} {b}= {a, b},

vi. $ab^*$: represents the set $\{ab\}^*$ = { , ab, abab, ababab, ...}

vii. $a^*$ ∪ $(ab)^*$: represents the set:

{a} $\{ab\}^*$= { , a, aa, aaa, ....ab, abab, ababab, ...}

viii. $a^*b$: represents the set $\{a\}^*\{b\}$={b, ab, aab, aaab, ...}

(b). Regular Expression

Regular expressions are another algebraic notation that describes exactly the same language as finite automata. Regular expressions are used to represent regular languages and their operations accurately. Their operators involve the following: union (+), concatenation (dot) and closure (star).

They are the set of regular expressions over an alphabet ∑, and defined by Song, (1998) as follows:

i. ϕ is a regular expression

ii. (empty string) is a regular expression

iii. x is a regular language if x ∈ ∑

iv. $r_1 \cup r_2$ is a regular expression if $r_1$ and $r_2$ are regular expressions.

v. $r_1 r_2$ is a regular expression if $r_1$ and $r_2$ are regular expressions.

vi. $r^*$ is a regular expression if r is a regular expression.

Example 4: Prove that if **A** $^*$ is a regular language, then the language

Prefix (**A**) = { ω ∈ $^*$ | x = ωy for some x ∈ **A** and y ∑$^*$} is also regular.

Solution: Let M = (Q, , , $q_0$, F) be a deterministic automaton which recognizes **A**. We construct the automaton $M^1$ = (Q, ∑, δ, $q_0$, $F^1$) where $F^1 \subset Q$ contains all the states of M that lie on a path between the start state and some accepting state. Let's prove that L($M^1$) =Prefix (**A**). Indeed, if ∈ L($M^1$), then reading w brings $M^1$ to some state $q^1$. But, by construction, there is a path from $q^1$ to some accepting state of M.

Suppose that the labels on this path construct the word x. Then ωx ∈ **A** and thus ω ∈ Prefix (**A**). Conversely, if ω ∈ Prefix (**A**), then there exist **x** $^*$, ∃ x ∈ **A**. But then, reading w brings M to some state q, starting at q aud reading x leads to some accepting state. But then q must be an accepting state of $M^1$, and so ω ∈ L($M^1$).

Equality of regular expressions: The regular expression $(p + q)^* = (p^*q^*)^*$ which represents the language of all strings over the alphabet {a, b} are said to be equal if and only if they correspond to the same language.

## 6. Pushdown Automata

A pushdown automaton is a (possibly nondeterministic) finite automaton with a special sort of auxiliary tape called a pushdown store (Ratson *et al.*, 2000). It makes use of a stack containing data; it is last in first out (LIFO), see Figure 6. The pushdown automata differ from normal finite state machines in these two ways, firstly, the top of the stack can be used to decide the transition to take and secondly, the stack can be manipulated as part of the transition. Pushdown automata recognizes a string x by the device as it gets into one of its final states. Given an input signal *p*, current state, and stack symbol, the automaton can follow transition to another state and optionally manipulate push or pop stack, but if a finite automaton is equipped with two stacks instead of just one, a more powerful device with Turing power is obtained. Pushdown automata can further be defined in the following ways:-

M = (Q, , ϕ, , s, , *F*) where:

Q is a finite set (of states)

∑ is a finite set of input symbols

ϕ is a finite set of the stack alphabet

is a finite transition relation.

(Q x (∑ ∪ { }) x ) P (Q x $^*$)

s is an element of Q the start state.

is the initial stack symbol

*F* is subset of states of Q. (The accepting or final states).

An element (p; a; M; q; α) of is called an instruction or transition of M. Finally, both stacks together can act as a working tape, and the machine can move both ways on that tape shifting the contents of one stack to the other by popping and pushing.

## 7. Turing Machines

History had it that a British mathematician, Alan Turing as a model of human computation, studied an abstract machine that possessed the capabilities of computers even before the advent of the first computer machine in 1930s (Hopcroft *et al.*, 2001). The machine, which was named after his inventor, was developed to eradicate the limitations posed by these three types of automata (finite, push down, and linear bounded automata). The machine was made to recognize all languages generated by phrase structured grammars. Turing machines can be likened to many present day computers except that computers have finite memory while Turing machines have infinite memory.

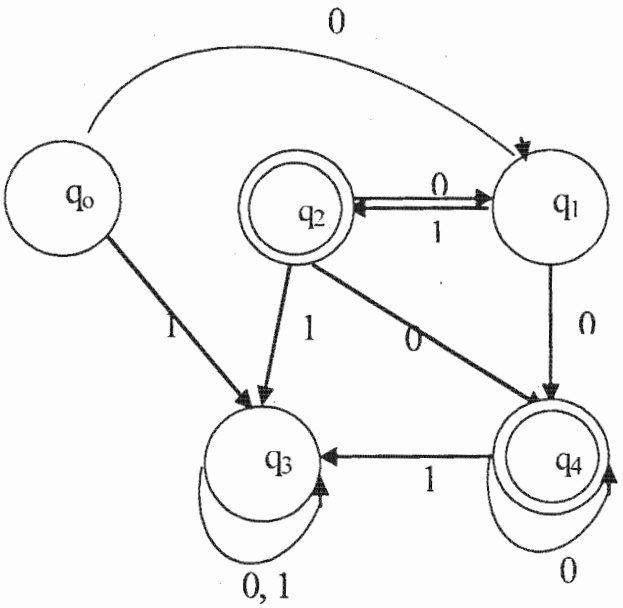


**Fig. 4**: Nondeterministic Finite Automaton

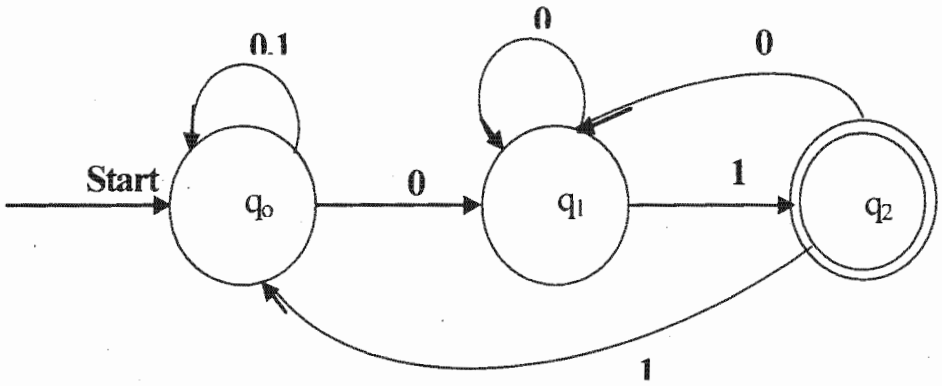


**Fig. 5**: The NFA accepting all strings that ends in 01

| States | Inputs | |
|---|---|---|
| | 0 | 1 |
| $\rightarrow q_0$ | $\{q_0, q_1\}$ | $\{q_0\}$ |
| $q_1$ | $\{q_1\}$ | $\{q_2\}$ |
| $*q_2$ | $\{q_1\}$ | $\{q_0\}$ |

* accepting state

**Table 3**: Transition table for the complete subset of figure 5

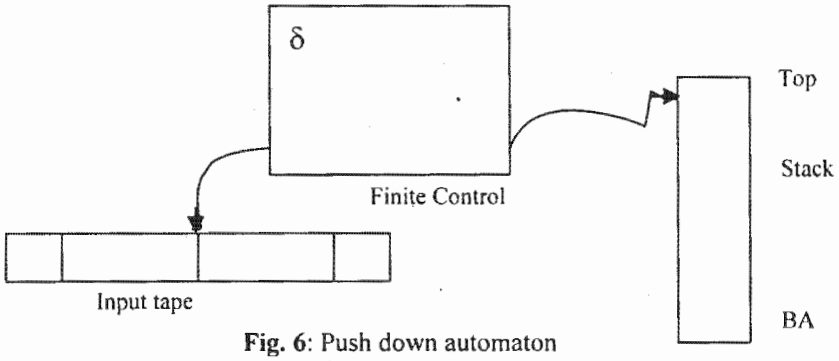


**Fig. 6**: Push down automaton

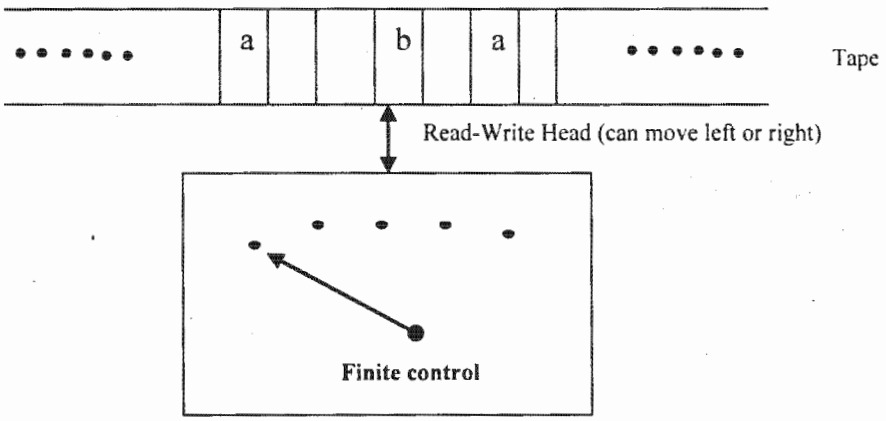


**Fig. 7**: A Turing Machine

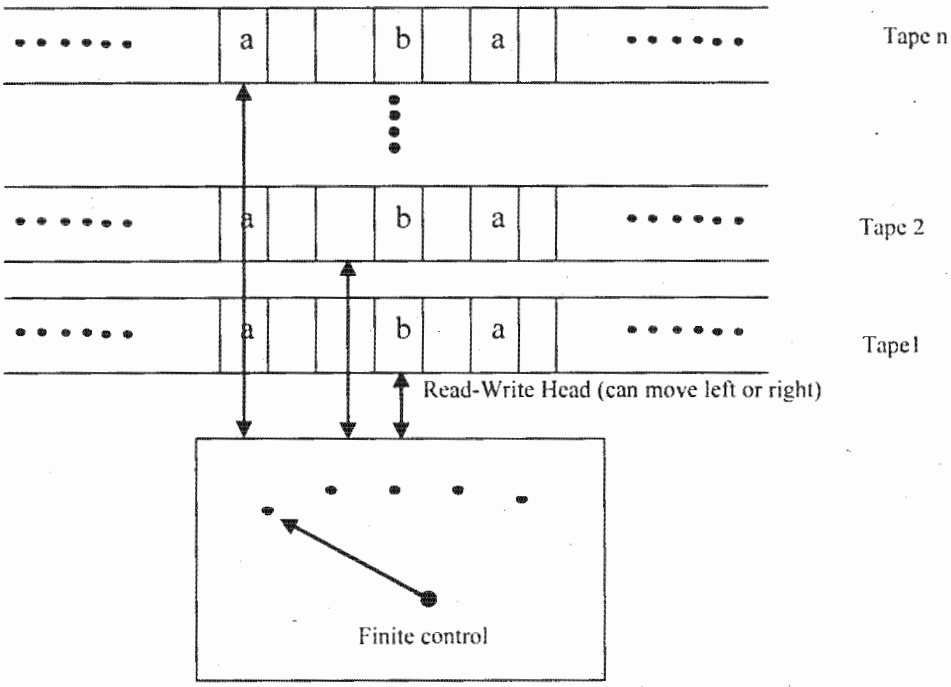


**Fig. 8**: A Multitape Turing Machine

However, a standard Turing machine, like finite automata, consists of a finite control and a tape; it is deterministic, and has no special input and output files. In Figure 7, we see that the tape has infinitely left and right ends. The tape is also divided into squares and a symbol can be written in them, but unlike finite automata, its head is a read-write head and it can move left, right or stay at the same square after a read or write action. However, when a string is input on the tape, a Turing machine starts at the initial state and at any state it reads the symbol under the head and then moves the head to left or right. But when it goes to the halt state, the Turing machine will stop its operation.

A Turing machine can be describe to have a 7-tuple $M = (Q, \Sigma, \Gamma, , q_0, \in, F)$, where

- Q is a finite set of internal states,
- $\Sigma$ is a finite set of symbols and it is the input alphabet.
- $\Gamma$ is a finite set of symbols called the tape alphabet.
- $q_0$ is the initial state.
- is the transition function, which is defined as f: $Q \times \Gamma^k \rightarrow Q \times \quad \times \{R, L, S\}^k$.
- $\in\in$ is a special symbol called blank.
- $q_0 \in Q$ is the initial state.
- $F \subseteq Q$ is the set of final state.

## 8. Equivalent Model of Computation

In the above, standard Turing machine was studied, but some Turing machine computation may require more tape than is available in the above one-tape deterministic. However, it is possible to extend one tape Turing machine to two or more tapes known as Multitape Turing machine.

The Multitape Turing machine is defined by a 7-tuple $M = (Q, , , , q_0, \in, F)$, where

- Q is a finite set of internal states,
- $\Sigma$ is a finite set of symbols and it is the input alphabet.
- $\Gamma$ is a finite set of symbols called the tape alphabet.
- $q_0$ is the initial state.
- is the transition function, which is defined as f: $Q \times \Gamma^k \rightarrow Q \times \quad \times \{R, L, S\}^k$.
- $\in\in$ is a special symbol called blank.
- $q_0 \in Q$ is the initial state.
- $F \subseteq Q$ is the set of final state.

## 9. Applications of Finite Automata

In computer science, automata theory is being applied in compiler construction. Pattern recognition, which is being handled by the lexical analyzer within the compiler, must be able to recognize which strings of symbols in the sources program should be considered as representations of objects like constants, variables and reserved words.

## 10. Decidable and Undecidable Problems

Decision problems are sets of questions whose answers are either yes or no. They are said to be decidable there is an algorithm such as Turing that solves the problem, or otherwise, it is said to be undecidable (Adleman, 1994). However, in computing, there are a lot of noncomputable functions, so also, are many decision problems that are undecidable.

Example 5: Let us consider a simple decision problem $P_{SQF}$ (as given by Song, (1998))of determining whether $p_n = 2^n - 1, \quad n \ N$ is a square-free (an integer n is square-free, denoted by $n \in SQF$, if it is not divisible by a perfect square):

$P_{SQF} = \{(p_n, yes) : p_n \in SQF) \cup (p_n, no) : p_n \notin SQF)\}$

$(p_n = 2^n - 1, \quad n \in Z^+)$

| Input | Output |
|---|---|
| $p_1 = 1$ | yes |
| $p_2 = 3$ | yes |
| $p_3 = 7$ | yes |
| $p_4 = 15 = 3.5$ | yes |
| $p_5 = 31$ | yes |
| $p_6 = 63 = 3^2.7$ | no |
| $p_7 = 127$ | yes |
| $p_8 = 255 = 3.5.17$ | yes |
| $p_9 = 511 = 7.73$ | yes |
| $p_{10} = 1023 = 3.11.31$ | yes |
| $p_{11} = 2047 = 23.89$ | yes |
| $p_{12} = 4095 = 3^2.5.7.13$ | no |
| $p_{13} = 8191 = 8191$ | yes |
| $p_{14} = 16383 = 3.43.127$ | yes |
| $p_{15} = 32767 = 7.31.151$ | yes |
| $p_{16} = 65535 = 3.5.17.257$ | yes |
| $p_{17} = 131071 = 131071$ | yes |
| $p_{18} = 262143 = 3^2.7.19.73$ | no |
| $p_{19} = 524287$ | yes |
| $p_{20} = 1048575 = 3.5^2.11.31.41$ | no |
| ..... | ... |
| ..... | ... |

A solution to a decision problem P is an algorithm that determines the appropriate answer to every question $p_n \in P$. It is clear that The Problem $P_{SQF}$ is decidable.

Nevertheless, many decision problems, for example, the halting problems(HP) for Turing machines is undecidable.

## 11. Success and Limitation of Automata

One of the automata technologies introduced is voice recognition. Voice recognition gives a computer the ability to understand spoken instructions. Now it is

possible for computers to understand conversation in the same way that human beings understand and this is possible by means of natural language. Expert systems, which partially mimic human specialist reasoning, are computer software programs that have two components. The components are a knowledge base and an inference engine. A knowledge base provides rules and data while an inference engine enables the expert system to form conclusions. The inference engine searches through the knowledge base and concludes which possible options are best solutions to the problems. However, the success of automata has been limited by programming techniques. According to Encarta (2006), it was reported that prior to the 1980s, almost all programming was done by structures designed for numerical processes such as calculating the sum of two numbers. New symbolic processes that use programming language, such as LISP, PROLOG, and C++, use symbolic logic, in which symbols represent the laws of reasoning. These languages and advances in programming techniques have stimulated new interest in automata theory. Despite these advances, progress of automata has been limited by modern computer technology in term of speed, storage, and application development of computer technology.

## 12. Conclusion

This work critically studied the theory of computation, automata and languages. It also examined the problems that could be solved by computer and those that could in principle be solved. Finite state automata are widely used in modeling of application behavior, network protocols, and the study of computation and languages. However, many other branches of science such as physics, biology and mathematics involve unbelievable levels of complexity that can be handled by automata theory. Therefore, more research should be encouraged in the area of automata theory, as this would contribute in no small way to a better scientific learning, which can lead to a technological advancement in Nigeria.

## REFERENCES

Adleman, L. M., 1994. *Molecular computation of solutions to combinatorial problems*, Science, 266, 11 November 1994, pp 1021-1024.

Chomspy, N., 1962. *Context free grammar and pushdown storage*. Quarterly Progress Report 65. Regular language induction with genetic programming Computation. IEEE Press. Volume I. Pages 396-400.

Encarta, 2006. *Microsoft Corporation*. 1993-2005.

Giles, C. L., Miller, C. B., Chen D., Chen, H. H., Sun, G. Z., and Lee, Y. C., 1992. *Learning and extracting finite state automata with second-order recurrent neural networks*. Neural Computation 4, 393-405.

Gold, E. M., 1967. *Language identification in the limit*, Inform. Contr. 10, pp. 447-474.

Gruau, F., 1995. *Genetic micro programming of neural networks*. In Pinnear, Penneth E. Jr. (ed.).

Hopcroft, J. Ullman, J., 1969. *Formal languages and their relation to automata*. Addison-Wessley, Reading, Mass., 1969.

Hopcroft, J. Ullman, J. 2001. *Introduction to automata theory, languages, and computation*. Reading, MA: Addison-Wesley, 2001.

Ratson, A., Reilly, E.D., and Hemmendinger, D., 2000. *Encyclopedia of computer science*. Nature Publishing Group 2000. 4th edition, 112-772.

Song Y. Y., 1998. *An introduction to formal languages and machine computation*. World Scientific Publishing Co. Pte. Ltd.