ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Issue 5, May 2014

Universal Turing Machine: A Model for all Computational Problems

Edward E. Ogheneovo

Lecturer I, Dept of Computer Science, University of Port Harcourt, Port Harcourt Nigeria.

ABSTRACT: Turing machines are the most powerful computational machines and are the theoretical basis for modern computers. Universal Turing machine works for all classes of languages including regular languages (Res), Context-free languages (CFLs), as well as recursively enumerable languages (RELs). In this paper, we discuss the concept of Universal Turing machine as a computing device that can be used for solving any problem that a computer or a human can solve. We show how the machine works in solving computational problems and design some algorithms showing to show the operational procedure of input symbols such as moving left, right, or stationary depending on the input symbol. Finally, we argue that the Universal Turing machine is a general-purpose machine that can be used to compute any computable problem.

KEYWORDS: Turing machines, Universal Turing machines, computation, computational model

1. INTRODUCTION

Computation has been around for a very long time. Alan Turing and others (Alonzo Church, Markov, Emil Post, Stephen Kleene, etc.,) independently considered the problem of specifying a system in which computation could be defined and studied. Turing focused on human computation and thought about the way humans compute things by hand. With this examination of human computation, he designed a system in which computation could be expressed and carried out. According to him, any trivial computation requires the following:

- A simple sequence of computing instructions
- Scrath paper
- A device for writing and erasing
- A reading device
- The ability to remember which instruction is being carried out

Turing then developed a mathematical description of a device processing comprising all these attributes. This device is called Turing machine, which is today referred as a special purpose computer. Turing machines are simple abstract computational devices intended to help investigate the extent and limitations of what can be computed. Turing machines are the most powerful computational machines and are the theoretical basis for modern computers [13], [15]. They possess an infinite memory in the form of a tape, and a head which can read and change the tape, and move in either direction along the tape or remain stationary. Universal Turing Machine (UTM) works for all classes of languages including regular languages, Context Free Languages (CFLs) as well as Recursively Enumerable Languages (RELs) [2], [7], [9]. That is, these languages are all acceptable by Turing machines.

The key property of Turing machines and all other equivalent models of computations is universality: there exists a Turing machine, T_u, capable of simulating any other Turing machine. This machine is referred to as Universal Turing machines T_u. Thus a Universal Turing Machine, T_u can be thought of as a Turing machine interpreter, written in the language of Turing machines. This capability of self-referencing is the source of the versatility of Turing machines and other models of computation. Thus T_u can simulate an arbitrary Turing machine on arbitrary input. The universal machine achieves this by reading both the description of the machine to be simulated and the input from its own tape. This universality of Turing machine makes it possible for it to solve or compute any problem that a computer can also compute [18]. Hence, the name universal Turing machine. Alan Turing introduced this machine in 1937.

Turing machines are equivalent to algorithms, and are the theoretical basis for modern computers. Yet it is often very difficult to create and maintain Turing machines for all problems. Doing this will consume a large amount of memory space. Moreso, the creation of TM_S for multiple tasks is very complex. This is the simple reason why a Universal Turing Machine (UTM) was designed. This Universal Turing machine is a machine that is able to simulate any other Turing machine, thus providing a single model and solution for all the computational problems [17]. A UTM is the abstract model for all computational models. A UTM (T_u) is an automation that, given as input the description of any Turing machine T_m, and a string, ω, can simulate the computation of T_m on ω. It reduces the memory usage when compared to using multiple TMs.

The transition function is the core part of a Universal Turing Machine. The UTM (T_u) works on the basis of the rules defined in it. The transition function σ for a T_u with single tape is defined as:

$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$$

The transition function δ is a partial function on $Q \times \Gamma$ and its interpretation gives the principle by which a Turing machine operates. The arguments of σ are the current state of the control unit and the current tape symbol being scanned. The result is a new state of the control unit, a new tape symbol which replaces the old one and a move symbol L or R **( )**.

Thus a Universal Turing Machine is a 7-tuple machine represented as:

$$T_u = (Q, \sum, \Gamma, \sigma, q_o, F)$$

where:	Q	= the set of all internal states
	$\sum$	= the input alphabet
	Γ	= finite set of symbols called the tape alphabet
	σ	= the transition function
	$q_0 \in Q$	= the initial state
	$\in \Gamma$	= a special symbol called blank
	$F \in Q$	= the set of all final states.

A UTM can solve any problem that can be solved using a FSA, PDA or even a Standard Turing Machine **( )**.

II. RELATED WORK

Jarvis and Lucas [10] implemented a Universal Turing Machine for the JFLAP platform. JFLAP is most successful and widely used tool for visualizing and simulating automata such as finite state machines, pushdown automata, and Turing Machines. They used JFLAP to simulate the Universal Turing Machine. Using their system, a user can get a direct and interactive experience of how this Turing Machine is capable of emulating other Turing Machines. This Universal Turing machine does not alter the portion of its memory that contains the list-of-final states of T_m and the list-of-transitions. Sumitha and Geddam [18] describes the implementation of Recursively Enumerable Languages (RELs) [8] using the Universal Turing machine for JFLAP platform. JFLAP is the most successful and widely used tool for visualizing and simulating all types of automata. They used the JFLAP to simulate many Turing machines and provide a single model and solution for all the computational problems.

Maheshwari and Dorairangaswamy [14] implement context-free languages in Universal Turing machines (UTMs) using JFLAP tool. They use JFLAP to simulate Turing machines to function as a UTM that acts as a single model and solution for all computational problems. Eberbach [6] proposed an evolutionary computation machine using a formal model of evolutionary computation. It investigated the convergence and convergence rate and provided sufficient conditions needed for the completeness and optimality of evolutionary search with particular reference to total optimality as an instance of the multi-objective optimization of the Universal Evolutionary Turing machine. The model was able to solve non-algorithmically the halting problem.

III. COMPUTATIONAL PROBLEMS

Computation is often defined formally using the mathematical concept of a Turing machine, a kind of computation whose operation is simple enough to be described with great precision. Such a machine needs to be powerful enough to perform any computation that a computer can, and Turing machines are known to be able to carry out any computation that current computers are capable of. Computation is any process that can be carried out by a computer [4]. It must be noted that computation as used here does not mean mathematical calculation such as the computation of the product of two numbers or the logarithm of a number. Computation as used here simply means includes all kinds of computer operations, including data manipulation, information storage and retrieval, etc [1]. It involves machine readability. That is, for a language to be machine readable, it must have a simple structure to allow for effective translation. First, there must be an algorithm to translate a language, that is, a step-by-step process that is unambiguous and finite. Secondly, the algorithm must not be too complexity [12].

Computable functions are functions that can be calculated using a mechanical calculation device given infinite amounts of time and storage space [3]. As stated earlier, Turing machines are very powerful. They can be used to compute any problem that is computable. That is, they can compute any problems that have effective procedure or algorithm that physical machine such as a computer can compute. Therefore, for a very large number of computational problems, it is possible to build a Turing machine that will be able to perform that computation. Turing's original paper is on computable numbers. According to him, a number is Turing-computable if there exist a

Turing machine which starting from a blank tape computes an arbitrarily precise approximation to that number [17]. Thus Turing machines can do more than just writing-down numbers. They can therefore also be used for computing numeric functions and any other computable functions.

According to Church-Turing thesis, any function which has an algorithm is computable [19]. The concept of algorithm as used here signifies a step-by-step solution to a problem such as a function as demonstrated in algorithm 1 and algorithm 2. There are many models of computation that have been proposed for solving functions that are computable. These computational models are, however, equivalent in terms of their power to compute functions. These computation models are:

- Turing machines proposed by Alan Turing in 1936.
- Lambda (λ) calculus proposed by Alonzo Church in 1936
- Post machine proposed by Emil Post in 1952
- Recursive function proposed by Stephen Kleene in 1952
- Markov Model proposed by A. Markov in 1943
- Register machines

Although these models use different representations for the functions, their inputs, outputs, and translations exist between any two models. Each computable function takes a fixed number of natural numbers as arguments. A function can be partial or total. A partial function is a function that is not defined for every possible choice of input [10, [16]. Thus a partial function is a function defined for a certain input and the resultant output is a single natural number. However, the output can be of interpreted as a list of numbers using functions that are paired.

It is possible to associate a partial function with each Turing machine. In this case, the input to the Turing machine is represented as an n-triple (e.g. $X_1, \ldots, X_n$). The integer represented by the maximal binary is separated by blank. This way, the Turing machine signs some bits or 0 if a blank is scanned when the machine halts. This is referred to as the output of the computation. This way, each Turing machine defines a partial function from n-tuples of integers onto the integers, $n \geq 1$. This type of function is said to be partial computable. However, for total computable functions, such functions are defined for all possible arguments. Thus, if the Turing machine halts for all inputs, then the function computed is defined for all arguments.

A function with the range (0, 1) is referred to as a predicate function. One major feature of a predicate function is that the predicate function has value TRUE or FALSE. The value is TRUE if the corresponding function has value 1 for some n-tuple of values for its arguments and is FALSE or undefined otherwise. For example, it is possible to design a Turing machine which when started on a tape representing a number terminates with TRUE on the tape if and only if the argument is a 1. Thus a Turing computable functions are recursive functions.

Consider X as a binary string. Thus for functions |X|, it is possible to see.

$$\left.\begin{aligned} f(x) &= \overline{x} \\ g(x) &= (\overline{x}\,y) \\ h(\overline{x}\,y) &= y \end{aligned}\right\} \quad \text{are all partial computable}$$

However, functions g and h are not total since the value for input 1111 is undefined. Thus the function $g^1(\overline{x}\,y)$ defined as 1 if x = y and as 0 if x ≠ y is a computable predicate. Also, if we consider x as an integer, then these functions are said to be total computable functions: the successor functions $g^{(1)}(x) = x+1$, the zero function $g^{(n)}(x_1, \ldots, x_n) = xm\ (1 \le m \le n)$.The function f(x, y) = $\overline{x}\,y$ is a total computable one-to-one mapping from pairs of natural numbers into the natural number. This scheme can easily be extended to obtain a total computable one-to-one mapping from k-triples of integers into the integers, for each fixed K. this can be defined as:

$(n_1, n_2, \ldots, n_k) = (n_1, (n_2, \ldots, n_k))$ or even

$(n_1, n_2, \ldots, n_k) = \overline{n_1} \ \ldots\ \overline{n_k}\ \text{-}\,1\,\overline{n_k}$ for k-tuples of integers

IV. FUNCTION THAT CAN BE COMPUTED

In the previous section, we showed that some functions are computable. Apart from functions, there are other problems that can be computed by Turing machines. In this section, we further discuss these problems, Turing machines can be used to compute arithmetic on natural numbers, encoding Turing machines. According to Alan Turing, a number is Turing computable if there exists a Turing machine which starting from a blank tape computes an arbitrarily precise approximation to that number. Computable numbers are the real numbers that can be computed to any desired precision by a finite, terminating algorithm. Turing defines computable numbers as sequences of digits interpreted as decimal fractions between 0 and 1. In a similar fashion, Minsky [15] defines a computable number as a number for which there is a Turing machine which, given a on its initial tape, terminates with the nth digit of that number.

A closer look at Minsky's definition shows that three important notions are worth noting: (1) that some n is specified at the start of the computation, (2) for any n the computation only takes a finite number of steps, after which the machine produces the desired result and then finally halt, and (3) that by using Turing machine, a finite definition in the form of machines table – is being used to define, what is potentially-infinite string of decimal digits.

Formally, we define a computable number as a real number *a* such that if it can be approximated by some function given any integer n, the function produces an integer are such there:

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Issue 5, May 2014

$$\frac{K-1}{n} \leq a \leq \frac{K+1}{n}$$

Using the Dedekind cut, we can define computable numbers as those number that when provided with a rational number r as input returns D(r) = TRUE or D(r) = FALSE where D is the Dedekind cut. Thus if D(r) = TRUE or D(r) = FALSE, then the following conditions are met.

$\exists$ r D(r) = TRUE
$\exists$ r D(r) = FALSE

(D(r) = TRUE) $\wedge$ (D(s) – FALSE) for r < s
D(r) = TRUE) $\wedge$ (D(s) = TRUE) for s > r

where s = irrational number

Therefore, a real number is computable if and only if there is a computable Dedekind cut D converging to it. The function D is unique for each irrational computable number. Finally, a complex number is said to be computable if its real and imaginary parts are computable.

V. CHARACTERISTICS OF COMPUTABLE FUNCTIONS

As stated earlier, every computable function must have an algorithm showing the step-by-step procedure of how a function can be computed. However, as seen earlier, many models of computation have been developed ranging from Turing machines μ–recursive functions, Lambda calculus, and a host of others. However, one interesting thing about these models is that even though they differ in their properties, they all give equivalent classes of computable functions in that they are all capable of solving the same problems. Thus every procedure or algorithm for computing a computable function has the following characteristics according to [5], [20].

i. There must be instructions (i.e. a program) finite in length, for the algorithm. The implication is that every computable function must have an algorithm that completely describes how the function is to be computed.
ii. If the algorithm is given a k-tuple *x* in the domain (i.e. set of all inputs for which the function is defined) of f, then after a finite number of discrete steps the procedure must terminate and produce f(*x*).
iii. If the procedure is given a k-tuple *x* which is not in the domain of f, then the procedure might go on forever, never halting; or it might get stuck at some point, but it must not pretend to produce a value for f at *x*. Thus if a value for f (*x*) is ever found, it must be the correct value.

VI. THE UNIVERSAL TURING MACHINE

The UTM has a tape infinite in both ends to hold the input and perform the computation. It also has a read/write head to position the input symbol an infinite memory [9], [11]. There are two other tapes also that are used for the

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Issue 5, May 2014

processing. The first tape holds the description of the original Turing machine T_m, and the other tape holds the internal state of T_m.

The input to the UTM, T_u, is given in the form $<T_m, \omega>$ where T_m is the Turing machine that is to be manipulated and ω is an input string for T_m. The execution of the Turing machine is specified by transition rules or delta rules. Each transition is of the form:

$$\delta (q_i, a) = (q_j, b, R)$$

where q_i = the initial state
a = the current read symbol
q_j = the next state final state
b = the write symbol
R = the direction to which the tape head has to move (i.e., left or right)

The tape head of the Turing machine and that of the UTM can move in either direct, move left specified by L or move right specified by R. The tape head can also decide to stay in its current position, this is specified by S. The encoded input is given to the UTM. The tape head scans the contents of the tape and reads the current input symbol and the current internal state. It checks the transition rules stored in the description tape and performs the operation as specified in the transition rule of the original T_m. When all the input symbols have been scanned, the T_u enters the final state, check section and performs the same as T_m. The internal structure of universal Turing machine is shown in the figure 1.

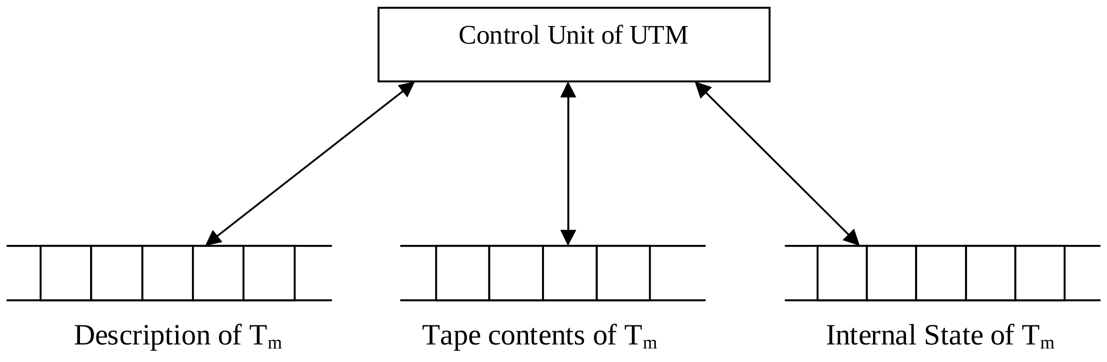

Fig. 1: A Universal Turing Machine

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Issue 5, May 2014

VII. UNIVERSAL TURING MACHINE'S ALGORITHM

Algorithm 1 shows the initialization of universal Turing machine. The algorithm is designed using the structured programming approach such as in Pascal programming language using the **while** , **for** and **repeat until** loops as the control structures.

Algorithm 1: Universal Turing machine, initialization

```
 1:  // Copy the transition function of M from the input tape to the description tape.
 2:  Move right past the first and second commas on the input tape.
 3:  while not reading ';' on input tape do
 4:        Read input symbol, copy to description tape, move right on both tapes
 5:  end while // end of while loop
 6:  // Write the identifiers of the halting states and the direction of which the input
 7:  // symbol moves
 8:  Move to start of input
 9:  Using binary addition subroutine, write m + n in binary on working tape
10:  for  i = 0, 1, 2, 3, 4, 5 do
11:        On special tape, write binary representation of m + n + I, followed by ';'
12:        // The value of m + n is stored in the working tape and not memory
13:  end for
14:  // Copy the input string x unto the working tape, ℓ, and separate the with commas
15:  Write m + n + 6 in binary on the state tape,  // to store the value of ℓ
16:  Starting from the left edge of input tape, move right until ';' is reached
17:  Move to left edge of working tape and state tape
18:  Move one step right on state tape
19:  repeat
20:        while state tape symbol is not ⊔
21:            Move right on input tape, working tape, and state tape
22:            Copy symbol from input tape to working tape
23:        end while // while loop ends
24:        Write ';' on working tape
25:        Rewind to left edge of state tape, then move one step right
26:  until input tape symbol is ⊔
27:  Copy m from input tape to state tape
28: // end initialization
```

The next algorithm, algorithm 2 is used for the main loop of the universal Turing machine as defined in the initialization algorithm 1. The algorithm shows the operation of Turing machine in terms of moving the input symbol left, right, or stay positional until the last input symbol is reached.

Algorithm2: Universal Turing machine, main loop.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Issue 5, May 2014

```
// Infinite loop. Each iteration simulations one step of $M$
Move to the left edge of all tapes
// First check if we need to halt
match ←TRUE
**repeat**
    Move right on special and state tapes
    **If** symbols do not match **then**
        match ← FALSE
    **end if**
**until** reaching ‘;’ on special tape
**if** match = TRUE **then**
    Enter “halt” state
**end if**
Repeat the above step two more time, with “yes”, “no”, in place of “halt”
Move back to left edge of description tape and state tape
**repeat**
    Move right on description tape to next occurrences of ‘(‘ or ⊔
    **match TRUE**
    **repeat**
        Move right on description and state tapes
        **if** symbols do not match **then**
            match ← FALSE
        **end if**
    **until** reaching ‘;’ on description tapes
    **repeat**
        Move right on description and working tapes
        **if** symbols do not match **then**
            match ← False
        **end if**
    **until** reaching ‘;’ on description tape
    **if** match = TRUE **then**
        Move to left edge of state tape
        Move left on working tape until ‘;’ is reached
    **end if**
    **repeat**
        Move right on description and state tapes
        Copy symbol from description to state tape
    **until** reaching ‘;’ on the description tape
    **repeat**
        Move right on description and working tapes
        Copy symbol from description to working tape
    **until** reaching ‘;’ on the description tape
    Move right on special tape past three occurrences of ‘;’, stop at the fourth
    match ← TRUE
    **repeat**
```

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Issue 5, May 2014

```
        Move right on description and special tapes
        **if** symbols do not match **then**
            match ←FALSE
        **end if**
    **until** reaching ‘;’ on special tape
    **if** match = TRUE **then**
        **repeat**
            Move left on working tape
        **until** reaching ‘;’ **or** ⊔
    **end if**
    match ←TRUE
    **repeat**
        Move right on description and special tapes
        **if** symbols do not match **then**
            match ← FALSE
        **end if**
    **until** reaching ‘;’ on special tape
    **if** match = TRUE **then**
        **repeat**
            Move right on working tape
        **until** reaching ‘;’ **or** ⊔
    **end if**
**until** reaching ⊔ on description tape
```

Algorithm 2 is used for the main loop of the universal Turing machine as defined in the initialization algorithm 1. The algorithm shows the operation of Turing machine in terms of moving the input symbol left, right, or stay positional until the last input symbol is reached. The algorithm describes the iterative simulation of universal Turing machine, T_u. The T_u starts reading the input symbol from the leftmost position of the of the tape content from a non-blank position. The tape header checks if there is any input symbol. If there is none, the machine halts. However, if the input symbol matches, the iteration process is repeated and the machine moves the tape header to the right until the end of the file and finally halts. The process is repeated two more times but this time around using the “yes” or “no” options in place of “halts”. The tape header the moves back to the left edge of the description tape and the state tape. This process is repeated until there are not more input symbols left and until reaching ⊔ on description tape.

VIII. CONCLUSION

In this paper, we have discussed the concept of Universal Turing machine and how it can be used to solve any problem that a computer can solve or any problem that is computable. Computable functions are functions that can be calculated using a mechanical calculation device given infinite amounts of time and storage space. As stated earlier, Turing machines are very powerful. They can be used to compute any problem that is computable. That is, they can compute any problems that have effective procedure or algorithm that physical machine such as a computer can compute. Therefore, for a very large number of computational problems, it is possible to build a Turing machine that will be able to perform that computation. Turing’s original paper is on computable numbers. Thus Turing machines can do more than just writing-down numbers. They can therefore also be used for computing numeric functions and any other computable function. The key property of Turing machines and all other equivalent models of computations is universality: there exists a Turing machine, T_u, capable of simulating any other Turing machine.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Issue 5, May 2014

This machine is referred to as Universal Turing machines T_u. Thus a Universal Turing Machine, T_u can be thought of as a Turing machine interpreter, written in the language of Turing machines. This capability of self-referencing is the source of the versatility of Turing machines and other models of computation. Thus T_u can simulate an arbitrary Turing machine on arbitrary input. The universal machine achieves this by reading both the description of the machine to be simulated and the input from its own tape. This universality of Turing machine makes it possible for it to solve or compute any problem that a computer can also compute.

XI. REFERENCES

1. Arora, S. and Barak B. (2009). Computational Complexity: A Modern Approach, Cambridge University Press.
2. Bassey, P. C., Asoquo, D. E., and Akpan, I. O. (2010). Undecidability of the Halting Problem for Recursively Enumerable Sets, World Journal of Applied Science and Technology, Vol. 2, No. 1, ISSN: 2141 – 3290, pp. 41-48.
3. Boolos, G. S. and Jeffrey, R. C. (1974). Computability and Logic, Cambridge: Cambridge University Press.
4. Davis, M. (1982). Computability and Unsolvability, New York: Mcgraw-Hill.
5. Enderton, H. (1977). Elements of Recursion Theory. Handbook of Mathematical Logic, Edited by Barwise, North-Holland (1977), pp. 527-566.
6. Eberbach, E. (2005). Towards a Theory of Evolutionary Computation, BioSystems, Vol. 82., pp. 1-19.
7. Gramond, E. and Rodger, S. H. (1999). Using JFLAP to Interact with Theorems in Automata Theory. In Proceedings of the SIGCSE, ACM, pp. 236-340.
8. Herken, R., (ed.) (1988). The Universal Turing Machine: A Half-Century Survey, New York, Oxford University Press.
9. Hopcroft, J., Motwani, R. and Ullman, J. (2006). Introduction to Automata Theory, Languages, and Computation, 3rd Edition, Addison-Wesley.
10. Jarvis, J. and Lucas, J. M. (2008). Understanding the Universal Turing Machine: An Implementation in JFLAp, Journal of ACM, Vol. 23, Issue 5, pp. 180-188.
11. Kleene, S. C. (1936). General Recursive Function of Natural Numbers, Mathematics Annalen, Vol. 112, pp. 727-742.
12. Lewis, H. R. and Papadimitrinu, C. H. (1981). Elements of the Theory of Computation, Englewood Cliffs, N. S. Prentice-Hall.
13. Lin, S. and Radó, T. (1965). Computer Studies of Turing Machine Problems, Journal of ACM, Vol. 12, pp. 196-212.
14. Maheshwari and Dorairangaswamy (2011). Implementation of Context-free Languages in Universal Turing Machines. Int'l Conference on Electronic Computer Technology, Vol. 5, pp 332-335, April 8 – 10, 2011.
15. Minsky, M. (1967). Computation: Finite and Infinite Machines, Prentice-Hall, Inc., N. J., 1967.
16. Petrzold, G. (2008). The Annotated Turing: A Guided Tour through Alan Turing's Historic Paper on Computability and Turing Machines, Indianapolis, Indiana, Wiley Publisher
17. Radó, T. (1962). On Non-Computable Numbers, with an Application to the Etscheidungsproblem. In Proceedings of London Mathematical Society, Ser. 2 , Vol. 42, pp. 230-265.
18. Sumitha, C. H. and Geddam, K. O. (2011). Implementation of Recursive Enumerable Languages in Universal Turing Machine. Int'l journal of Computer Theory and Engineering, Vol. 3, No. 1, 1793-8201, pp. 153-157.
19. Turing, A. M. (1936). On Computable Numbers with Application to the Etscheidungsproblem. In Proceedings of London Mathematical Society, 42, pp. 230-265; correction in 43 (1937), pp. 544-546; reprinted in [Davis, 1965, pp. 115-154]
20. Turing, A. M. (1937). Computability and λ-Definability. The Journal of Symbolic Logic, Vol. 2, pp. 153-163.

BIOGRAPHY

Edward E. Ogheneovo is a Lecturer in the Computer Science Department of the Faculty of Physical Science and Information Technology at the University of Port Harcourt, Rivers State, Nigeria. He received his B.Sc. in Computer Science/Mathematics, and Ph. D in Computer Science from the University of Port Harcourt; while he received his M.Tech. in Information Technology from the Federal University of Technology, Akure. Dr. Ogheneovo is also a certified Oracle10g and 11i Database Professional. His research interests are in the area of computer security, programming, software dependability, and database management. He has published several articles in Learned Journals both in Nigeria and Internationally.