Digital Innovations & Contemporary Research in SCIENCE & ENGINEERING

**The Digital Innovations & Contemporary Research in Science & Engineering Journal** is an International Journal of
The Multidisciplinary Research Group
International Centre for IT & Development, USA

**Article Citation Format**
E.E. Ogheneovo (2016) The Limits of Turing Machine as a Computational Model. Journal of Digital Innovations & Contemp Res. In Sc., & Eng Vol. 4, No. 4. Pp 1-12

**Article Progress Time Stamps**

**Article Type:** Research Article
**Manuscript Received:** 13rd September, 2016
**Review Type:** Blind
**Word Count Post Review:** 6107
**Review/Acceptance Information Sent** : 16th Dec, 2016
**Final Acceptance:**: 31st Dec, 2016
**DOI Prefix:** 10.22624

# The Limits of Turing Machine as a Computational Model

**E. E. Ogheneovo**
Department of Computer Science
University of Port Harcourt
Port Harcourt, Nigeria.
E-mail: edward_ogheneovo@yahoo.com
Phone: +2348038591480

## ABSTRACT

The general belief among computer users is that there is no problem or function that a computer cannot solve if giving enough time and space in the computer memory. Since Turing machines and by extension computer was born knowing its limitations, there are still some problems or functions that computer cannot solve. These functions are said to be non-computable. By uncomputability we mean certain natural computational problems that cannot be solved by universal Turing machine or by computer programs. This leap from the notion of universality to impossibility is based on two main factors: 1) the difficulty in determining the "ultimate" behavior of a program; and 2) the self-referential character of the universal Turing machine. This paper discusses the limits to the power of Turing machine as a computing device with regards to what it can compute and what it cannot compute, that is, problems that can be solved and those that defy solution by computer and any known algorithm. Finally, the paper argues that there are still numerous problems that their algorithms cannot be designed; such problems are therefore said to be uncomputable or undecidable.

**Keywords:** Turing machine, computability, non-computability, computer programs & computation

## 1. INTRODUCTION

The quest by scientists and humanity in general to understand nature and to master it has led to several questions and numerous efforts to build artificial intelligence machines in order to provide answers to these questions. One such machine is the Turing machine proposed by Alan Turing in 1936 [1]. Since then, the Turing machine has been a standard acceptable model of computation. However, ever since the Turing machine was proposed, several questions have been raised by mathematicians, computer scientists, and physicist as to the computational power and dynamic behavior of the machine [2]. Turing machines have three main properties that model algorithmic computation: (1) it has closed computation; (2) their recourses (time and memory) are finite; and (3) they have fixed behavior (i.e., all computations starts in the same configuration) [3] [4].

Since computer science was born knowing its limitations, the strength of the universal machine (i.e., Turing machine) leads directly to a negative consequence of uncomputability [5] [6] [7]. By uncomputability we mean certain natural computational problems that cannot be solved by Turing machines or, by extension computer programs. This leap from the notion of universality to impossibility is rooted in two basic issues: 1) the difficulty in determining the "ultimate" behavior of a program; and 2) the self-referential character of the universal Turing machine T [8]. In the first case, recall that our universal machine $T_u$ simply performed a step-by-step simulation of a Turing machine $T_m$ on an input *n*. This means that if $T_m$ computes forever, without halting, then $T_m$'s simulation will run forever as well. This phenomenon is often referred to as "infinite loop." A situation whereby a machine or program run forever without producing desired result—a program keeps running without producing any result or output [9]. This situation exists if we use GO TO statement in a program if there is no other line of code or statement to halt the program

The major property of Turing machines and all other equivalent models of computations is universality. That is, there exists a Turing machine, $T_u$, that can simulate any other Turing machine. Turing machine is very versatile in that it can reference itself. This ability of self-referencing is the main reason that makes Turing machines and other models versatile for computing [10]. 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 inputs 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 [11]. 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, hence, a Turing machine is limited in its computational capabilities [12] [13].

## 2. DEFINING ALGORITHM AND TURING MACHINES

The concept of algorithm was used in building Turing machines [14] [15]. Various definitions of algorithm abound. However, we consider a few in this paper including the one by Knuth. In defining algorithm, we consider these definitions:

**Definition 1:** An algorithm is a step-by-step technique for solving a problem.
**Definition 2:** An algorithm is a self-contained step-by-step set of operations to be performed in solving a problem.
**Definition 3:** An algorithm is a finite set of instructions that, if followed, accomplishes a particular task.

**Definition 4:** An algorithm is an effective technique expressed as a finite list of well-defined instructions for calculating a function.

From these definitions, it is very clear that there are three main properties that model algorithmic computation or Turing machines: (1) it has closed computation; (2) their recourses (time and memory) are finite; and (3) all computations start in the same configuration [16]. All these properties are also exhibited by an algorithm. From these definitions, three common properties of all algorithms are as stated earlier. That is, those properties listed as the properties of Turing machines.

## 2.1 Describing Turing Machines

A Turing machine TM is specified by a finite alphabet $\sum$, a finite set of state K with a special element s (the starting state),and a transition function $\sigma : K \times \sum \rightarrow \{K \cup \{yes, no, halt\}) \times \sum \times \{\leftarrow, \rightarrow, -\}$. It is assumed that $\sum$, K, {yes, no, halt}, and $\{\leftarrow, \rightarrow, -\}$ are disjoint sets, and that $\sum$ contains two special elements $\rhd$, $\sqcup$ representing the start and end of the tape, respectively. Turing machines are simple, abstract computational devices intended to help investigate the extent and limitations of what can be computed [17]. They are similar to finite-state automata and pushdown automaton except that they are more powerful in their computational capabilities than these other computing devices [18]. Both have a finite-state machine as a central component; both have additional storage. However, the major differences between them are: (1) a pushdown automaton uses a stack for storage, a Turing machine uses a tape that may be infinite at both ends or bounded at one end; (2) the read/write head of a pushdown automaton allows it to move in only one direction—left to right, whereas in a Turing machine, the read/write head allows it to move in either direction (left-to-right or right-to-left) thus allowing a Turing machine to access items in an arbitrary manner; (3) Turing machines receive their input written on the same tape which they also use for storage; and (4) Turing machines control the head position to where reading and writing on the tape is performed [19] [20].

Also, a Turing machine differs from a finite automaton in a number of ways:

- A Turing machine can both write on the tape and read from it
- A read/write head can move in both directions, that is, to the left and to the right
- The tape is infinite
- The special state for rejecting and accepting take effect immediately.

However, unlike the other automata we have considered so far, a Turing machine does not read "input". Instead, there are usually symbols on the tape before the Turing machine begins; the Turing machine can decide to read some, all, or none of these symbols [21] [22]. As said earlier, a Turing machine is an automaton with random—access memory. They differ from other automata we considered earlier in that they have an infinite memory—infinite in both directions.

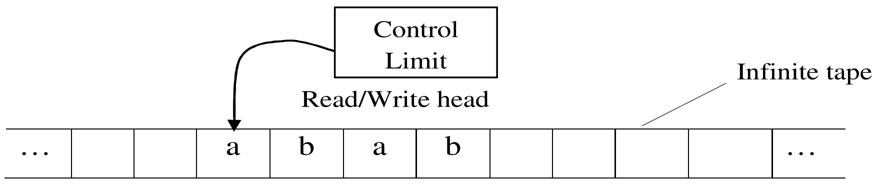

**Fig 1: A typical Turing machine**

Figure 1 shows a typical Turing machine. As seen in the figure, a Turing machine consists of three components, namely:

- **Tape:** The tape consists of an infinite set of cells, each of which can hold exactly one symbol. The tape acts as the memory of a Turing machine
- **Control Unit:** This part consists of an enhanced form of finite state machine, and is used for processing of symbols.
- **Read/Write Head:** It can also be simply referred to as head. It mediates between the tape and the control unit. The control unit component interacts with the tape by reading from the current position of the head and also writing to the current position on the tape. The control unit can also make the head move to the nearest cell (left or right) of the tape.

Depending on the current state the control unit is in, and depending on the tape contents at the location (cell) where the head is, a Turing machine then determine:
(i) Its next state
(ii) How to interact with the tape. This can be done in either of three (3) ways:

- Write a symbol on the tape and remain in the same position (i.e., not moving)
- Move one cell to the right
- Move one cell to the left

As we know it today, Turing machines form the basis or foundation of modern theoretical computer science based on Church-Turing Thesis which state that a function is computable in the intuitive sense if and only if it is Turing computable. That is, is a function is computable in the intuitive sense, then it is a Turing machine that computes it or it is Turing-computable [23].

## 2.2 Computation Defined

Several researchers have variously defined computation in different ways. In the early 1930s at about the period Turing proposed his machine, the Turing machine, Kurt Gödel (1934), Alonzo Church (1934), Emil Post (1936), and Alan Turing (1937) independently defined computation [24]. Gödel defined computation as the evaluation of recursive functions; Church defined it in terms of lambda calculus, that is, the evaluation of lambda expressions; Post defined computation as series of strings successively rewritten based on a given set of rules; and Turing defined it as the sequence of states of an abstract machine with a control unit and a tape [25]. However, the generally acceptable definition at the time which was based on the concept of Turing machine is "the execution of sequence of halting Turing machines or their equivalent." Such a machine must provide an answer (yes or no) or halt. Thus if the machine halts, it means that such a function or problem being solved has no solution and is therefore said to be non-computable or undecidable [26] [27].

Formally, we define computation using the mathematical concept of Turing machine as any process that can be carried out by a computer [28]. The formal definition of computation is as follows:

**Definition:** Let $M$ = Q, $\sum$, $\sigma$, $q_o$, F) be a finite automata and w= $w_1w_2 \ldots w_n$ be a string over $\sum$. Then $M$ accepts w if a sequence of states $r_o$, $r_1$, …,$r_n$ exist in Q such that the following holds:

i. $r_o = q_{o,}$ i.e., the machine starts its computation in the start state;
ii. $\sigma(r_i, w_{i+1}) = r_{i+1}$ for i = 0, 1, …, n-1, i.e., as long as input is available the machine goes from state to state according to its transition function $\sigma$
iii. $r_n \in$ F, i.e., the machine accepts its input if it ends up in an accept state.

Since computation is defined based on the principles of Turing machine, it suffices to say that such a machine must be powerful enough to perform any computation that a computer can compute. Turing machines are known to be able to carry out any computation that current computers are capable of. 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 all kinds of computer operations, including data manipulation, information storage and retrieval, etc [29]. 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 complex [30].

### 2.2 Computational Problems

Computable functions are functions that can be calculated using a mechanical calculating device given infinite amounts of time and storage space [31]. 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 problem that has 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. According to Alan Turing, 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 [32]. 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. As noted by Milner [33], Turing machine is capable of solving natural, interactive, and continuous information processes often seen today.

According to Church-Turing thesis, any function which has an algorithm is computable [34] [35]. 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 computers. 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. However, the output can be of interpreted as a list of numbers using functions that are paired. We can 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. $X_1$, ..., $X_n$). 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, $n \geq 1$.

## 3. COMPUTABLE FUNCTIONS

A function is computable if there exist an algorithm that can be designed to solve such a function. By computable function, we mean, given an input of the function domain, it can return the expected output. That is, a function is said to be computable if for a given input its output can be calculated by a finite mechanical procedure. Computable functions are also called recursive functions.

**Definition:** A function is computable or effectively calculable or simply calculable if it can be calculated by a finite mechanical procedure.

A natural definition of a computable function f on $\mathbb{N}$ allows for the possibility that f(x) may not be defined for all $x \in \mathbb{N}$, because algorithms do not always halt. According to Alan Turing, a number is Turing computable if there exist a Turing machine that is capable of computing an arbitrarily precise approximation to that number. Computable numbers are the real numbers that can be computed to any desired precision by an algorithm. Turing defines computable numbers as sequences of digits interpreted as decimal fractions between 0 and 1. In a similar fashion, Minsky [36] defines a computable number as a number for which there is a Turing machine which, given $a$ on its initial tape, terminates with the n digits of that number. A closer look at Minsky's definition shows three important notions which 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, (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.

## 4. UNCOMPUTABLE PROBLEMS

Influenced by Gödel Incompleteness theorems, Church, Post, and Turing later saw that there are certain functions or problems that could not be computed by any known algorithms. In fact, it was Turing himself who discovered that his own machine is incapable of solving all problems after he initially claimed that his universal Turing machine is capable of solving all problems even though his machine can be simulated to any other machine. Such problems are said to be uncomputable or undecidable. These problems include: Entscheidungsproblem, the halting problem, NP-complete problems, Post correspondence problem, Gödel Incompleteness theorems, combinatorial problems such as molecular computing and DNA-interconnection processes in nature, cellular automata, quantum mechanics, and many other natural phenomenon [37].

Addleman [38] in his study on NP-complete, combinatorial problems, and molecular computation uses both the Hamiltonian path problem using DNA computing. Addleman uses DNA computing by encoding vertices and edges in DNA such that a large amount of random paths is formed in parallel. He found out to his surprise that in every path he considered using DNA code, the quantity of the DNA needed growth exponentially with the number of nodes, n. Thus it was observed that exponential complexity is not eliminated but only shifted from time to space. Addleman concludes that it is possible to carry out computations at molecular level. However, what he fails to tell us is whether the same success can be achieved if the same approach is used to solve large instances of the same problem. This is because as the number of nodes increases, the polynomial time and space increases exponentially and solution to the problem becomes more and more unfeasible.

Wolfram [39] discussed the concept of cellular automata using computation complexities. A cellular automaton is a machine that has cells in form of grids, each of which have a finite number of states such as on (1) and off (0). These grids usually have a finite size. In other words, the cellular automaton is a computer model consisting of memory locations, each having one bit. These bits are updated from time to time at regular intervals. Wolfram compared the cellular automata to the universal computing machine, the Turing machine, lambda calculus by Church and many other computational machines. However, since the cellular automaton is in many ways similar to the universal Turing machine, it is believed that it is limited in its computational capabilities and it may not be capable of computing certain problems including NP-problems.

The physicists use the concept of quantum mechanics to discuss computation using hypercomputer, a computing machine that is far more powerful than the Turing machine [40]. Although, the hypercomputer is very fast, robust, and capable of solving a wider range of problems, yet it is still limited in its functions. Thus the general belief is that since it is limited in its functions, there is no doubt that Turing machine has far more limitations. A few other non-computable problems are discussed in this paper.

## 4.1 Entscheidungsproblem

In1928, David Hilbert proposed the Entscheidungsproblem, David Hilbert posted 23 problems that defied solution at the time. One of such problems is the Entscheidungsproblem. In fact, it was Hilbert 10th mathematical problem as listed by him. The Entscheidungsproblem problem asks for an algorithm that takes as input a statement of a first-order logic based on some axioms and answer "Yes" or "No" according to whether the statement is universally valid, i.e., valid in every structure satisfying the axioms. By first-order logic we mean a statement is universally valid if and only if it can be deduced from the axioms. Thus the Entscheidungsproblem simply ask if an algorithm exists that can decide whether a given statement is provable from the axioms using the rule of logic [41]. Entscheidungsproblem is a decision problem for first-order logic. Thus Entscheidungsproblem ask: Is there an effective procedure (an algorithm) which, given a set of axioms and a mathematical proposition, decides whether it is or is not possible from the axioms? Mathematicians of that era such as Kurt Gödel, Alonzo Church, Stephen Kleene, Alan Turing and a host of others, provided some explanations and proofs. In fact, in 1936, both Church and Turing independently showed that there is no such algorithm, specifically, there is no algorithm for the theory of arithmetic [42]. Church used the concept of lambda calculus and Turing used his machine now called Turing machine. All these proofs point to the fact that the Entscheidungsproblem was undecidable. The works of Alonzo Church and Alan Turing were later combined and referred to as the Church-Turing Thesis.

## 4.2 Gödel Incompleteness Theorems

In 1931, Kurt Gödel published his First and Second Incompleteness Theorems often referred to as Gödel's Theorems. However, in this paper, we will not be concerned with the proofs of the theorem. Our major concern will be to investigate the implications of the theorems to computation and why some problems are said to be undecidable. The first theorem states that any consistent formal system S within which a certain amount of elementary arithmetic can be carried out is incomplete with regard to statements of elementary arithmetic. That is to say, in any true axiomatic theory that sufficiently rich to enable the expression and proof of basic arithmetic propositions, it will be possible to construct an arithmetical proposition P such that neither P nor its negation not(P), is provable from the given axioms. Hence, the system must be incomplete. P must, of course, be a true statement of arithmetic [43]. The second theorem states that no consistent axiomatic theory is sufficiently rich enough to enable the expression and proof of basic arithmetic proposition can prove its own consistency.

These two theorems are hinged on three main concepts: consistency, completeness, and decidability. By consistency we mean the set of axioms should be consistent and provable; by completeness we mean all mathematical truths should be deducible from those axioms; and by decidability we mean there should be a clearly formulated procedure such that, given any statement of mathematics, it can be shown within a finite time whether or not that statement logically follows the given axiomatic [44]. In dealing with these concepts, standard syntactic relationship must be followed rather that semantic relationship.

Digital Innovations
& Contemporary Research in
SCIENCE & ENGINEERING
Vol. 4 No. 4, Dec. 2016

That is, the relationship should be based on structure rather than on truth and meanings. Therefore, it suffices to say that a consistent system cannot contradict itself. That is, it is not possible to prove both a proposition, say P, and its negation not(P). On the other hand, a complete system is one in which it is possible to prove neither P or not(P) for any proposition P that is expressible within the system.

Gödel's first theorem is based on the theory of computable function and the second theorem is born out of the first to provide an explicit explanation whose existence is asserted in the first theorem. Gödel Theorems played a significant role in shaping the general intellectual context of the 20th century. They are among those mathematical discoveries which became most widely known oust mathematics itself [45]. Gödel's invented a numbering system to logical formulas in order to reduce logic to arithmetic and used it to prove his incompleteness theorem. His numbering system was able to show that simple mathematical operations such as addition, subtraction, multiplication, and recursion can be used to prove his incompleteness theorem.

Gödel gave a vivid illustration of undecidability by using his numbering system. According to him, we can imagine a powerful computer that accept inputs and produce outputs based on certain instructions. Suppose there is an integer number, say N, $N > 1$, where N is a prime number (not divisible by any positive integer other than 1 and N itself) by asking the computer to divide N by every integer between 1 and N – 1 and then stop when the division is evenly distributed or it reaches N – 1. Then Gödel's theorem states that even though the number is properly keyed into the computer and by extension a Turing machine, no computer can solve it. Such a number is said to be undecidable or non-computable. This assertion will always be true if we consider a number like 2. The square root of 2, ($\sqrt{2}$), for instance has no precise solution no matter how long it takes the computer or Turing machine to solve it. Another good example is pi ($\Pi$). They will eventually end up generating recursive values. Thus the theorems demonstrate the inherent limitations of every formal axiomatic system containing basic arithmetic.

## 4.3 The Halting Problem

The halting problem [46] is one of the most famous problems in computer science. This is because it has profound implications on the theory of computation and on how we use computers in our daily lives. The halting problem is a decision problem in computability theory and computation in general. It is stated this way: Given a description of a Turing machine and its initial input, determine whether the program, when executed on this input, ever halts (completes). The alternative is that it runs forever without halting. In other words, it is like asking a Turing machine to answer a question about another Turing machine [47]. It can be shown that it is not possible to construct a Turing machine that will answer this question in all cases. Thus the only general way to know for sure if a given program will halt on a particular input in all eases is simply to run it and see if it halts in which case it may halt or not.. However, it is possible for a program to run forever and in this case there will be no solution to the problem as the machine tends to run forever especially if the program has the GO TO statement.

The halting problem is therefore called non-computable or undecidable. However, the halting problem is easy to solve if the Turing machine is allowed to run forever given input that represent a Turing machine that does not itself halt. The halting language is therefore recursively enumerable. It must be noted that because Turing machines have the ability to "back-up" in their input tape, it is possible for a Turing machine to run for a long time in a way that is not possible with the other computation models previously described. It is possible to construct a Turing machine that will never finish running (halt) on some inputs.

We say that a Turing machine can decide a language if it eventually halts on all and give an answer. Thus there is limitation to the computing power of Turing machines since no Turing machine can solve halting problems. The halting problem is important irrespective of the theoretical implications of the question of whether there are languages that are recognizable by a Turing machine but are not decidable [48]. For instance, it has implications on the termination properties of the software we write and use. Computer scientists and programmers use software that they expect to behave in a decidable manner. As an example, consider the compiler for a particular language, we would expect that irrespective of the source code it is asked to compile, the execution of such software will always terminate, either successfully (with the compiled code) or unsuccessfully, with an indication of where the error in the source code lies.

Although, it has been argued by some researchers that the halting problems is not always undecidable, That is, the problems can be solved by carefully designing a correct algorithm and they supported their argument using some theorems. However, if we rely on the works of Alan Turing and many other contemporaries of his time such as Alonzo Church, Kurt Gödel, Emil Post, and Stephen Kleene, who all agreed that Turing machine can be simulated to any other computing machine which they called universal Turing machine, a standard machine that can be used for computation and it is also limited in its computing powers, then one will be tempted to conclude that there are still several problems in real life that Turing machines cannot solve.

## 5. CONCLUSION

The general belief among people or computer users is that there is no problem or function that a computer cannot solve no matter how difficult it is, if giving enough time and space in the computer memory. Since computer science was born knowing its limitations, there are certain problems or functions that computer cannot solve. These problems are said to be uncomputable. This paper discusses the limits to the power of Turing machine and by extension a computer as a universally accepted computing device with regards to what it can compute and what it cannot compute, that is, problems that can be solved and those that defy solution by any known algorithm. The strength of the Turing machine leads directly to a negative consequence of uncomputability. Thus there are limitations to the power of Turing machines. The halting problem, Gödel incompleteness problem, the Entscheidungsproblem, the Post correspondence problem, the DNA-interconnection processes in nature, cancer, the Cellular automata, the quantum mechanics and other natural phenomenon are just some few examples. 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, hence, a Turing machine is limited in its computational capabilities. Finally, the paper argues that there are still numerous problems that no algorithms exist for solving them.

## REFERENCES

[1] 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).
[2] Seigelmann, H. T. (1995). Computation Beyond the Turing Limit, Science, Vol. 268, pp. 545-598.
[3] Turing, A. (1947). Intelligent Machinery, in Collected Works of Alan Turing: Mechanical Ingtelligence, ed. D.C. Ine., Elsevier Science, 1992.
[4] Ogheneovo, E. E. (2014a). Universal Turing Machine: A Model for All Computational Problems, Int'l Journal of Innovative Research in Computer Science and Communication Engineering, Vol. 2, Issue 5, May 2014, pp. 4436-4446.
[5] Wegner, P. and Göldin, D. (2002). Computation Beyond Turing Machines, Communications of the ACM, Vol. 46, No. 4, pp. 100-102
[6] Turing, A. M. (1937). Computability and λ-Definability. The Journal of Symbolic Logic, Vol. 2, pp. 153-163.
[7] Davis, M. (1982). Computability and Solvability, New York: McGraw-Hill
[8] Maruoka, A. (2011). Universality of Turing Machine and Its Limitations. Concise Guide to Computation Theory, pp. 161-181, Springer-Verlag.
[9] Knuth, D. E. (1976). Mathematics and Computer Science, Coping with Finiteness. Science 194 (4271), pp. 1235-1242. doi: 10.1126/scince.194.4271.1235.
[10] Cleland, C. E. (1993). Is the Church-Turing Thesis True? Minds and Machines Vol. 3,
[11] Copeland, B. J. (2002b). Hypercomputation, Minds and Machines, Vol. 12, pp. 461-502.
[12] Cotogno, P. (2003). Hypercomputation and the Physical Church-Turing Thesis, British Journal for the Philosophy of Science, Vol. 54, pp. 181-223.
[13] Golding, D. Smolka, S., Wegner, P. (2001). Turing Machines, Transition Systems, and Interaction. In Proceedings of the 8th Int'l Workshop on Expressiveness in Concurrency, Aalborg, Denmark, August 2001.
[14] Knuth, D. E. (1966). Algorithm and Program, Information and Data Communications of the ACM, Vol. 9, No. 9, p. 654.
[15] Gurevich, Y. (2011). What is an Algorithm? Technical Report MSR-TR-2011-116, July 2011, Microsoft.
[16] Eberbach, E. (2002). On Expresssiveness of Evolutionary Computation: Is EC Algorithm? In Proceedings of 2002 World Congress on Computational Intelligence, WCCI'04, Honolulu, HI, pp. 564-569.
[17] Ogheneovo, E. E. (2014b). Turing Machine and the Conceptual Problem of Computational Theory, Research Inventy: Int'l Journal of Engineering and Science, Vol. 4, Issue 4, April 2014, pp. 53-60.
[18] 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.
[19] Machtey, M. (171). Classification of Computable Functions by Primitive Recursive Classes. In Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, Shaker Heights, Ohio, USA, May 3-5,1971, pp. 251-257. doi: 10.1145/800157.805054.
[20] Bolding, D. (2000). Persistent Turing Machines as a Model of Interactive Computation, FoIKS'00, Cottbus, Germany.
[21] Davis, M. (2003). The Myth of Hypercomputation. In Christof Teuscher, Editor, Alan Turing: Life and Legacy of a Great Thinker, pp. 195-212, Springer, 2003.

[22] Desshowitz, N. and Falkovich, E. (2011). A Formalization and Proof of the Extended Church-Turing Thesis. In Proceedings of the 7th Int'l Workshop on Developments in Computational Models (DCM'11), Zurich, Switzerland, July 2011.
[23] Piccinni, G. (2007). Computationalism, The Church-Turing Thesis and the Church-Turing Fallacy, Springer 2007, 154: 97-120. Doi 10.1007/s11229-005-0194-z.
[24] Kálmar, L. (1959). An Argument against the Plausibility of Church Thesis, in A. Heyting (ed.), Constructivity in Mathematics, North-Holland, Amsterdam, pp. 72-80.
[25] Church, A. (1936). An Unsolvable Problem of Elementary Number Theory, The American Journal of Mathematics, Vol. 58, pp. 345-363.
[26] Van, L. T. and Wiedermann, J. (2000). The Turing Machine Paradigm in Contemporary Computing, in Enquist, B. and Schmidt, W. (eds.) Mathematics Unlimited 2001and Beyond, LNCS, Spriger-Verlag, 2000.
[27] Denning, P. J. (2010). What is Computation? A Ubiquity Symposium Opening Statement, FINAL v5, pp. 1-11.
[28] 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.
[29] Arora, S. and Barak B. (2009). Computational Complexity: A Modern Approach, Cambridge University Press.
[30] Akl, S. G. (2005). The Myth of Universal Computation, Parallel Numerical'05, in Vajtersic, Trobec, R., Zinterhof, P. and Uhl, A. (eds.), pp. 167-192.
[31] Lewis, H. R. and Papadimitrinu, C. H. (1981). Elements of the theory of Computation, Englewood Cliffs, N. S. Prentice-Hall.
[32] Kleene, S. C. (1988). Turing's Analysis of Computability and Major Applications of It. In Half-Century Survey on the Universal Turing Machine, pp. 17-54, New York NY. Oxford University Press. doi: 10.1007/978-3-7091-6697-3
[33] Milner, P. (1997). Turing, Computing and Communication, a Lecture for 60th Anniversary of Turing's "Entscheidungsproblem" paper, Kings's College, Cambridge England.
[34] Church, A. (1936). An Unsolvable Problem of Elementary Number Theory, American Journal of Mathematics 58, pp. 343-363.
[35] Church, A., Review of Turing A. M. On Computable Numbers, with an Application to the Entshceidungsproblem. In Proceedings of the London Mathematical Society, Vol. 2, No. 42, pp. 230-265), Journal of Symbolic Logic, Vol. 2, 1937, pp. 42-43. doi: 10.2307/2268810.
[36] Minsky, M. (1967). Computation: Finite and Infinite Machines, Prentice-Hall, Inc., Englewood Cliffs, N. J.
[37] Post, E. L. Absolute Unsolvable Problems and Relatively Undecidable Propositions: Account of an Anticipation. In M. Davis, Editor, Solvability, Provability, Definability: The Collected Works of Emil L. Post, pp. 375-441. Birkhaüser, Boston, MA, 1994, Unpublished Paper, 1941.
[38] Addleman, L. M. (1994). Molecular Computation of Sciences to Combinatorial Problems, Science, 266.1021-1024.
[39] Wolfram, S. (1959). A New Kind of Science, Wolfram Media, Inc. http://www.wolframscience.com/nksonline/toc.html
[40] 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.

[41] Turing, A. (1937). Computable Numbers with an Application to the Entscheidungsproblem. In Proceedings of the London Mathematical Society, 42:2, 1936, pp. 230-265; A Correction, ibid, 43, 1937, pp. 544-546.
[42] Quinn, F. (2012). A Revolution in Mathematics? What Really Happened a Century Ago and Why it Matters Today, Notices of the America Society of Mathematics, Vol. 59, No.1, pp. 31-37.
[43] Desshowitz, N. and Gurevich, Y. (2008). A Natural Axiomatization of Computability and Proof of Church's Thesis. Bulleting of Symbolic Logic, Vol. 14, No. 3, pp. 299-350. doi: 10.2178/bsl/1231081370.
[44] Gödel, K. (1934). On Undecidable Propositions of Formal Mathmatical Systems, Lecture Notes by S. C. Kleene and J. B. Rosser, Institute for Advanced Study, Princeton.
[45] Beklemishev, L. D. (1945). Gödel Incompleteness Theorems and the Limits of Their Applicability. Russian Math Surveys 0:0, 1-000.
[46] Bienvenu, L, Desfonfaines, D. and Shen, A. (2016). Generic Algorithms for Halting Problems and Optimal Machines Revisited. Logical Methods of Computer Science, Vol. 12, No. 2, Issue 1, pp. 1-29.
[47] Sassaman,L., Patterson, M. L., Bratus, S. and Shubina, A. (2011). The Halting Problems of Network Stack Insecurity, Login: Vol. 36, No. 6.
[48] Eberbach, E. (2003). Is Entscheidungsproblem Solvable? Beyond Undecidability of Turing Machines and its Consequence for Computer Science and Mathematics, Modeling and Simulation, Narosa Publishing House, New Delhi, chapter 1, pp. 1-32.