2023 International Conference on Computational Science and Computational Intelligence (CSCI) | 979-8-3503-6151-3/23/$31.00 ©2023 IEEE | DOI: 10.1109/CSCI62032.2023.00073

Understanding the Computational Complexity of Diverse Classes of Turing and Super-Turing Computational Models

1st Ghada Abdelmoumin
Department of Electrical Engineering and Computer Science
Howard University
Washington, DC 20059 USA
ghada.abdelmoumin@bison.howard.edu

2nd Chunmei Liu
Department of Electrical Engineering and Computer Science
Howard University
Washington, DC 20059, USA
chuliu@howard.edu

3rd Danda Rawat
Department of Electrical Engineering and Computer Science
Howard University
Washington, DC 20059, USA
danda.rawat@howard.edu

***Abstract*—The need to solve scientific problems using automated means and hence analyze the performance and measure the complexities of their algorithmic solutions had led to the realization of various computational models. The underlying of these computational models are the mathematical models that provide intuitions for the problems in question and formally describe the problem(s) to be solved by a particular model. In the heart of these models of computations is the Turing Machines abstractions model, a mathematical model of computation that formed the basis of successive models of computations and provided the theoretical foundation of computability. In this paper, we explore various classes of computational models and intuitively understand their computational complexity. We focus on different Turing and super-Turing models of computations and study their complexity in terms of space and time. We provide a comparative analysis of infinitary, fuzzy, hyper and super-Turing models using computational characteristics pertinent to Turing computational models in general. Further, we show the conjectured relationships of the four models to the Turing machines and universal Turing machine.**

***Index Terms*—computational models, complexity and computability, infinitary, fuzzy, hyper-computing, super-computing, neural, quantum**

I. Introduction

The model of computations introduced by Alan Turing in 1936, known as the Turing Machine (TM), which abstractly defined the notion of the programmable machine, forms the basis of diverse formal models of computations that existed or are in existence today. Further, the Turing model abstractions inspired differentiable versions of classical and contemporary computational models. The classical TM, which formalized the notions of algorithm, computation, and computable, has several variants with the same computational power, making it an appropriate model of computation [17], [23]. In his thesis titled "The Church-Turing Thesis," Turing conceived using mathematical rigor the notion of a universal single computing machine that can adequately describe the use of algorithmic means to perform tasks and simulate other computing machines [8], [17], [20]. The assertion that there exists a universal Turing Machine (UTM) made it possible to study the computation properties of every possible computing device and simulate any other TM.

Following the introduction of the UTM was the development of numerous classes of computational models, both classical and contemporary, some of which are merely variations or extensions of the fundamental or standard TM model, and others claim to either surpass or break the Turing limits, i.e., the TM has computational limits where it can never solve the halting problem [3], [15], [19], [24], [26]. For example, the class of hyper-computation models challenges the assertion that no machine can compute more than the TM, and the super-computing category of computational models challenges the affirmation that no computing device can surpass the computational model of TM or more potent than the UTM.

The literature on the computability and complexity of computational models reports on numerous classical and contemporary formal models that range from Turing and super-Turing to Non-Turing models. Consequently, several classes of computational models exist. These classes include but not limited to the discrete, analog, serial, symbolic, dynamic, stochastic, hyper-computing, super-computing, and mem-computing types of models. In this paper, we focus on models that characterize as discrete, symbolic, analog, hyper-computing, and super-computing classes of computational models. We study the classical TM model and its variants, Infinite Time Turing Machines (ITTMs), Fuzzy Turing Machines (FTMs), and their variants, Neural Turing Machines (NTMs), and Quantum Turing Machines (QTMs). While the first three models of computations classify as discrete (or serial) and symbolic models, the third and fourth models classify as

2769-5654/23/$31.00 ©2023 IEEE
DOI 10.1109/CSCI62032.2023.00073

analog hyper-computing, and analog super-computing models, respectively.

The remainder of this paper organizes as follows. In section II, we provide an overview of the TM and UTM as the ultimate computational model and the standard model of computations. In section III, we discuss the computability and complexity theories and further define computational complexity from a TM perspective as a reference model. In section IV and V, we present the computational models in focus and discuss their functions and computational complexity. In section VI, we presents some comparison results. Then, we conclude and discuss future work in section VII.

II. The Standard Turing Machine Model

Computational models before the Turing machine model had either limited memory or a restrictive memory allocation method, which inevitably hindered their generalization to solving general-purpose computing problems. Unlike these early models, the TM computational models addressed these shortcomings and accurately depicted a general-purpose computer that can solve problems lying within the model theoretical boundaries [27]. With it is infinite tape serving as unlimited memory, a read/write head that moves across the tape, a set of alphabets, and the ability to compute, it was strictly more powerful compared to other models such as finite automata.

A formal description of the model defines a state transition function (δ) that describes the transition from one state to another using simple transitions until the computation terminates in a halting, i.e., accept or reject or non-halting state. Fig. 1 [27] shows the TM formal definition.

A ***Turing machine*** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where Q, Σ, Γ are all finite sets and

1. Q is the set of states,
2. Σ is the input alphabet not containing the ***blank symbol*** $\sqcup$,
3. Γ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta\colon Q \times \Gamma \longrightarrow Q \times \Gamma \times \{\text{L}, \text{R}\}$ is the transition function,
5. $q_0 \in Q$ is the start state,
6. $q_{\text{accept}} \in Q$ is the accept state, and
7. $q_{\text{reject}} \in Q$ is the reject state, where $q_{\text{reject}} \neq q_{\text{accept}}$.

Fig. 1. The Turing Machine (TM) computational model formal description

It is worth noting that the TM halting problem is undecidable, i.e., there exists no algorithmic solution to tell whether a given TM halts on a given input [11]. Typically, a TM accepts a collection of strings that comprise the language recognized by the TM. Hence, a TM can either decide or recognize a language it accepts as an input. In general, a TM can "decide" an enumerable language and "recognize" a recursively enumerable language.

Two subsequent variants of the classical TM are deterministic TM (DTM) and non-deterministic TM (NDTM). These two variants directly correspond to the deterministic and non-deterministic algorithms, respectively [22]. The single computation model is what distinguishes the DTM from the NDTM with its many-to-many mapping and several possibilities. Other classical variants include single-tape TM, multi-tape TM, and write-once TM. A multi-tape TM has a read-only input tape and a finite number of read-write tapes. Sipser [27], states that every NDTM has an equivalent DTM, and a multiple-tape TM is convertible into a single-tape TM. There exist contemporary TM variants or differentiable versions, therein, such as infinitary TM, ordinal TM, four-dimensional parallel TM, one-tape two-way one-head off-line linear-time TM, multiply-exponential write-once TM to name some [4], [18], [29], [30], [33]. We cover the other variants and differentiable versions of TM we noted in this paper in section IV.

One of the most fundamental concepts of TM is its ability to simulate other machines. The Church-Turing thesis has an explicit algorithmic assertion, as well as an implicitly-stated physical statement, in principle [8] [17] [27]. That is, a UTM can compute every naturally computable function and simulate every realizable or natural physical system. Consequently, TM has an established view in the field of theoretical computing as the reference model for subsequent and successive computational models. Further, due to its simplistic mathematical foundation and closeness to real computing machines, TM can be used to analyze and measure the complexity of an algorithm in a meaningful manner.

III. An Overview of Computational Complexity

To solve a problem using algorithmic means, one must have the ability to measure the time and space taken by the algorithm used to solve the computational problem. Respectively, computational complexity provides the required framework of analysis to investigate the computational resources required to solve computable problems. Two essential measures to classify computational problems are the time and space taken by their respective algorithms. While the time complexity constitutes the basis for classifying problems according to the amount of time taken to solve the problem, space complexity classifies the problems based on the amount of space or memory requirements.

It worth noting that the terms "computability" and "complexity" are not equivalent. Computability implies that all reasonable models of computation are of equivalence, i.e., they all decide the same class of languages [27]. Conversely, for complexity, the choice of the model affects the time complexity of computational languages. Hence, for complexity, all languages are not decidable using the same amount of time. It is notable to mention that we don't mention space complexity because, as noted by Sipser in [27], we can establish a relationship between the complexity of time and space. In this paper, we focus on the model complexity to understand the computational complexity of the variants and differentiable TMs we are investigating.

Concisely, the running time of an algorithm is a function of the length of the input string irrespective of any other parameters [27]. Generally, it is strenuous to estimate the running time of an algorithm accurately. As such, an estimate of the

$P = \bigcup_k TIME(n^k)$ where P is the class of languages that are decidable in polynomial time on a deterministic single-tape Turing Machine and n is the size of input and k is a constant.

Analogous to P, $PSPACE = \bigcup_k SPACE(n^k)$ is the class of languages that are decidable in polynomial space on a deterministic Turing Machine and n is the size of input and K is a constant.

Fig. 2. The theory of complexity definition

time is enough to study the time complexity of any algorithm. Given the expression of the running time of an algorithm, the asymptotic analysis provides the required frame of analysis to estimate the running time. However, it stipulates an estimation of the algorithm running time, considering only "large" inputs and higher-order terms of the running time expression. When performing asymptotic analysis, it is imperative to establish upper and lower bounds, often referred to in the literature as *Big-O* notation and *small-o* notation. When applied to functions, *Big-O* notation indicates that one function $f(n)$ is no more than another $g(n)$, i.e., $f(n) \leq g(n)$, whereas, small-o notation signals that one function is asymptotically less than another (i.e., $f(n) < q(n)$).

Although asymptotic notations help establish the upper and lower bounds necessary to analyze the running time of an algorithm for "large" inputs, notions such as polynomial time and exponential time create further meaning. For example, polynomial-time algorithms are faster than exponential-time algorithms, and exponential algorithms exhaustively perform the necessary computations to solve the problem. Therefore, asymptotic analysis and polynomiality play a substantial role in understanding the time complexity of computational models. A polynomial algorithm has a polynomial upper bound on the number of steps it uses on an input of size n and performs each computational step in polynomial time. Given this and the definition in [27] that shows the description of complexity theory (see Fig. 2), we may deduce that a particular class of languages is decidable in polynomial time if there exists a computational model that runs in polynomial time.

To understand the relationship between the computational models investigated in this paper, we use asymptotic analysis and polynomiality to assess their computational complexity. Also, in developing an understanding of how to compare computational models using simulations and the classical TM as a reference model, we consider the result established in [8] that all deterministic models are polynomially equivalent. For the analysis of space complexity, we perceive that deterministic machines use a small amount of space to simulate non-deterministic ones. Moreover, we observe the derivations that a TM of time $t(n)$ can use at most $t(n)$ space for $t(n) \geq n$ and vice versa, and $P \subseteq PSPACE$, i.e., the speed of the machine is proportional to its space requirement.

IV. Variants of TMs: Infinitary and Fuzzy Models

In this section and the subsequent section, we present four computational models, some of which are variants of the classical TM, and others are differentiable (invariant) versions. We first introduce the infinite Time Turing Machines (ITTMs) and Fuzzy Turing Machines (FTMs), which are variants of the standard or classical TM. ITTMs is the first natural infinitary model of computability. Later, in section V, we introduce the Neural Turing Machines (NTMs) and Quantum Turing Machines (QTMs), respectively.

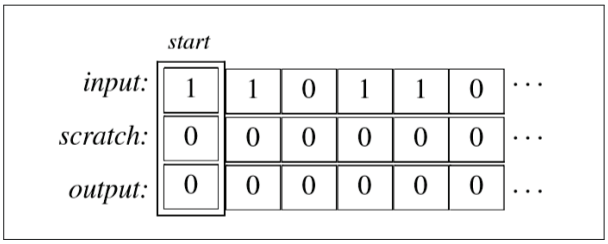

Fig. 3. The hardware of the ITTM depicting the initial state

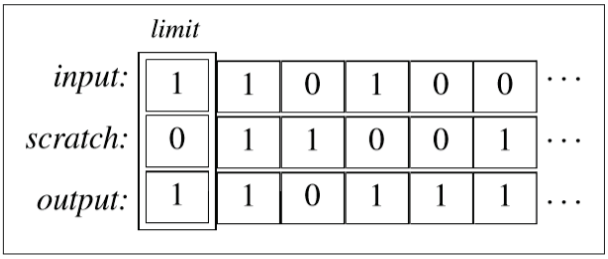

Fig. 4. The ITTM hardware showing the limit configuration

A. Infinite Time Turing Machines (ITTMs)

ITTMs is an extended version of the classical TM as defined by Jeff Kider in 1989 and introduced by Hamkins and Lewis in 2000 [15]. In their work, the authors extended the operation of the classical TM to infinite ordinal time, later expressed as transfinite in [16], and used the set of real numbers ($\mathbb{R}$) to investigate the computational complexity of the resulting model of supertasks (model with infinite computation steps). Their concept of ITTM has led to the notion of relative computability, and in examining the supertask computability, the authors were able to establish that the resulting ITTM can decide every $\pi^1{}_1$ set (a set that codes a well-ordered relation on $\mathbb{N}$) and the semi-decidable sets from a portion of the $\Delta^1{}_2$ sets.

Underlying an ITTM is the notion of infinite computability, i.e., the model can complete infinite computational steps in finite time. This model, which has the three tapes and a read-write head, carries the computational steps in time, similar to ordinal numbers. Fig. 3 and 4 [16] describe the hardware in the initial and limit configurations, and Fig. 5 describes the operation of an ITTM. The computational steps orders ordinally as 0, 1, 2, … ω, ω+1, ω+2, . . ., ω+ ω+1, ω+ ω+2, ω+ ω+3, . . ., and after each stage α, there is a distinctive stage α+1.

ITTMs can compute any task that is computable by the classical TM; hence it is more powerful than classical TM.

1. Proceed with computations ordinally in time (i.e., 0, 1, 2, . . .)
2. If the computation doesn't halt at any finite stage 0, 1, 2, . . .
3. Enter the first infinite stage ω
4. Continue with stages $\omega + 1$, $\omega + 2$, $\omega + 3$, . . .
5. If computation doesn't halt at any stages $\omega + 1$, $\omega + 2$, $\omega + 3$, . . .
6. Enter the second limit stage $\omega + \omega$
7. Continue with stages $\omega + \omega + 1$, $\omega + \omega + 2$, $\omega + \omega + 3$, . . .
8. and so . . .

Fig. 5. The operation of ITTM in time similar to ordinal numbers

In particular, it decides the halting problem , undecidable by classical TM, in ω many steps by simulating the operation of program p undecidable by TM. To understand the time complexity, Hamkins and Lewis defined a clockable mechanism that he based on the notion of the clock and computation step α, such that "α is clockable if there is a computation on input 0 that takes exactly α many steps to complete [15]. "The authors used a simulation to run the clock and the computations simultaneously, and when the ITTM halts, the clock stops. A simple analysis using the clockable mechanism and ITTM showed that α is clockable if the machine halts after n step. Similarly, ordinal ω and ω^2 are clockable when the machine halts when it encounters the limit.

The space complexity of ITTMs has been investigated by Carl, Löwe, and Winter [5], [18], [32], and Deolalikar et al. addressed ITTMs time complexity in [7]. Carl suggests that unlike others, ITTMs computations don't use the whole tape length ω, and therefore, don't have the same space usage. Conversely, Löwe suggests that tape of length ω can simulate larger-order tapes, hence the contents of the tape during computations is a recommended measure of space usage instead of the number cells used by the machine. Further, he suggests that a relationship between time and space complexity for ITTMs exists, a statement that Carl disapproved mathematically by showing that sets of real numbers are ITTM-decidable in space but are undecidable in time bounded by α. Hence, the complexity of the contents doesn't influence the time complexity of the machine. Furthermore, Carl argued that equalities $P = PSPACE$, $P^+ = PSPACE^+$ and $P^{++} = PSPACE^{++}$ don't hold for ITTM.

The time complexity analysis of ITTMs is as follows. Given the related problem $P = NP$, Deolalikar et al. established that, for ITTMs, $P \subset NP \cap co\text{-}NP$; where $NP \cap co\text{-}NP$ is for the class of hyperarithmetic sets, and $P^+ \subset NP^+ \cap co\text{-}NP^{++}$, where $NP^+ \cap co\text{-}NP^{++}$ is for the more general classes of $\mathbb{R}$. Further, the authors show that for contiguous and non-contiguous blocks of clockable ordinals, $P_\alpha \neq NP_\alpha \cap co\text{-}NP_\alpha$, and $P_\beta \neq NP_\beta \cap co\text{-}NP_\beta$, respectively. Additionally, they establish that $P^f \neq NP^f \cap co\text{-}NP^f$ for more functions $f : \mathbb{R}^2 \to ORD$. In the following section, we present the Fuzzy Turing Machines (FTMs) and discuss their computational complexity.

A non-deterministic Fuzzy Turing machine is a 9-tuble, $(S, T, I, \Delta, b_1 q_0, q_f, \mu, *)$, where

S and T are finite sets and

1. S is the finite set of states,
2. T is the finite set of tape symbols,
3. I is the set of input symbols; I⊆T,
4. Δ is the next-move relation which is a subset of S × T × S × T × {−1;0;1},
5. b, in $T - I$, is the blank,
6. q_0 is the initial state,
7. q_f is the final, or accepting state,
8. $\mu: \Delta \to [0,1]$ is a function that to each move δ assigns the truth degree $\mu(\delta)$ of its membership in Δ, and
9. $*$ is a t-norm.

Fig. 6. The formal definition of the non-deterministic FTM

B. Fuzzy Turing Machines(FTMs)

The need to develop models of computations that can solve the undecidable problems formulated by the Church-Turing thesis and the establishment of the fundamentals of fuzzy languages (fuzzy logic) has led to the development of formal models of random or fuzzy computation [17], [22], [31]. The classical FTMs are non-deterministic TM with fuzzy algorithms or a set of fuzzy instructions, each with an assigned degree of truth or a value between 0 and 1. Wiederman in [31] observes that Fuzzy Turing Machines or FTMs can solve undecidable problems and shows that FTMs accept fuzzy languages that correspond to the union ${\Sigma^0}_1 \cup {\pi^0}_1$ of recursively enumerable languages and their complements. Additionally, he shows that the P class (polynomially time-bounded computations) of FTMs corresponds to the union $NP \cup co\text{-}NP$.

With the introduction of FTMs, the once sought notion of a super-Turing machine revived again. Thus, leading to two variants: non-deterministic FTMs with classical-acceptance criteria and non-deterministic FTMs with partially computable acceptance criteria. While the later was an invariant of the Church-Turing thesis, the former was able to solve undecidable problems demonstrating super-Turing computational power, hence a variant of the classical TM. The non-deterministic FTM (NFTM) with a single-tape is a 9-tuple $(S, T, I, \Delta, b_1 q_0, q_f, \mu, *)$. Fig. 6 shows the formal definition of NFTM. A t-norm is a binary operation $*$ on $[1, 0]$, where $*$ is commutative and associative, non-decreasing in both arguments, and for all $x \in [0, 1], 1 * x = x$ and $0 * x = 0$.

The fuzzy language accepted by NFTM is the fuzzy set of ordered pairs. Assume F to be an NFTM, then $L(F) = \{(w, e(w)) | w$ is accepted by F with truth degree $e(w)\}$. To analyze the time complexity and specify the efficiency of the polynomial time-bound NFTM, one must first define the time complexity for the classical FTM and the corresponding classes of complexity. Fig. 7 shows these definitions, as well as the specification of the efficiency of the polynomial time-bounded NFTM, where FNP corresponds to ϕ.

Li [17] studied, in addition to the NFTM, other variants of FTMs such as deterministic FTM (DFTM), and universal FTMs (UFTM), i.e., a universal FTM that can simulate any

Time Complexity. Let F be an NFTM and $L(F)$ be the language recognized by F, F is of time complexity $T(n)$ if for all n and all inputs of length n, F accepts all strings from $L(F)$ of length n in at most $T(n)$ steps.

Corresponding Complexity Class: $NFTM(Time(T(n)))$represents the class of all fuzzy languages recognized by a classical $NFTM$ in time $T(n)$. FNP denotes the class of all fuzzy languages recognized by a classical $NFTM$ in polynomial time.

Polynomially time-bounded NFTM: $FNP = NP \cup co\text{-}NP = \Sigma_1^P \cup \Pi_1^P$

Fig. 7. Definitions of classical FTM time complexity and corresponding classes, and characterization of polynomial time-bounded NFTM

other FTM, and provided several notable formulations of these variants. Also, he showed that while DFTM and NFTM are nonequivalent in recognizing fuzzy languages, they are equivalent in deciding fuzzy languages. Further, he indicated that a UFTM only exists if the fuzzy set membership is restrictive to the fixed finite set [0, 1] and proposed the notions of fuzzy recursively enumerable languages and fuzzy recursive languages. In the next subsection, we provide an overview of Neural Turing Machines (NTM) and Quantum Turing Machines (QTMs).

V. Differentiable Versions of TMs: Hyper- and Super-Computational Models

The models introduced above broadly classify as discrete or serial models. These models mainly performed classical or natural computations on discrete data sets. The recent advancements in computing have led to powerful computing machines with intricately complex algorithms. With the revival of neural computing, analog data models are rapidly emerging, and the new paradigm shift in computing with the advancement in quantum computing has sparked an interest in hyper- and super-computations. Consequently, the widely accepted assertion that no machine can be more potent than universal TM or UTM can no longer remain unchallenged. Hence, there is a growing interest in formal models of computation, such as neural or hyper-computation and super-Turing computation models, that challenges this notion of the UTM and go beyond the limits of classical computations [8], [24], [25], [31]. Principles of physics such as quantum, relativistic, or infinitary computing had sparked a new interest in models that can compute beyond the UTM. Hence, emerged FTM, neural Turing machines (NTMs), and Quantum Turing machines (QTMs), to name a few.

A. Neural Turing Machines(NTMs)

NTMs are instances of memory augmented neural networks that decouple computation from memory through the addition of an external memory unit [6], [12]. Generally, NTMs have powerful random access memory and perform read/write operations [13].

The architecture of an NTM consists of a neural network controller, which interacts with the external environment using input and output vectors, and a memory bank. Fig. 8 [12] shows the underlying architecture of NTMs. On each cycle of the machine, the controller receives input and emits output. It uses a set of parallel read/write heads to read from

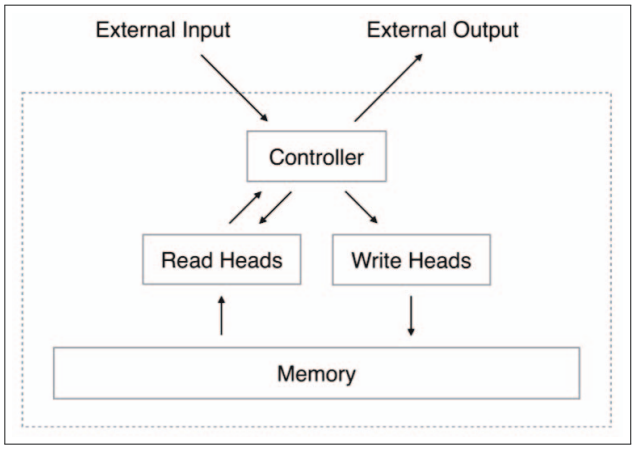

Fig. 8. The underlying architecture of NTMs

or write to the external memory. Graves et al. [12] have empirically shown that NTM they implemented can solve a problem by learning compact internal programs. While the NTM is analogous to a TM, "it is differentiable end-to-end." Siegelemann [26] introduced an analog neural network that keeps tracks of computational constraints while allowing for supra-Turing computational power. The author proposed the NTM as a standard for hyper-computation analogous to the UTM and established a connection between the proposed neural computational model and classical computability. Siegelemann proved that classical TM is a subset of the recurrent networks, a set of neural network algorithms that can process sequential data (e.g., calculate continuous values between 0 and 1). Thus, A network with rational weights can simulate any multi-tape Turing machine in real-time (the interconnection weight in a network are natural numbers). Despite the various implementation of NTMs presented in the literature and this paper that viewed in large as super-Turing computational models that compute beyond the limit of classical UTM, Sharma, and Upadhyay [24] expressed an opposing view. In their work, the authors proved that NTMs are not necessarily computing the incomputable, i.e., solve the undecidable problem. They further asserted that models compute beyond the Turing limits if they have in advance a hyper computer as a supplementary unit. Otherwise, they are equivalent to the TM model. Siegelmann [26] analyzed the computational power of hyper-computational models and stated that the analog recurrent neural network (ARNN) model, which is hyper-computational, can compute in polynomial time exactly the function in P/poly, where P is the class of all recursive functions that are computable in polynomial time and poly is used to denote that the sizes of the different neural networks are polynomial in the input length.

B. Quantum Turing Machines(QTMs)

In 1985, Deutsch [8] showed that the notion of universality in computation could extend beyond the UTM by noting that there exists a class of computational models that is a quantum

A ***quantum Turing machine (QTM)*** *informally* is a triple, $\langle \alpha, q, \beta \rangle \in \Sigma^* \times Q \times \Sigma^*$, where

Q is a set of finite states and,

1. $q \in Q$ is the current state,
2. $\beta \in \Sigma^+$ is the right content of the tape, where
   a. Σ is the tape alphabet,
   b. The first symbol of β is the one in the current cell of the tape,
   c. $\beta = \mu\beta'$, where μ is the content of the current cell and β' is the longest string ending with different from blank, for example, when the current cell and all right content of the tape is empty, QTM can be written as the triple $\langle \alpha, q, \lambda \rangle$, where λ is the empty string, and
3. α , which is either λ or the longest string on the tape, is the left content of the tape.

Fig. 9. The Informal Definition of QTM

generalization of Turing machines and further proved that universal quantum computers are compatible with the Church-Turing thesis assertion of a universal Turing machine or UTM. He prophesized that quantum Turing machines (QTMs) would have remarkable non-producible properties by classical TMs. In accord with Deutsch, Muller [20] showed that there exists a universal quantum Turing machine (UQTM) that simulate every other quantum Turing machine (QTM) until it halts and then halts itself with probability one; its predecessor UQTM can simulate every other QTM for arbitrary pre-assigned number of time steps. In his work, Muller introduced the notion of mutually orthogonal space and showed that the q-bit encoded information could decompose into a classical and quantum part. The QTM proposed by Muller, similar to TMs, consists of an infinite tape, a control, and a read/write head. Unlike the classical TM, the number of computational steps is unknown in advance; hence the machine evolves unitarily in discrete time steps. Guerrini et al. [14] introduced a new class of QTMs (Fig. 9 shows an informal definition of QTM) and defined a class of quantum computable functions. These quantum functions are a general mapping form quantum state to natural numbers with a probabilistic distribution.

The proposed QTM doesn't change the content written in its tape when it enters the final state; hence "its evolution continues to remain in the final state." This unitary evolution of the machine is possible through the use of a counter. Such evolution forces the machine to keep modifying its configurations. An essential characteristic of this QTM is that it doesn't have a halting state. Instead, it continues its evolution since the only possible evolution is to increment the counter by one and keep the rest of its configurations unchanged. These configurations include the internal state, read/write head position and tape content. To define the transitions of QTM graphically, first, a formal definition of QTM is obtained by defining a pre-QTM (pQTM) and time evolution operator. Fig. 10 show QTM formal definition and Fig. 11 [14] shows QTM transitions using a graph.

Given the formal definition of pre QTM, QTM formal definition is as follows: A pQTM is a Quantum Turing Machine (QTM) when its time evolution operator U_M unitary.

The above transitions graph shows three types of transition: a looping transition of the target node (left), a self-loop on any target node (right), and a transition for every non-target node

A ***pre quantum Turing machine (pQTM)*** is an ordered 7-tuple, $\left(\Sigma, Q, Q_s, Q_t, \delta_0, q_i, q_f\right)$ where

$\Sigma, Q, Q_s, and\ Q_t$ are all finite sets and

1. Q is the set of finite states,
2. $Q_s \subseteq Q$is the set of source states,
3. $Q_t \subseteq Q$ is the set of target states,
4. $q_i \in Q_s$ is a source state, called the initial state,
5. $q_f \in Q_t$ is a target state, called the final state,
6. $\delta_0 : \left((Q \setminus Q_t) \times \Sigma\right) \to \ell^2((Q \setminus Q_s) \times \Sigma \times \mathbb{D})$ is the quantum transition function of M, where $\mathbb{D} = \{L, R\}$.

Fig. 10. Formal Definition of QTM in terms of pQTM

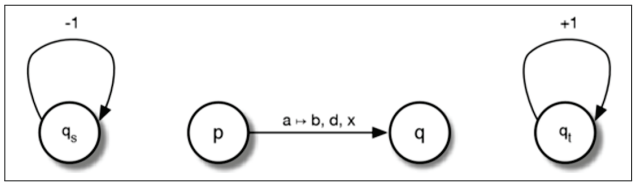

Fig. 11. QTM Transitions Graph

(middle). In [29], Tusˇarova´ defined a probabilistic quantum Turing machine (pQTM) that is slightly different than the classical TM to provide a formal definition to a QTM. Unlike classical TM, which outputs one triple of the tape content, new state, and new transition, pQTM output the triple's probabilistic distribution. Based on the pQTM definition, Tusˇarova´ informally defined QTM to be similar to pQTM except that QTM output complex numbers or amplitudes instead of probabilities and provided the formal definition shown in Fig. 12. Additionally, Tusˇarova´ defined the complexity class of QTM as BQP (Bounded-error quantum polynomial time) where he specifies BQP as follows: "A language L is in BQP if there exists a polynomial $p(n)$ such that L is accepted by some quantum Turing machine with time complexity $p(n)$." He further suggested that BQP $\in$ PSAPCE. Bernstein and Vazirani [1] provided the first formal evidence that QTMs violate the modern formulation of the Church-Turing thesis, and showed that there exists a problem solvable in polynomial time on a QTM, but requires super-polynomial time on a bounded-error probabilistic TM. Further, they asserted that this problem is not in the class BPP (bounded-error probabilistic polynomial time), and argued given BPP $\subseteq$ BQP $\subseteq \mathrm{P}^{\#P}$, that it is not possible to mathematically prove that QTMs are more robust than the classical probabilistic TM.

A ***quantum Turing machine (QTM)*** is an ordered 6-tuple, $\left(\Sigma, \Lambda, Q, q_i, q_f, \delta\right)$ where

$\Sigma\ and\ Q$ are both finite sets and

1. Q is the set of states,
2. Σ assume it equals $(0,1,\Lambda)$ is a finite set, called alphabet, of all possible symbols,
3. $\Lambda \in \Sigma$ is the blank symbol,
4. $q_i \in Q$ is the initial state,
5. $q_f \in Q$ is the final state, and
6. $\delta : \Sigma \times Q \longrightarrow H$ is the transition function and H is the Hilbert space spanned by base vectors corresponding to triples from $\Sigma \times Q \times \{L, R\}$.

Fig. 12. The QTM formal definition

Fortnow [10] argued that BQP and BPP exhibit similar

behavior, mainly in their ability to perform searching and information hiding. Spakowski et al. [28] introduce EQP (zero-error BQP or exact quantum polynomial time), BQP, and NQP (non-deterministic quantum polynomial time) as three quantum complexity classes, analogous to P, BPP, and NP, respectively, that represent the computational power of quantum computers. All three classes represent the class of languages L accepted by a QTM running in polynomial time, but with varying probabilities. EQP, BQP, and NQP represent the classes of languages L such that, for each $x \in \Sigma^*$, if $x \in L$, QTM accepts EQP with probability 1, BQP with probability $2/3$ at least, and NQP with nonzero probability, and if $x \notin L$, QTM accepts EQP with probability 0 and BQP with probability 1/3 at most.

VI. Results and Discussion

In section IV and V, we described the variants and differentiable end-to-end versions of the classical TMs and examined their computational complexity. In this section, we compare the four models: ITTMs, NTMs, FTMs, and QTMs based on computational model characteristics such as type, accepted language(s), undecidable problems, and computational complexity (shown in Table 1, 2, 3 and 4). Also, we show the conjectured relationship among the TM, UTM, and each one of the four models in Fig. 13, separately. Given Fig. 13, the first set of ovals on the top-left corner shows that UTM can compute any tasks computed by TM and that ITTM can compute any task computed by UTM and surpass UTM by deciding the halting problem in finite time given infinite computational steps. The second set of ovals on the top-right corner shows that NFTM and DFTM can compute any task computed by UTM and surpass it by solving undecidable problems. The third set of ovals on the bottom-left corner shows that the NTM can compute any task computed by UTM. NTM can compute beyond the limits of the classical UTM. Finally, the set of ovals on the bottom-right corner shows that QTM can compute any tasks computed by the UTM and, further, can surpass the TM limits by continuing to remain in the final state, i.e., QTM does not have a halting state.

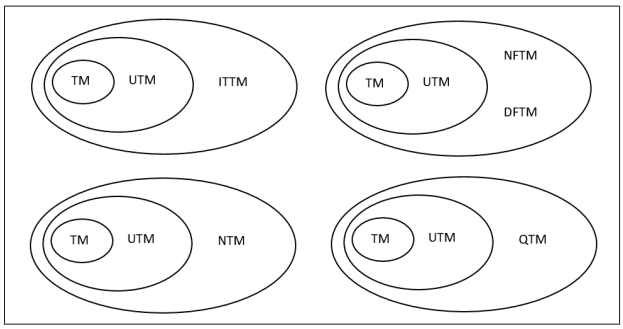

Fig. 13. The Conjectured Relationship Between TM, UTM, and each of the Four Models (ITTM, NFTM, NTM, and QTM

Our interest in the future is to evaluate all four models as arbitrary models over arbitrary domains and measure the

TABLE I
Comparison Result Based on Type

Model Type	ITTMs	NFTMs	NTMs	QTMs
Discrete	X	X	–	–
Serial	X	X	–	–
Analog	–	–	X	X
Hyper-computation	–	–	X	–
Super-computation	–	–	–	X

TABLE II
Comparison Result Based on Acceptable Languages

Model Language	ITTMs	NFTMs	NTMs	QTMs
Recursive	–	–	–	–
Recursively Enumerable (r.e.)	X	X	–	–
Co- r.e.	–	X	–	–
EQP, BQP and NQP	–	–	–	X
Context-free	–	–	X	–
Regular	–	–	X	–

TABLE III
Comparison Result Based on the Halting Problem

Model Decidability	ITTMs	NFTMs	NTMs	QTMs
Decidable	X	X	X	X
Undecidable	–	–	–	–

TABLE IV
COMPARISON RESULT BASED ON TIME COMPLEXITY (P)

Model Complexity	ITTMs	NFTMs	NTMs	QTMs
$P \subset NP \cap co\text{-}NP$	X	–	–	–
$NP \cup co\text{-}NP$	–	X	–	–
P/poly	–	–	X	–
BPP$\subseteq$BQP$\subseteq$ $P^{\#P}$or BQP $\subseteq$ PP	–	–	–	X

accepting times using a commonly accepted language and the UTM as a reference model. More interesting is to show if a conjectured relationship among TM, UTM, and all four models can be formally derived.

VII. CONCLUSION AND FUTURE WORK

In this paper, we presented four computational models spanning four classes of computations, classical, infinitary, neural or hyper, and quantum or super. First, we introduced the classical model of computations and provided an overview of the notion of universality computation asserted by the Church-Turing thesis and computational complexity in terms of time and space. We then introduced the four computational models in pursuit; ITTMs, FTMs, NTMs, and QTMs. We identified their computational class, provided their formal definitions, computable languages, connection to the universal computing machine (UTM), and complexity analysis.

The class of quantum and neural computations is still evolving, and several computation models are still materializing. Hence, they are more open questions to address concerning their computational complexity and super-Turing properties non-reproducible by UTM. Other interesting open issues for future work include comparing the computational power of the four models we presented in this paper over arbitrary domains using the conceptual comparison framework formalized by Boker and Dershowitz [2], and similar to Nayak and Dash [21], investigating the gap between infinitary, hyper-computation and super-computation models using a commonly accepted language as input and acceptance timing as the evaluation criterion. Perhaps more interesting than the open issues mentioned above is to use Diaz's approach [9] to describe and hence classify the behavior of the four models based on speed, entropy, and complexity using machine learning. Further, to evaluate current and subsequent super-Turing models, there is a need to define the notion of super-Turing universality and construct a super-Turing universal model to establish as a reference model that can simulate any super-Turing models.

ACKNOWLEDGMENT

This work was supported by NSF under grant agreement DMS-2022448, and the Center for Science of Information (CSoI), an NSF Science and Technology Center, under Grant Agreement CCF-0939370.

REFERENCES

[1] Bernstein, E., Vazirani, U.: Quantum complexity theory. In: in Proc. 25th Annual ACM Symposium on Theory of Computing, ACM, pp. 11–20 (1993)
[2] Boker, U., Dershowitz, N.: How to compare the power of computational models. In: New Computational Paradigms, pp. 54–64. Springer (2005)
[3] Cabessa, J., Villa, A.E.P.: Interactive evolving recurrent neural networks are super-turing universal. In: Artificial Neural Networks and Machine Learning – ICANN 2014, pp. 57–64. Springer International Publishing (2014)
[4] Carl, M.: Effectivity and reducibility with ordinal turing machines. arXiv:1811.11630 [math] (2018)
[5] Carl, M.: Space and time complexity for infinite time turing machines. arXiv:1905.06832 [math] (2019)
[6] Collier, M., Beel, J.: Implementing neural turing machines. arXiv:1807.08518 [cs, stat] (2018)
[7] Deolalikar, V., Hamkins, J.D., Schindler, R.D.: P is not equal to NP intersect coNP for infinite time turing machines. Journal of Logic and Computation **15**, 577–592 (2005)
[8] Deutsch, D.: Quantum theory, the church–turing principle and the universal quantum computer. Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences **400**, 97–117 (1985)
[9] Diaz, N.: Behavior classification for turing machines. Complex Systems **26**(3) (2017)
[10] Fortnow, L.: One complexity theorist's view of quantum computing. arXiv:quant-ph/0003035 (2000)
[11] Gajser, D.: Verifying time complexity of turing machines. Theoretical Computer Science **600**, 86–97 (2015)
[12] Graves, A., Wayne, G., Danihelka, I.: Neural turing machines. arXiv:1410.5401 [cs] (2014)
[13] Grefenstette, E., Hermann, K.M., Suleyman, M., Blunsom, P.: Learning to transduce with unbounded memory. In: Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2, pp. 1828–1836. MIT Press (2015)
[14] Guerrini, S., Martini, S., Masini, A.: Quantum turing machines computations and measurements. arXiv:1703.07748 [cs] (2017)
[15] Hamkins, J.D., Lewis, A.: Infinite time turing machines. The Journal of Symbolic Logic **65**, 567–604 (2000)
[16] Hamkins, J.D., Lewis, A.: Infinite time turing machines. Minds and Machines **12**, 521–539 (2002)
[17] Li, Y.: Fuzzy turing machines: Variants and universality. IEEE Transactions on Fuzzy Systems **16**, 1491–1502 (2008)
[18] Löwe, B.: Space bounds for infinitary computation. In: Logical Approaches to Computational Barriers, pp. 319–329. Springer (2006)
[19] Maruoka, A.: Universality of Turing Machine and Its Limitations. In: Concise Guide to Computation Theory, pp. 161–181. Springer (2011)
[20] Muller, M.: Strongly universal quantum turing machines and invariance of kolmogorov complexity. IEEE Transactions on Information Theory **54**, 763–780 (2008)
[21] Nayak, T., Dash, T.: Quantum finite automata, quantum pushdown automata & quantum turing machine: A study. International Journal of Computer Science and Information Technologies **3**, 5 (2012)
[22] Pan, Y., Zou, T.: On the complexity of turing machine accepting fuzzy language. In: 2012 9th International Conference on Fuzzy Systems and Knowledge Discovery, pp. 112–116 (2012)
[23] Robič, B.: The Turing Machine. In: The Foundations of Computability Theory, pp. 111–153. Springer (2020)
[24] Sharma, A., Upadhyay, A.: Neural hypercomputation: A decisional approach. In: 2016 International Conference on Inventive Computation Technologies (ICICT), vol. 1, pp. 1–6 (2016)
[25] Siegelmann, H.T.: Computation beyond the turing limit. Science **268**, 545–548 (1995)
[26] Siegelmann, H.T.: Neural and super-turing computing. Minds and Machines **13**, 103–114 (2003)
[27] Sipser, M.: Introduction to the Theory of Computation. Cengage Learning, Inc (2013)

[28] Spakowski, H., Thakur, M., Tripathi, R.: Quantum and classical complexity classes: Separations, collapses, and closure properties. Information and Computation **200**, 1–34 (2005)
[29] Tadaki, K., Yamakami, T., Lin, J.C.H.: Theory of one-tape linear-time turing machines. Theoretical Computer Science **411**, 22–43 (2010)
[30] Uchida, Y., Sakamoto, M., Taniue, A., Katamune, R., Ito, T., Furutani, H., Kono, M.: Some properties of four-dimensional parallel turing machines. Artificial Life and Robotics **15**, 385–388 (2010)
[31] Wiedermann, J.: Characterizing the super-turing computing power and efficiency of classical fuzzy turing machines. Theoretical Computer Science **317**, 61–69 (2004)
[32] Winter, J.: Is p = pspace for infinite time turing machines? In: Infinity in Logic and Computation, pp. 126–137. Springer (2009)
[33] Zielenkiewicz, M., Schubert, A., Chrząszcz, J.: On multiply-exponential write-once turing machines. arXiv:1407.7592 [cs] (2014)