Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.12785/amis/080536

Continuous Turing Machine: Real Function Computability and Iteration Issues

Xiaoliang Chen[*], *Wen Song, Zexia Huang and Mingwei Tang*

School of Mathematics & Computer Science, Xihua University , Chen'du 610039, P. R. China

Received: 8 Sep. 2013, Revised: 6 Dec. 2013, Accepted: 7 Dec. 2013
Published online: 1 Sep. 2014

Abstract: Contemporary computer theory is governed by the discretization of continuous problems. Classical Turing machines (TMs) are originally built to solve computation and computability problems, which main feature is discreteness. However, even some simple numerical calculations problems, e.g., iterations in $\mathbb{R}^n$, generate difficulties to be described or solved by constructing a TM. This paper explores the computability of continuous problems by proposing a class of continuous Turing machines (CTMs) that are an extension of TMs. CTMs can be applied to the standard for the precision of algorithms. First, computable real numbers are precisely defined by CTMs and their computations are regarded as the running of the CTMs. CTMs introduce the coded recursive descriptions, machine states, and operations with the characters of computer instructions in essence compared with usual computable continuous models. Hence, they can precisely present continuous computations with the form of processes. Second, the concepts of CTM computable and CTM handleable are proposed. Moreover, the basic concepts on approximation theory such as convergency, metric space, and fixed-point in $\mathbb{R}^n$ are defined in a new space CTM_{R^n}. Finally, an iterative algorithm is shown by constructing a CTM to solve linear equations.

Keywords: Computational mathematics, computer theory, continuous Turing machine, real number computability, iteration.

1 Introduction

The development of computability theory in information science begins with the generation of recursive functions that depend on logical theory. These recursive functions are considered as the precise definitions of intuitive algorithms [1]. Turing describes computations by a class of mathematical machines (theoretical computers), usually called Turing machines (TMs). The machines precisely present the concept of computations with the form of processes by introducing machine states and the operations with respect to the characters of computer instructions. TMs are equivalent to recursive functions. Hence, computability problems are equivalent to Turing computability [2].

A theory is said to be an systematic approach if its deduction and reasoning depend on a standard mathematical model. Computations are model-based processes in the solution of a given calculated problem. However, the existing TM computability theory cannot properly presents computable real functions since TMs are discrete in essence [3,4]. Thereafter, some mathematical models and approaches are developed to analyze the computability of real numbers. Mazur [5] defines computable real functions by the proposed sequence computability. Kreitz and Weihrauch [6,7] take into account the presentation of real number computability by introducing type-2 theory of effectiveness (TTE), which is based on the theory of representations and is an approach of computable analysis. Edalat [8,9,10,11] studies computable real functions by domain theory.

Many constructive analysis methods are also proposed. Moore [12] proposes $\mu-$ hierarchy to interpret recursion theory on reals and constructs flowcharts of continuous time to handle real number computability and halting problems. Doraszelski and Satterthwaite [13] define computable real numbers by the established Markov arithmetic. Blum, Cucker *et al.* [14,15] analyze the computable problems of real numbers by constructing real-RAM models. However, the fact is that not all real number computability that are described above are equivalent. For example, Banach-Mazur computability is not equivalent to Markov computability for computable real numbers [16]. On the other hand, the described

* Corresponding author e-mail: xdxlchen@gmail.com

theories and models are difficult to compatible with the classical model TMs. Hence, this paper attempts to construct a class of extension TM to deal with real number computability.

Numerical analysis involves the methods for real number calculations. However, it does not consider computability problems. Different computational models can obtain inconsistent results on whether a real number problems is computable. For example, a serious distortion or an entire wrong conclusion may be obtained if the real numbers computability is considered by TMs. Hence, discrete machines do not properly demonstrate real number computability. On the other hand, although the theory of numerical analysis makes great achievements in the past, its developments necessarily need a reliable computable theory.

This paper begins with an extension from discrete TMs to continuous TMs. Then, a class of autonomous continuous Turing machines (CTMs) is proposed in section 2. The rationality by using CTMs to explore the computability of real numbers is considered in section 3. CTMs have mainly two strengths compared with the usual models. Firstly, the classic methods of TMs deal with the computations of natural numbers, the sets of natural number, and the arithmetical functions. A CTM covers continuous computations and include discrete computations. Second, it is realistic and feasible since the concept of computable is defined by constructing CTMs. An algorithm is said to be computable if a CTM can be constructed for a certain input to reach an output at finite steps.

CTMs have simple structures, basic operations, and precise descriptions of computations in the form of processes. A CTM series with respect to greater power can be constructed by an iterative or recursive construction of CTMs. Iterative technology based on CTMs is considered in section 4, which demonstrates an approach to prove CTM-computable and to explain how to construct a complex machines. Finally, a typical example is given to illustrate real function computability, which can be regarded as a methodology to solve a class of computable problems. In section 5, we state the results of this paper.

2 Extension: discrete TM to continuous TM

The simplest way for a TM to compute four arithmetic operations is that the representation of numbers only uses '0' [3], where notation '0' is a character in the tape of TMs, which is distinguished with the numerical zero. However, the representation method can lead to the increase of storages. Importantly, by considering iterative computations, a self-iterative TM can difficultly be constructed since computations, e.g., iterations, cannot be easily represented by integers. Many researchers make extensions from discrete models to continuous ones, where continuous automatons, continuous Petri nets, and Hybrid nets *et al* [17,18] are proposed. These models do not means that the number of new models is increase in the series of computational models. Its purpose is to correctly and easily present, solve, and analyze a class of continuous problems. A good approximation generally proves very valuably to solve a complex problem. Therefore, an approximate method is considered in TMs in this paper.

Generally, continuous models are time-related. However, CTMs are regarded as time-independent. This section expands TMs to CTMs by introducing an example for a non-output and two-type nondeterministic TM M_1 (Fig. 1(a)). First, M_1 can be constructed by the following algorithm.

Algorithm for M_1 construction.
TM $\mathbf{M}_1$:="On input $\omega = \omega_1, \omega_2$: // ω_1 and ω_2 represent initial inputs of the two tapes, respectively.
DO {
step1: If there is a '0' on tape1, then move it to tape2 or retain it in tape1 in a random manner.
step2: If there is a '0' in tape2, then move it to tape1 or retain it in tape2 in a random manner.
}while .T." //Notation .T. means that the logical condition of the loop 'while' is always true. □

According to the view of machine computations, a character '0' in M_1 can be considered as a certain mount of resources. Hence, a single resource is represented by a single '0' and several resources are represented by multiple '0' (0^*) in M_1. TMs are theoretical models of computers. In a real-world computer, the number of '0' can be represented as the amount of information. For example, a single '0' can be interpreted as $1G$ information and '00000' as $5G$ information.

Second, a transformation is considered to divide each '0' into k equal parts. This new TM is denoted by M_1^k and shown in Fig. 1(b). The world 'block' is assumed as an unit of '0' in initial configurations. Each block is divided into k. The new unit that is one *k-th* of block is called 'piece'. For example shown in Fig 1, the initial configuration of M_1 (Fig 1(a)) leads to the configuration of M_1^k (Fig 1(b)) in which the resources are expressed in pieces.

Generally, the transition functions of multiple nondeterministic Turing machine (MNTM) have the form

$$\delta : Q \times \Gamma^K \to \boldsymbol{P}(Q \times \Gamma^k \times \{L,R,S\}^k),$$

where Q is the set of states, Γ is the tape alphabet, k is the number of tapes, and $\boldsymbol{P}$ is power set. The expression

$$\delta(q_i, a_1, \cdots, a_k) \in \boldsymbol{P}(q_j, b_1, \cdots, b_k, L, \ldots, R), a_i, b_j \in \Gamma$$

means that if the sate of a machine is q_i and read-write head 1 through k are reading symbols a_1 through a_k, respectively. The machine goes to one of possible states q_j and writes symbols b_1 through b_k. Correspondingly, transition functions directs each head to move left, right,

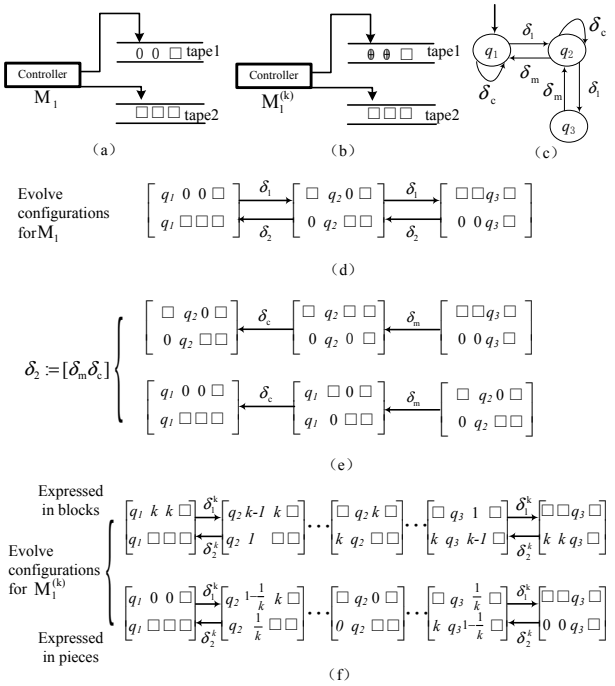

Fig. 1: Transformation of a TM: (a) ordinary TM M_1, (b) transformed ordinary TM M_1^k, represented for k=4, (c) state graph of M_1, (d) evolve configurations for M_1, (e) implied evolve configurations in Fig(d), and (f) evolve configurations for M_1^k.

or to stay put. By considering M_1 as an example and its state graph shown in Fig 1(c), its evolutions contain three types of transition functions:

$$\delta_1 : (q_k, 0, \square) = (q_{k+1}, \square, 0, R, R)$$

$$\delta_m : (q_k, \alpha, \square) = (q_{k-1}, \alpha, \square, L, L), \forall \alpha \in \Gamma$$

$$\delta_c : (q_k, \square, 0) = (q_k, 0, \square, S, S)$$

Symbol $\square \in \Gamma$ denotes that there has not a resource at corresponding positions on the tape, i.e., blank. Now, we consider M_1^k with an strategy for the segmentation of resources. Its transition functions are similar to M_1. However, let the scale of its evolutions be the unit of piece. For example, δ_1, δ_m, and δ_c in M_1 are represented by the following functions combinations:

$$\delta_1^k : \begin{cases} (q_k, \alpha, k-\alpha) = (q_k, \alpha-1, k-\alpha+1, S, S), \ \forall \alpha = 1, 2, \cdots, k \\ (q_k, 1, k-1) = (q_{k+1}, \square, k, R, R) \end{cases}$$

$$\delta_m^k : (q_k, \{\alpha, \square\}, \square) = (q_{k-1}, \{\alpha, \square\}, \square, L, L), \ \forall \alpha = 1, 2, \cdots, k$$

$$\delta_c^k : (q_k, k-\alpha, \alpha) = (q_k, k-\alpha+1, \alpha-1, S, S), \ \forall \alpha = 1, 2, \cdots, k$$

By considering M_1 and its evolved configurations shown in Fig. 1(d), the execution of transition function δ_1 consists of removing a block from tape1 and adding a block to tape2. Correspondingly, by considering M_1^k and its evolved configurations shown in Fig. 1(f), the execution of δ_1^k consists of removing a piece from tape1 and adding a piece to tape2. Hence, the evolved configurations can be expressed in blocks (integer) or in pieces (rational number if k is finite). Let C_i^k be a configuration that is expressed in pieces in M_1^k and $C_i = C_i^k/k$ be the corresponding configuration that is expressed in blocks in M_1. Obviously, the computational processes of M_1 are included in the processes of M_1^k.

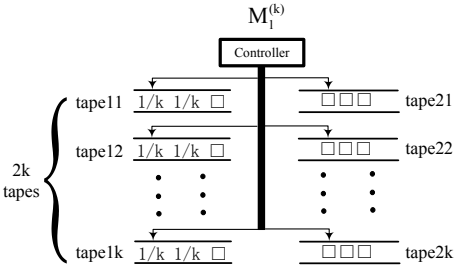

Fig. 2: Structure of 2k-type M_1^K.

The fact that transition functions of a k-type TM execute simultaneously is denoted by $(\delta_i \delta_j \cdots \delta_k)$, where $\delta_i, \delta_j, \cdots$, and δ_k are transition functions from tape i to k, respectively. The structure of M_1^k, which can execute δ_1 k times simultaneously, is shown in Fig. 2. The execution of δ_1 in M_1 equals the execution of $(\delta_1^k \delta_1^k \cdots \delta_1^k)$ in M_1^k. We can change a way to describe transferred processes by introducing some new notations. It facilitates to discuss the extension from discrete TMs to continuous TMs.

Let $[\delta_i \setminus \delta_j \setminus \cdots \setminus \delta_k]$ be a class of orderly executive sequences of transition functions.

Let $[\delta_i \delta_j \cdots \delta_k]$ be a class of synchronized executive sequences of transition functions.

Let $[\delta_i]^\alpha = [(\delta_i)^\alpha]$ be a class of special executive sequences, which performance means that TM implements transition function δ_i total α times simultaneously and removes or adds α piece resources in its tapes to produce a new configuration, where α is a non-negative number.

Fig. 3(a) shows a set of possible transitions of M_1 that are concerned with two block resources. In addition to single execution of δ_1 or δ_2, multiple transitions by the execution of $[\delta_1 \setminus \delta_2]$, $[\delta_1]^2$, and $[\delta_2]^2$ are also represented. The possible transitions of M_1^k for $k = 4$ are shown in Fig. 3(b). M_1^k contains many and finite multiple transitions, e.g., $[\delta_1]^3$, $[\delta_2]^2$ and $[\delta_2]^6$. We apostrophe read-write head and state alphabet 'q_i' for simplification. By observing the execution of $[\delta_1]^3$, its transition process can be expressed in pieces as:

$$\begin{matrix} tape1 \\ tape2 \end{matrix} \begin{bmatrix} 1\ 4\ \square \\ 3\ \square\ \square \end{bmatrix} \xrightarrow{[\delta_1]^3} \begin{bmatrix} \square\ 2\ \square \\ 4\ 2\ \square \end{bmatrix}$$

Or it can be expressed in blocks as:

$$\begin{matrix} tape1 \\ tape2 \end{matrix}\begin{bmatrix} 0.25 & 1 & \square \\ 0.75 & \square & \square \end{bmatrix} \xrightarrow{[\delta_1]^{0.75}} \begin{bmatrix} \square & 0.5 & \square \\ 1 & 0.5 & \square \end{bmatrix}$$

Furthermore, we can also describe in pieces as:

$$(5,3) \xrightarrow{[\delta_1]^3} (2,6)$$

or express in blocks as:

$$(1.25,0.75) \xrightarrow{[\delta_1]^{0.75}} (0.5,1.5)$$

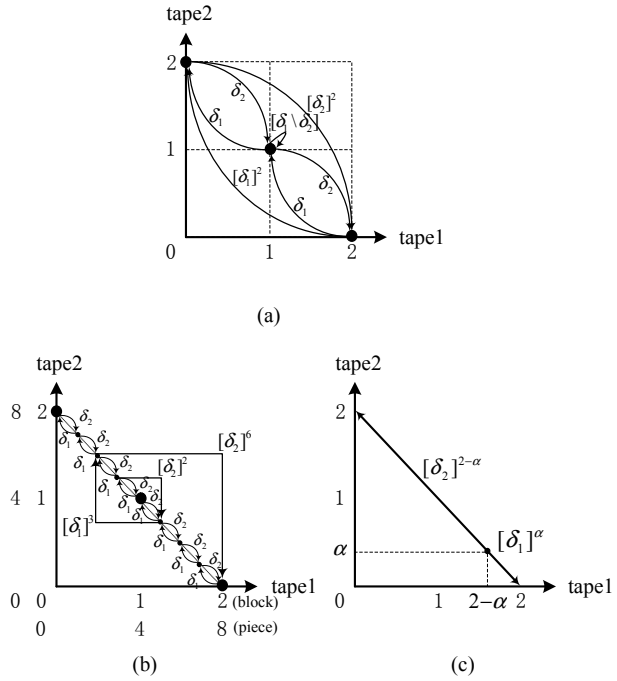

Fig. 3: From discrete to continuous turing machine: (a) graph of resource transition of M_1 in Figure 1(a), (b) graph of resource transition of M_1^k in Figure 1(b) for k=4, and (c) graph of resource transition of M_1^∞, for $k \to \infty$.

The number of possible multiple transitions become infinite if k tends to infinity. These transitions can be denoted by an segment of a line between (2,0) and (0,2) shown in Fig. 3(c). For example the description of the following transition

$$(2-\alpha,\alpha) \xrightarrow{[\delta_2]^\beta} (2-\alpha+\beta,\alpha-\beta),$$

implies that the configurations $(2-\alpha,\alpha)$ and $(2-\alpha+\beta,\alpha-\beta)$ are expressed in pieces. Transition function δ_2 can be executed β times from the configuration $(2-\alpha,\alpha)$ at a moment, where α is any real number in the range [0,2] and β is called a transferred quantity and satisfies the inequality $2-\alpha \le \beta \le 2$. Similarly, if there is an execution of $[\delta_1]^\gamma$, then γ satisfies $0 \le \gamma \le 2-\alpha$. Multiple executions taken a form of $[(\delta_2)^\beta(\delta_1)^\alpha]$ are possible. TMs M_1 and M_1^k discussed above belong to MNTM in essence. The defined machines are recognizers [3] of languages in this paper if there are not special remarks.

Lemma 1.*Two-type nondeterministic TM M_1 has an equivalent five-type deterministic TM M_d.*

Proof. The ideal is make mutual simulations between M_1 and M_d. The fact that M_1 simulates M_d is simple since deterministic TM M_1 is an special case of nondeterministic TM M_d and we only needs to construct a nondeterministic computational branch in M_1.

On the other hand, if M_d is constructed to simulate M_1, M_d needs trying all possible branches of nondeterministic computations of M_1. The machine M_d can be established by constructing five tapes. As shown in Fig. 4, we assume that every tape has a particular function. Tape 1 and 2 are similar to the tapes in M_1. They contain constant strings that copy from the initial inputs of M_1. Tape 3 and 4 are simulation tapes that maintain a copy from the tapes of M_1 for a branch of its nondeterministic computations. The data in tape 3 and 4 contain evolved configurations at the branch. The function of tape 5 is to generate the address string $\omega_{address}$ of nondeterministic computational branches from the length one to length infinite, constantly.

Let Σ^* be an infinite set of all address strings, which contains all possible branches of nondeterministic computations. Any address string consists of finite kinds of alphabets, which are connected with the number of states. By considering M_d, its address strings consist of three kinds of alphabets '1', '2', and '3', which come from the subscripts of the three states of M_1 (Fig. 1(c)). Σ^* is countable according to Cantor's theory since the number of strings in some certain length is finite and the union of denumerable countable sets is a countable set.

A list of Σ^* can be constructed by writing down all strings of length zero, length one, length two and so on. The total number of address strings can be expressed as $\sum_{n=0}^{+\infty} 3^n$. We can easily make a mapping from any string to $n \in \mathbb{N}$. Not all the address strings are valid. For example, address string '212' is valid, which represents that the current state is q_2 which configuration is displayed in tape 3 and 4. The next state is q_1 that is obtained by executing δ_m, where δ_m is stored in controller of M_d. Then, the finally state is q_2 that is obtained by executing stored δ_1. The process can be denoted by $q_1 \xrightarrow{[\delta_m \backslash \delta_1]} q_2$. However, address string '233' is invalid since there not exists a transition function in the process from state q_3 to q_3 in Fig. 1(c). The existence of invalid strings is reasonable since they can be considered as null addresses. Proved process is just a constructive process, we construct M_d as follow:

Algorithm for M_d construction:
TM $\mathbf{M}_d$:="On input $\omega = \omega_1, \omega_2, \omega_3, \omega_4, \omega_5$:
step1: Initial tape 1 and 2. Check correctness of ω_1, ω_2. Tape 3 to 5 stay empty.
step2: Generate a string $\omega_{address}$ in tape 5 according to the rule of the increase of string lengths.
step3: Check the validity of $\omega_{address}$ by checking transition functional grid in controller. If $\omega_{address}$ is valid, go to **step**4, else go to **step**2.
step4: Copy the data in tape 1 and 2 to tape 3 and 4, respectively. Simulate the transition of states from the first alphabet of $\omega_{address}$ to the last one. Go to **step**2." □

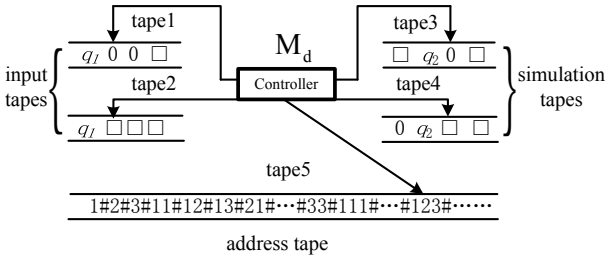

Fig. 4: Structure of five-type deterministic M_d.

Lemma 2.*Five-type deterministic TM M_d has an equivalent single-type deterministic TM M_{sd}.*

Proof. The ideal is make mutual simulations between M_d and M_{sd}. The fact that M_d simulates M_{sd} is simple. We only use any one of the tape in M_d that can simulate the single-tape of M_{sd}. The ideal is to show how to simulate M_d with M_{sd}. Fig. 5 illustrates that single tape can be used to represent five tapes. M_{sd} simulates the functions of M_d by storing their information on its single tape, where we use symbol 'B' as a delimiter to separate different work-spaces. Tape symbol with a small '∧' above it is used to mark the position of the head on the tape. These new symbols have been added to the tape alphabet. In other words, M_{sd} contains virtual work-spaces and heads. we construct M_{sd} as follow:

Algorithm for M_{sd} construction:
TM $\mathbf{M}_{sd}$:="On input $\omega = B\omega_1 B\omega_2 B\omega_3 B\ \omega_4 B\omega_5 \cdots$:
step1: M_{sd} puts its tape into the format and the formatted tape contains $B\hat{0}0\hat{\square}B\hat{\square}\hat{\square}\square B\hat{\square}\hat{\square}\square B\hat{\square}\hat{\square}\square B\cdot\hat{\cdot}$.
step2: M_{sd} scans its tape from the left first 'B' in order to determine the symbols under the virtual heads. Then M_{sd} update its tape according to M_d's transition functions.
step3: When M_{sd} moves virtual heads to the right onto another 'B'. It means that the corresponding head of M_d has been moved onto the blank portion of the tape. Then, it continuous the simulation as before." □

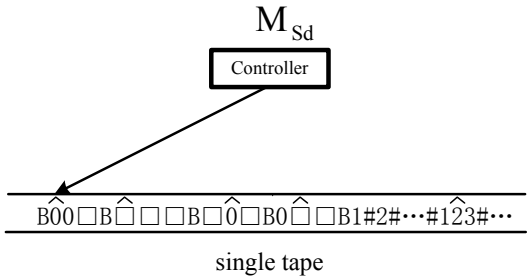

Fig. 5: Structure of single-type deterministic M_{sd} with a snapshot in Fig. 4.

Lemma 3.*Two-type nondeterministic TM M_1 has an equivalent single-type deterministic TM M_{sd}*

Proof. By Lemma 1 and 2, we have the conclusion. □

Lemma 4.*Two-type nondeterministic TM M_1^k has an equivalent single-type deterministic TM M_{sd}^k.*

Proof. Two-type TM M_1^k can be expressed by 2k-type TM M_1^k with an unit of pieces, which is denoted by Fig. 2. We assume that the unit of M_{sd} is also expressed in pieces. By considering the 2k-type structure of M_1^k shown in Fig. 2, according to Lemma 3, the computations in the two tapes of any level are equivalent to the computations of M_{sd}. Hence, the computations of 2k-type M_1^k equal the computations of k M_{sd} machines. M_{sd}^k can be constructed by combining the resources of the corresponding positions of k M_{sd} machines , where resources are expressed in blocks. Hence, M_1^k is equivalent to M_{sd}^k. □

Lemma 5.*Single-type deterministic TM M_{sd}^k has an equivalent single-type deterministic TM M_{sd}.*

Proof. The fact that M_{sd}^k simulates M_{sd} can be achieved by setting the parameter $k = 1$. Using M_{sd} to simulate M_{sd}^k needs little changes about the inputs of M_{sd}. A new wider range delimiter is increased to separate piece work-groups (a piece work-group equals five original work-spaces, which is shown in Fig. 5). The k piece work-groups are combined by copying the initial data of M_{sd} total k times to its single type in order of priority. Then, simulation can be achieved by changing the unit of blocks into pieces and the extended transition functions. □

Corollary 1.*Two-type nondeterministic TM M_1 has an equivalent two-type nondeterministic TM M_1^k*

Proof. By lemma 3, 4 and 5, we have the conclusion. □

Theorem 1.*Single-type deterministic TM M_{sd}^∞ has an equivalent single-type deterministic TM M_{sd}.*

Proof. M_{sd}^{∞} can be constructed by M_{sd}^{k} if k tends infinite. We have its equivalent machine M_{sd} by Lemma 4, where the number of work-spaces tends infinite. Hence, the mutual simulations between them are enabled. Of course, the length of the single tape of M_{sd} is necessarily infinite and M_{sd} is a non-haltable TM in general sense. Note that machine halt in general sense equals classical TM halt [3, 4]. The concept corresponds to machine halt at the sense of convergence of computational histories. For example, a machine is a non-haltable machine if the number of resource divisions tends infinite. It has not acceptive and rejective states in general sense since these states cannot be achieved within a finite time. However, machine halt in convergent meaning of computational histories is to illustrates some important problems in CTMs. □

Theorem 1 guarantees that the division of resources does not increase the power of TMs. Importantly, any TM with resource division has an equivalent ordinary TM no matter how many tapes it has and it is deterministic or nondeterministic. This equivalency necessarily satisfies machine non-haltable in theoretical sense. Actually, the equivalency cannot be guaranteed in real-world hardware environments. For example, infinite division of resources is not possible by the bit restriction of computers.

Hence, a more general situations should be considered. For example, we deal with the machines that are similar to M_{sd}^{∞}, which can machine halt in general sense. These machines can achieve acceptive or rejective states. The judgments of these states should based on the length of computational histories since any simple computation is infinite if k tends infinite. In other words, we consider the approximate calculation of M_{sd}^{∞}.

The classical computational history of a TM is usually considered as the configurations of the TM. However, if real number is tacked, any configuration may has an infinite length. Hence, classical computational history cannot be applied to describe the dynamic behavior of the TM with real numbers. Hence, a class of computational history with respect to state transitions is proposed.

Definition 1.*A string denoted by $\omega_h = ijk\cdots$ is said to be a computational history if the string composed of the subscripts of the existent states, where every two adjacent characters i and j represents that the machine restores the computation path from state q_i to q_j.*

In this case, the computational history of an non-haltable TM is a string with infinite length, even if the TM has finite states. For example, the states and transitions shown in Fig. 1(c) can generate computation history with infinite length. Obviously, the length of computational histories is finite if the machines are haltable since the states of them are necessarily achieve acceptive or rejective states.

Definition 2.*A computational history of M^k is said to be convergent if there exists an absolute difference between the output of M^k and another output of M^k with the length l of computational history, which is less than any given ε.*

Definition 3.*$HALT_{TM} = \{\langle M^K, \omega\rangle \mid M^K$ is a TM that the length of its computational history is finite or its computational history is convergent$\}$.*

Theorem 2.*TM M_{halt}^{∞} has an equivalent single-tape deterministic TM $M_{ordinary}$ (ordinary means that it is a haltable TM).*

Proof. The fact that k tends infinite means that the number of tapes tends infinite in unit piece. If the computational history of M_{halt}^{∞} is convergent, its approximate computations allows that there exists a reasonable length l of computational history for transition functions at the sense of convergency. The evolution of M_{halt}^{∞} achieves terminal states and the machine outputs approximate computational results if the number of state transitions comes to l.

By using Theorem 1, we can find a single-tape deterministic TM M_{sd} and make the length of its tape tends infinite. The single-tape is divided into infinite number of work-groups. Actually, it is just attach the longitudinal k tapes in M_{halt}^{∞} to the single-tape of M_{sd} such that M_{sd} is equivalent to M_{halt}^{∞}.

In this case, if the computational history of M_{halt}^{∞} is convergent, M_{sd} is haltable. The haltable M_{sd} at the sense of convergency is denoted by $M_{ordinary}$. Hence, $M_{ordinary}$ and M_{halt}^{∞} have the same computational results. □

Theorem 3.*M_{sd}^{∞} has not an equivalent TM M_{halt}^{∞}.*

Proof. By using Theorems 1 and 2, we only need to proof that M_{sd} is not equivalent to $M_{ordinary}$. M_{sd} cannot achieve acceptive or rejective states since M_{sd} cannot ensure machine halt. However, $M_{ordinary}$ has terminal states by considering the approximation. Hence, M_{sd} cannot simulate the terminal states of $M_{ordinary}$. Actually, M_{sd} can be regarded as the limiting state of the computations of $M_{ordinary}$. They are not equivalent in the sense of convergence of computational history. □

Theorems 1 and 2 guarantee the rationality of the extension from TMs to CTMs such that CTMs can be formally defined. In this paper, single-tape deterministic CTMs (ordinary CTMs) and the ordinary CTMs at the sense of convergence in computational history are defined. The former are models of computational theory for real numbers and the latter are theoretical models of real functional approximation. Other classes of CTMs are equivalent to them.

Definition 4.*An ordinary CTM is a 7-tuple, $(Q, \Sigma, \Gamma, \delta^{\alpha}, q_1, q_{accept}^{LIM}, q_{reject}^{LIM})$, where*

1. Q is a finite and non-empty set of states.

2. Σ is the set of input alphabets. It comes from $\mathbb{R}_+$ and is expressed in blocks.

3. $\Gamma = \Sigma \bigcup \{\sharp, \wedge\square, \cdots\}$ is the set of tape alphabets, where '$\sharp$' is a symbol of work-space delimiter, '$\wedge$' is a symbol of virtual head, and '$\square$' is blank.

4. δ^{α}: $Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}$ is the transition function, where α is transfer quantity that is defined as a positive rational number.

5. $q_1, q^{LIM}_{accept}, q^{LIM}_{reject} \in Q$ are the initial state, acceptive state and rejective state, respectively.

If we have a CTM and want to construct a CTM algorithm with the form '0' that is similar to '0' in TM, we only need to guarantee that Σ of the CTM comes from the range $(0,1]$ (a subset of $\mathbb{R}_+$). α is defined as a positive rational number since any transfer quantity of transition functions is stored in the controllers of CTMs with the form of data table. In other words, α is a fixed value and is not a variable. Hence, the continuity of CTMs are ensured since the data of computations of CTMs are real numbers.

Definition 5.*A haltable CTM at the sense of computational history convergency is a 7-tuple, $(Q, \Sigma, \Gamma, \delta^\alpha, q_1, q^{LIM}_{accept}, q^{LIM}_{reject})$, where*

1. Q is a finite and non-empty set of states.

2. Σ is the set of input alphabets. It comes from $\mathbb{R}_+$ and is expressed in blocks.

3. $\Gamma = \Sigma \bigcup \{\sharp, \wedge \square, \cdots\}$ is the set of tape alphabets, where '$\sharp$' is a symbol of work-space delimiter, '$\wedge$' is a symbol of virtual head, and '$\square$' is blank.

4. δ^α: $Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}$ is the transition function, where α is transfer quantity that is defined as a positive rational number.

5. $q_1, q^m_{accept}, q^m_{reject} \in Q$ is the initial state, acceptive state and rejective state when the length of computational history is m, respectively.

The length m follows the hardware restrictions or the computational precision demands. By considering Definitions 4 and 5, the machines M^∞_{sd} is an ordinary CTM and M^∞_{halt} is a haltable CTM at the sense of convergency.

A configuration of CTMs is similar to that in TMs, which contains three items: the current state, tape contents, and head position. It is represented by the string 'uqv', where the current state is 'q', the current tape contents are 'u' and 'v' and the current head position is the first symbol of 'v'. Any symbol in 'u' and 'v' comes from $\mathbb{R}_+$.

A configuration is a description of computations of a TM at some moment. The number of configurations in a haltable TM is finite. However, any change of states may contain infinite configurations in a CTM by the influence of the continuous. Hence, the configurations in TMs does not suit to represent the evolution of the computations in CTMs.

Importantly, state graphs cannot express computations of CTMs in detail. Hence, the concept of configuration evolution graphs (CEGs) is proposed to better describe and analyze the configuration change of CTMs. Using CEGs to describe some special CTMs without considering the complex CTM construction algorithms is hence available.

Definition 6.*The CEG of a CTM $(M_{CTM}, C_0^{q_1})$ is a digraph CEG $(M_{CTM}, C_0^{q_1}) = (V, E)$, where $C_0^{q_1}$ means the initial configuration of M_{CTM}, $V = \{C_i | \ C_i$ is a configuration of $M_{CTM}\}$ and $E = \{(C_i^{q_j}, \delta_j^\alpha, (C_i^{q_j})') | \ C_i^{q_j}, (C_i^{q_j})' \in V, C_i \xrightarrow{\delta_k^\alpha} C_i'\}$ are sets of vertexes and edges respectively, where $(C_i^{q_j})'$ is a successor of $C_i^{q_j}$.*

The CEG for CTM M_1^∞ (M_1^∞ comes from M_1^k) is shown in Fig. 6. The infinite configurations and state transition processes can be expressed in simple form with finite elements (vertex, edge, and arc). The initial configuration is denoted by $C_0 = [2 \ \sharp \ 0]^{q_1}$, where q_1 means that C_0 is just in the state q_1, the number of both sides of '$\sharp$' represents the total amount of the resources in the two tapes. The range of changing of them is $\mathbb{R}_+$.

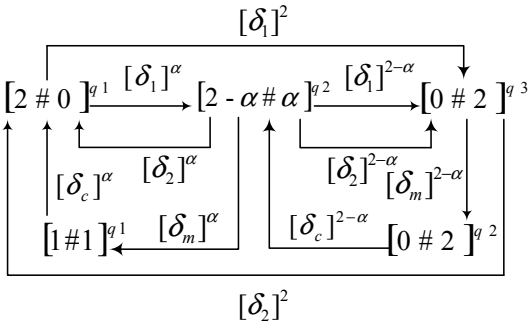

Fig. 6: CEG for CTM M_1^∞ ($M_1^k, k \to \infty$, M_1^k is shown in Fig. 1(b))

However, the moments when different classes of transition functions are executed are only recorded as the vertexes of the CEG. The weights on arcs means transfer qualities executed according to transition functions. The amount of configurations in any two adjacent vertexes with the transfer quality α is infinite. The significance of CEGs for CTMs are as follows:

– Infinite configurations are expressed by finite vertexes.
– The transition function grid in detail from the graph can be obtained.
– The state graphs by folding vertexes and arcs according to superscript of vertexes can be obtained.

3 CTM computable functions

A CTM computes a function by adding the inputs of the function to its tape and halting with the outputs of the function on the same tape at the sense of computational history convergency.

Definition 7.*A function $f : \Sigma^*_{CTM} \to \Sigma^*_{CTM}$ is a CTM computable function if some CTMs for every inputs ω can halt with just $(f(\omega))^*$ on its tape at the sense of computational history convergency, where $(f(\omega))^*$ denotes precise values or theoretical values.*

Definition 8.*A function $f : \Sigma^*_{CTM} \to \Sigma^*_{CTM}$ is a CTM handleable function if some CTMs for every inputs can halt with the unique finite length computational history $(f(\omega))^m$ on its tape by approximation, where $(f(\omega))^m$ denotes the approximate values of length m of computational history.*

The 'unique' in Definition 8 lies on hardware restrictions or approximate precisions. CTM computable functions contain classic TM computable functions since the terminal states of TMs are special situations of the limiting forms of CTMs. Discrete characteristics of TMs make them to achieve the terminal states in finite times.

Any CTM computable function is necessarily a CTM handleable function by Definition 7 and 8. In other words, any CTM computable function can find an approximation function in certain hardware restriction. Verse is not true. The purpose to define CTM handleable functions is to describe the numerical and function approximation problems in $\mathbb{R}_+$. For example, by considering iterative methods for matrix eigenvalues calculation, it may possible for a computer to get an approximate value but a precise value. Computers are hardware limited. It can machine halt through the 'overflow'. However, the computations of the theoretical machine CTM is platform irrelevant , which may not lead to machine halt since the length of computational history may be infinite.

Hence, we cannot to say that the computational processes of eigenvalues are CTM computable since the machine cannot halt in precise value, and we cannot also to say it is not CTM computable since computations always access to precise value. Consequently, the definition of CTM handleable is necessary. If a function is CTM handleable, its limiting evolved states are CTM computable. The concept is special occurs in CTMs and not appear in TMs, which will consider in detail in next section.

CTM computable has two meanings: usual arithmetic operations on reals and the transformations of machine coded descriptions. On the former, the inputs can achieve (not approximation) the outputs according to the computations of CTMs at the sense of convergency. For example, we construct a CTM that takes an input $< r_1, r_2 > r_1, r_2 \in \mathbb{R}_+$ and returns $r_1 - r_2$. Note that the substraction is assumed as true substraction:

$$r_1 - r_2 = \begin{cases} r_1 - r_2 & r_1 \geq r_2 \\ 0 & r_1 < r_2 \end{cases}$$

Let $M_{substract}$ be a CTM that can do this work. Its CEG can be shown in Fig. 7, where $\alpha \in (0, max(r_1, r_2))$ and related transition functions are as follows:

$$\delta_1 : (q_k, m, n) = (q_{k+1}, m-1, n-1, R, R) \;\; m \in (0, r_1], n \in (0, r_2]$$

$$\delta_2 : (q_k, 0, n) = (q_{reject}, 0, 0, R, R) \;\; n \in (0, r_2 - r_1]$$

$$\delta_3 : (q_k, m, 0) = (q_{accept}, r_1 - r_2, 0, L, L) \;\; m \in (0, r_1 - r_2]$$

According to transition directions (the directions of arcs) in Fig. 7, the CTM $M_{substract}$ can achieve terminal states in its limits. Although $(f(\omega))^*$ as a CTM piecewise function has two possible values $r_1 - r_2$ and 0, it will halt with just the value of $(f(\omega))^*$ on its tape. Hence, function f is CTM computable according to Definition 7.

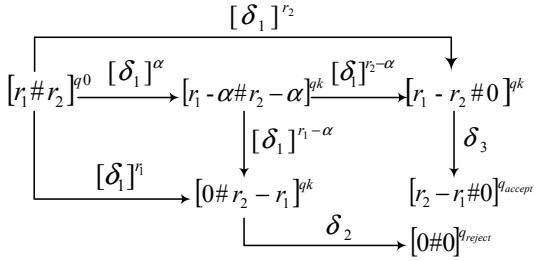

Fig. 7: Configuration evolution graph of $CTM_{substract}$.

On the latter, A TM can get its machine coded description $< TM >$ and can make computations for coded description of other machines by self reference and recursion theorem [3] in classic computation theory. CTMs can also get coded descriptions by similar methods. A CTM computable function is a class of transformation of CTM coded descriptions. For example, a CTM computable function f takes an input $\omega =< M >$ and returns another coded description $< M' >$, which is kept as $f :< M > \overset{\delta}{\to} < M' >$, where M' and M recognize the same language but locate on different configurations. Therefore, whether a function is CTM computable can be proved by constructing a CTM to compute it and returning unique coded strings at the sense of computational history convergency.

Coded descriptions can greatly enhance the described ability of machines. For example, if we proof that a complex function $\varphi = \sum_{i=1}^{n}(R - r_i), R, \forall r_i \in \mathbb{R}_+$ is CTM computable, we only need to proof function $f_1 :< r_1, r_2 > \overset{\delta}{\to} < r_1 - r_2 >$ is CTM computable (the conclusion is proved by the construction of $M_{substract}$), then we proof the function $f_2 :< M_{substract1}, M_{substract2} > \overset{\delta}{\to} < M_{substract1} + M_{substract2} >$ is CTM computable (it is easily to construct M_{add} since adding and subtraction are similar). Finally, we construct coded function $\varphi :< f_1, f_2 >$. If there exists a CTM with the inputs of $< f_1, f_2 >$ that can outputs the unique string by the transition functions of φ, function φ is CTM computable.

Actually, substraction of real is CTM computable, addition is the opposite of the operation of subtraction, and multiplication is the result of continuous additions. Hence, function φ is CTM computable. There are many works that are not CTM computable. Divergent iterative process is an example.

Mapping reducible at the sense of CTMs is considered whcih can extend the proof directions of CTM computable.

Definition 9.*CTM $< M_A >$ is mapping reducible to CTM $< M_B >$, written $< M_A > \leq_m < M_B >$, if there is a CTM computable function $f : \Sigma^*_{CTM} \to \Sigma^*_{CTM}$ such that for every ω,*

$$< M'_A, \omega' >_{input\ \omega} \Leftrightarrow < M'_B, f(\omega)' >_{input\ f(\omega)}$$

The function f is called the reduction from machine $< M_A >$ to $< M_B >$, where the meaning of $< M'_A, \omega' >_{input\ \omega}$ is that the coded machine $< M_A >$ computes the string ω (ω can be a function) and generates a new string ω' on its tape when $< M_A >$ halt at the sense of computational history convergency. M'_A and M_A locate in different configurations but recognize the same language.

Actually, Definition 9 is recursive. It is possible to construct a CTM to compute or decide a reductional function f. For example, $< M_A >$ and $< M_B >$ in Definition 9 may be the machines to computing another reductional functions. If a CTM computable problem is reducible to another problem, which is proved if the original problem is CTM computable. The following theorem can illustrates this ideal.

Theorem 4.*Let $< M_A >$ and $< M_B >$ be machines to compute functions f_1 and f_2, denoted as $< M_A, f_1 >$ and $< M_A, f_2 >$, respectively. Function f_1 is CTM $< M_A >$ computable if $< M_A > \leq_m < M_B >$ and $< M_B >$ is CTM computable.*

Proof. We assume that $< M_B, f_2 >$ is CTM $< M >$ computable and function f is the reduction from $< M_A >$ to $< M_B >$. We construct a new CTM N to compute $< M_A, f_1 >$ such that machine N can halt with just code $<< M'_B, f'_2 >^* >$ on its tape.

Algorithm for CTM N construction:
CTM **N**:="On input $\omega =< M_A, f_1 >$:
step1: According to reductional function f, we have $f(\omega) =< M_B, f_2 >$. Run machine $< M_B, f_2 >$ and output $<< M'_B, f'_2 >^* >$ if $< M_B, f_2 >$ machine halt at the sense of limit.
step2: Run machine $< M >$ with the input $<< M'_B, f'_2 >^* >$ and output whatever $< M >$ outputs."

Obviously, if $< M_A > \leq_m < M_B >$ and f_2 is CTM $< M_B >$ computable, then f_1 is M_A computable. □

An example of the proof for CTM computable by using reduction is proposed. By considering a function $f_1(x) = x,\ x \in [-1, 1]$, we construct CTM M_{f1} as follow:

Algorithm for CTM M_{f1} construction:
CTM $\mathbf{M}_{f1}$:="On input $\omega = x$:
Output x, then accept." □

Obviously, f_1 is a CTM computable function. Let another function be $f_2(y) = \cos(y),\ y \in [0, \pi]$ which computable property is not clear and the work of constructing a CTM to compute 'cos' is also complicated. We assume such a CTM is existent and is denoted by M_{f2}. The mapping from f_2 to f_1 is clearly exist by their geometric meanings such that for any $y \in [0, \pi]$, we have

$$< M'_{f2}, y' >_{input\ y} \Leftrightarrow < M'_{f1}, f_2(y)' >_{input\ f_2(y)}$$

Then, function $f_2(y) = \cos(y),\ y \in [0, \pi]$ is also CTM computable according to Theorem 4.

4 Iterative technology based on CTMs

The concept of iterations appears in computational mathematics and the theory of programming. Almost all high-level programming languages support iterations. The solution of many mathematic approximation problems needs iterations. However, iterations in $\mathbb{R}$ are not TM computable. In real-world, computers can handle approximate and iterative computations in $\mathbb{R}$. Hence, TMs should have an ability to do and do better the works since TMs are a class of platform-independent theoretical models. However, their description abilities do not match usual computers. It is unreasonable that TMs are models of this class of computations. The proposed CTMs are the expansions of TMs, which can deal with these issues. Therefore, the computable properties of iteration problems under CTMs are discussed in this section, which can as a classical example of CTM applications.

The process of iteration in numerical calculations is considered. It takes an initial point and an iterative formula. Then, let the obtained solution be the next initial point and the process keeps iteration until the fixed point is approximated. The computation ends when the adjacent approximative solutions satisfy a precision requirement.

This section introduces CTM to compute the iteration problems of system linear equations, which can make better understanding for the significance of CTM computable compared with considering simple iteration problems. The key problems to prove iteration computability is that the concept of convergence and a type of fixed-point theorem can be described by CTMs. On the former, we define n-dimensional real value vector as the input of CTMs. All these CTMs constitute a space. Then, the concepts of distance, convergence can be defined. On the latter, we can use CTM recursion theorem to get the definitions.

It is necessary to ensure convergence and fixed-point theorem. If the computations of a CTM $M_{iteration}$ are convergence and satisfy the fixed-point theorem, the iteration process is CTM $M_{iteration}$ handleable. Its limit case is CTM $M_{iteration}$ computable. Otherwise, the iteration process is neither $M_{iteration}$ computable nor $M_{iteration}$ handleable.

Theorem 5.*(Recursion theorem in CTM) Let T be a CTM that computes a function* $t : \Sigma^*_{CTM} \times \Sigma^*_{CTM} \to \Sigma^*_{CTM}$*, then there exists a CTM U that computes a function* $u : \Sigma^*_{CTM} \to \Sigma^*_{CTM}$ *for every* ω*, such that*

$$u(\omega) = t(< U, \omega >)$$

The proof is is abbreviated since it is similar to the recursion theorem in TMs [3]. Recursion theorem indicates that CTMs can output the descriptions of themselves and continuous perform a computation by these descriptions. Hence, any complex CTM can be described by recursive coded methods. Theorem 5 is the basis of the following definitions.

'ω' is used to denote a possible string in a CTM. For any $x = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$, there exists a CTM M that takes $\omega = x_1 x_2 \ldots x_n$ as the input, which is written as $< M, \omega >$. All of these CTMs constitute a new complete space that is called $CTM_{\mathbb{R}^n}$ space.

The concept of distance in $CTM_{\mathbb{R}^n}$ space is similar to Euclidean distance. For any strings $\omega_1 = x_1 x_2 \ldots x_n$ and $\omega_2 = y_1 y_2 \ldots y_n$, we can obtain $< M, \omega_1 >, < M, \omega_2 > \in CTM_{\mathbb{R}^n}$. The distance in $CTM_{\mathbb{R}^n}$ space is defined as follows:

$$\rho(< M, \omega_1 >, < M, \omega_2 >) =< M_{sqrt}, <$$

$$M_{sum(i=1\ to\ n)}, < M_{square}, < M_{sub}, x_i - y_i >>>> .$$

If $x, y \in \mathbb{R}$, we have the distance

$$\rho(< M, \omega_1 >, < M, \omega_2 >) =$$

$$< M_{abs}, < M_{sub}, x - y >> .$$

It is easy to prove that they satisfy distance axioms.

Definition 10.*Let* $CTM_{\mathbb{R}^n}$ *be a metric space and* $< M, \omega_n >$, $n = 1, 2, \ldots$ *be CTMs in* $CTM_{\mathbb{R}^n}$ *space. A CTM* $< M, \omega_n >$ *is convergent to* $< M, \omega >$*, written as*

$$\lim_{n \to \infty} < M, \omega_n >=< M, \omega >,$$

if $\rho(< M, \omega_n >, < M, \omega >) \to < M, 0 >$

Definition 11.*Let* $CTM_{\mathbb{R}^n}$ *be a metric space. Function* $< M_T, \omega >: CTM_{\mathbb{R}^n} \to CTM_{\mathbb{R}^n}$ *is a CTM with the ability of contraction (it is similar to the contraction operator in mathematic) if* $\exists\ \theta$, $0 \leq \theta < 1$, $\forall\ < M, \omega_1 >, < M, \omega_2 > \in CTM_{\mathbb{R}^n}$*, we have*

$$\rho(< M_T, < M, \omega_1 >>, < M_T, < M, \omega_1 >>) \leq$$

$$< M_{mul}, \theta\ \sharp \rho(< M, \omega_1 >, < M, \omega_2 >) >$$

Definition 12.*Let* $CTM_{\mathbb{R}^n}$ *be a metric space.* $< M_T, \omega >: CTM_{\mathbb{R}^n} \to CTM_{\mathbb{R}^n}$*. CTM* $< M, \omega^* >$ *is called the fixed-point of* $< M_T, \omega >$*, if there exists a CTM* $< M, \omega^* > \in CTM_{\mathbb{R}^n}$ *such that*

$$< M, \omega^* >=< M_T, < M, \omega^* >>$$

Theorem 6.*(Fixed-point theorem in CTM) Let* $CTM_{\mathbb{R}^n}$ *be a complete metric space,* $\exists < M_T, \omega > \in CTM_{\mathbb{R}^n}$*, CTM* $< M_T, \omega >$ *possesses unique fixed-point such that*

$$< M, \omega^* >=< M_T, < M, \omega^* >>$$

The proof is abbreviated since it is similar to that of the fixed-point theorem in mathematic.

This section introduces an example of solving a linear system equation

$$\sum_{j=1}^{n} a_{ij} x_j = b_i,\ i = 1, 2, \ldots, n$$

by constructing a CTM $M_{iteration}$ to compute it. The equation can be denoted by the form

$$x_i = \sum_{j=1}^{n} (\delta_{ij} - \alpha_{ij}) x_j + b_i,\ i = 1, 2, \ldots, n.$$

If $m_{ij} = \delta_{ij} - \alpha_{ij}$ is hold, the iterative scheme can is represented as

$$x_i^{(k+1)} = \sum_{j=1}^{n} m_{ij} x_j^{(k)} + b_i,\ i = 1, 2, \ldots, n,$$

$$k = 0, 1, 2, \ldots$$

The CTM $M_{iteration}$ can be constructed recursively by Theorem 5. Firstly, we construct CTM_{sub} (it is similar to $M_{substract}$) to compute m_{ij}. Second, CTM_{sub} is constructed to compute $m_{ij} x_j^{(k)}$. Finally, CTM_{sum} is introduced to deal with $\sum_{j=1}^{n} m_{ij} x_j^{(k)} + b_i$. Machine $M_{iteration}$ can be executed by inputting the coded descriptions of CTM_{sub}, CTM_{sub}, CTM_{sum}, and their corresponding inputs on the tapes of $M_{iteration}$. Fig. 8 shows the structure of $M_{iteration}$.

Proposition 1.*Iterative computation to solve system linear equations is CTM handleable if it satisfies Theorem 6 (Fixed-point theorem in CTM).*

Proof. If the iterative computation does not satisfy fixed-point theorem, then the computation may divergent within two situations. First, the computational history may be finite by embedding a controllable CTM such that $M_{iteration}$ can machine halt at a given length. However, the fact violates the meaning of approximation.

Second, many convergent values lead to different approximate values, which can generate different computational histories. Hence, the iterative computation is not CTM handleable by Definition 8. The proposition is proofed. □

Actually, if the iterative computation satisfies fixed-point theorem, related results are necessarily convergent, uniqueness, and reasonable approximation. According to algorithm 6, if halting judgment condition ε is given, $M_{iteration}$ is necessarily machine halt and the length of the unique computational history is finite. According to Definition 8 (CTM handleable), iterative computation to solve system linear equation is $M_{iteration}$ handleable.

Algorithm for $M_{iteration}$ construction:
CTM $\mathbf{M}_{iteration}$:="On input $\omega = \omega_1, \omega_2, \omega_3, B, \omega_5, \omega_6, \omega_7, B, \omega_9$ ('B' means no initial input on the tape):
step1: Copy In/Output tape to d-tape3 (save $x^{(k)}$), delete checked symbol '•'.
step2: Format c-tape4, copy In/Output tape to c-tape4, delete checked symbol '•', add checked symbol '∧' on x_1(for middle computation).
step3: In c-tape1, run machine $< CTM_{sub} >$ with the data of c-tape1 and d-tape1 by checked symbol '∧' (compute m_{ij}). Copy the result $< CTM'_{sub} >$ and current x_j in c-tape4 to c-tape2 and separate them with symbol '♯'.
step4: In c-tape2, run machine $< CTM_{mul} >$ with the right information (compute $m_{ij} \cdot x_j^{(k)}$). Copy the result $< CTM'_{mul} >_j$ to c-tape3, the subscript j comes from x_j. Initial c-tape2.
step5: If not move '•' in In/Output tape to blank, go to **step**6, else go to **step**7.
step6: If '∧' in c-tape4 does not removed to blank, then consider c-tape1, c-tape4, and d-tape2, move '∧' one step to the right, go to **step**3, else copy b_i in d-tape3 to c-tape3 by checking '•'. In c-tape3, run machine $< M_{sum} >$ with the right information (compute $x_i^{(k+1)} = \sum_{j=1}^{n} m_{ij} x_j^{(k)} + b_i$). Replace current x_i in In/Output tape with the result $< CTM'_{sum} >$. Move all current '•' one step to the right. Initial c-tape3. The symbols '∧' in c-tape1 and d-tape1 are moved one step to the right and locate on the right first symbol of '$'. Go to **step**2.
step7: Copy In/Output tape and d-tape3 to c-tape5 and run $< M_{jud} >$. If the result $< M'_{jud} >$ satisfies the precision requirement ε, output In/Output tape (display approximation $x^{(k+1)}$), then machine $M_{iteration}$ halt, else initial all tapes except In/Output tape, go to **step**1." □

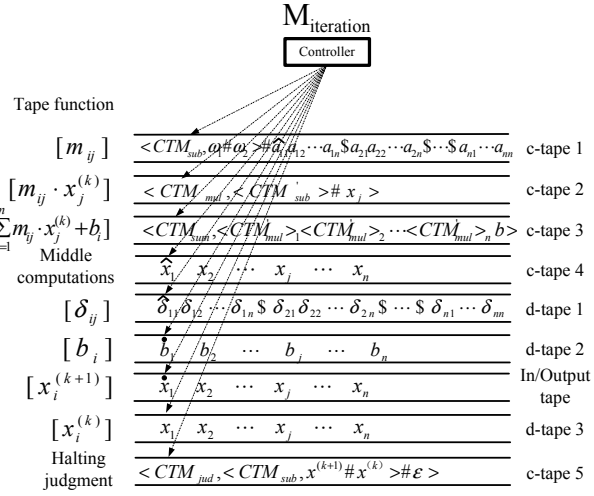

Fig. 8: Structure of CTM $M_{iteration}$.

Proposition 2.*If iterative computation to solve system linear equations is CTM handleable and $M_{iteration}$ can machine halt with $\varepsilon = 0$, it is CTM $M_{iteration}$ computable.*

Proposition 2 is obvious and the proof is abbreviated.

5 Conclusion and future works

This paper deals with a class of time-independent continuous Turing machines (CTMs) by extending from homologous discrete counterpart to continuous. Some equivalent proofs between CTMs and TMs and the constructive algorithms in the sense of limitations are given. The important concepts such as CTM, CTM handleable, CTM computable are defined, which provide an unified framework of computable theory for real numbers (real functions) computations.

If the state graphs of TMs are using in CTMs, the fact will generate infinite configurations and the incomplete information descriptions. Hence a graph based representation methods CEGs are proposed, which can describe infinite configurations in CTMs by finite elements and can effectively present the behaviors of CTMs.

On the other hand, machine description with recursive methods is proposed and a class of machine space $CTM_{\mathbb{R}^n}$ is defined to discuss the approximate problems in $\mathbb{R}^n$. Then, some basic concepts of approximations such as convergence of machine computation, distance in machine space $CTM_{\mathbb{R}^n}$, and Fixed-point theory in CTM etc, are given. Finally iteration approximate computations in solving system linear equations are proved to be CTM handleable or computable.

Future works will consider the complexity theory in CTMs. Researchers have discovered an elegant scheme to classify a problem according to there computational difficult. Hence, we will use CTMs as measure tools to analysis algorithmic time complexity and space complexity.

Acknowledgements

This research was supported in part by the National Nature Science Foundation of China (No. 61271413), Fundamental Research Funds for Science and Technology Department of Sichuan Province (No. 2013JY0089), Fundamental Research Funds for Educational Department of Sichuan Province (No. 08ZA029), key Scientific Research Fund of Xihua University(No. z1312624), and key Scientific Research Fund of Xihua University (No. Z1222623).

References

[1] D.P. Yang and A.S. Li, Theory of conputability. Beijing: Science press, (1975).

[2] Z.Z. Zhao, Introduction to computational science. Beijing: Science press, (2002).
[3] M. Sipser, Introduction to the Theory of Computation, Boston: Course Technology Inc, (2005).
[4] J. Hopcroft, R. Motwani, and J. Ullman, Introduction to Automata Theory, Languages, and Computation. Boston: Addison Wesley, (2006).
[5] S. Mazur, A. Grzegorczyk, and H. Rasiowa, Computable analysis. Pa'n stwowe Wydawn. Naukowe, (1963).
[6] C. Kreitz and K. Weihrauch, Theoretical computer science, **38**, 35-53 (1985).
[7] K. Weihrauch, Computable analysis: An introduction. Springer Verlag, (2000).
[8] A. Edalat, Information and Computation, **120**, 32-48 (1995).
[9] A. Edalat, Theoretical Computer Science, **151**, 163-193 (1995).
[10] A. Edalat, Bulletin of Symbolic Logic, **3**, 401-452 (1997).
[11] A. Edalat and P. Snderhauf, Theoretical Computer Science, **210**, 73-98 (1999).
[12] C. Moore, Theoretical Computer Science, **162**, 23-44 (1996).
[13] U. Doraszelski and M. Satterthwaite, The RAND Journal of Economics, **41**, 215-243 (2010).
[14] L. Blum, F. Cucker, and S. Smale, International Journal of Bifurcation and Chaos, **6**, 3-26 (1996).
[15] L. Blum, Complexity and real computation. New York: Springer Verlag, (1998).
[16] P. Hertling, Annals of Pure and Applied Logic, **132**, 227-246 (2005).
[17] L. Recalde, E. Teruel, and M. Silva, Application and Theory of Petri Nets, **1639**, 107-126 (1999).
[18] R. David and H. Alla, Discrete, Continuous, and Hybrid Petri Nets, Berlin: Springer-Verlag, (2005).

Xiaoliang Chen received B.S. and M.S. degrees from Xihua University, Chengdu, China, in 2007 and 2010, respectively. He is currently a Ph.D. student in School of Electro-Mechanical Engineering, Xidian University, Xi'an, China. His research interests include supervisor control and fault diagnosis of discrete event systems.

Wen Song is a professor of Xihua University since 2005. senior member of Petri net Special Commission. His main research interests are theory of Petri nets and mathematical logic.

Zexia Huang received B.S. and M.S. degrees from Xihua University, Chengdu, China, in 2007 and 2010, respectively. She received Ph.D in Capital Normal University, Beijing, China. Her research interests include function approximation theory and mathematical analysis.

Mingwei Tang is an associate professor with the School of Mathematics and Computer Science Technology of Xihua University. He received a Ph.D. degree at the School of Computer Science and Engineering from University of Electronic Science and Technology of China in 2011. His current research interests include network security and information hiding.