A Novel Computing Model Supporting Cooperation

LIU SHAOHUA, WEI JUN
Technology Center of Software Engineering, Institute of Software,
The Chinese Academy of Sciences, Beijing 100080, P.R. China
E-mail: liushaohua@ios.cn

Abstract

As computing technologies are applied deeper and wider, their basic models are evolving towards stronger expressiveness and higher naturalness. Because the cooperation is ubiquitous in the real life, the focus of computation is turning from individual entities to dynamic coordination across them. Accordingly, future computing would be based on cooperation.

This research proposes a novel model of computation, namely Cooperative Turing Machine (CTM). It extends classical Turing Machine (TM) by introducing a teamwork tape, which is shared by all participants within a team, so as to achieve cooperation with teammates during computation.

Some formal aspects of CTMs including its co-inductive recursive function and behavioral streams language are defined in this paper. Correspondingly, many new features such as late binding, teamwork sensitive, non-determination, highly situated are discussed. Based on these fundamental notions, we also explain its expressiveness, and finally put it into system practice involving service cooperation.

We prove that CTMs are more expressive than TMs. Thereupon the CTMs would leverage traditional computation and usher us into the new era of cooperation.

Categories and Subject Descriptors: F.1 [Theory of Computation]: Computation by Abstract Devices; H.5.3 [Group and Organization Interfaces]: User Interfaces - Collaborative computing; Computer-supported cooperative work; Theory and models; I.1.3 [Computing Methodologies]: Symbolic and Algebraic Manipulation

General Terms: Design, Theory, Algorithms

Additional Key Words and Phrases: Cooperative Turing Machine, teamwork tape, recursive function, behavioral streams, expressiveness, service cooperation
1. INTRODUCTION

The greatest achievement of computer science in its fast evolution from early abstract Turing Machine to its present status is the cognition and improvement of the computation models. Abstract automata such as finite state automata (FSAs), pushdown automata (PDAs), linear-bounded automata and Turing Machines (TMs) are theoretical tools for modeling different kinds of computation, more specifically, algorithmic computation [1]. In common, an environment, which is independent of the machine, constructs a sequence of input. Thereupon the machine begins to read and process the input, finally produces an output sequence. Note that it would shut out the external world during computing. Over a long period of time, this closed schema has predominated in most mainstream fields, including software engineering, artificial intelligence, signal processing, virtual reality, etc.

However, computation in the real world today increasingly interweaves input and output, where the input may depend on the previous output, and often composes individuals to work together, where the output may be affected by the teamwork configuration in a group, either.

For that reason, many scholars have pointed out the classical algorithmic paradigm ability not equal to its ambition, because the TM-computable function may no longer be fully appropriate to deal with all features of present-day computing [2], such as component composition, service cooperation, business process integration, multi-agent coordination, computer supported cooperative work (CSCW), cross-organizational collaboration, Peer to Peer (P2P), Grid and so on. Their common feature is the sociality, that is, individual behaviors involve the collective efforts, in particular, concurrent teamwork and nondeterministic interaction. A theory of concurrency and interaction requires a new conceptual framework, not just a refinement [3]. Meanwhile, it has been proven that non-terminating interactive and reactive systems like operating systems can not be modeled by Turing Machine [4].

Thus, much work had to regard especially multiple interactions as single one, and then explored the interactive computing devices under such simplified framework. Subsequently, some researchers suggested that the interactive models of computation would be more expressive than “algorithmic” ones such as Turing machines [5].

Although the interactive machine proposed can depict well some modern systems whose hallmarks are interactive and reactivity [6], it still falls short when it comes to modeling the cooperative behaviors affected by the team (workmates) between the individual and (observer) environment. Fortunately, such interaction framework has already established a foundation for the next better solution.
Cooperation is ubiquitous in real life. In fact, many problems are too complicated to be solvable if we insist that all work must be strictly done just by an individual. Moreover, most cooperation problems are more naturally described via teamwork rather than individual treatment. Therefore, this paper proposes Cooperative Turing Machines (CTMs) that extend the classical TMs to address the situation of teamwork.

Cooperative Turing Machine (CTM) is an abstract model for cooperative computation, based on co-inductive semantics with effective constructability. It is a kind of multi-tape Turing Machine with a common work tape which contents allow to be shared and co-operated by partners within the same team, thus this tape is also called the teamwork tape.

Thereby, CTMs can provide dynamic stream based semantics for cooperative computing entities such as web services, business processes, and intelligent agents and so on. CTMs model computation as an open and cooperative process in which the individual, teammates or colleagues, and external observers can participate together, in the way that cannot be captured by past models.

2. RELATED WORK

Some founders of the theory of computation, including Alan Turing and Edward Moore, gave attention to cooperation involved in the abstract machine model. For example, Moore explicitly outlined his Moore machine that could produce an output symbol upon each input symbol, which would be used within teamwork settings in some sense, such as his Gedanken-Experiments [7]. Earlier, Turing also sketched out a “choice machine” as an alternative to his “automata” algorithmic machine, where after the machine had started according to an environmental input an operator could participate into the computation process [8]. Regrettably, they had not proceeded further with theory for such kind of machine.

Subsequently, these soft touches have basically been ignored by most computer scientists. It is not because these ideas lack any merit but because the whole research community in the past lacked some proper conceptual tools to make progress on them.

Until now, it has become clear gradually that inductive approaches adopted in Turing Machine to reasoning and modeling are not sufficient to model all today’s computation. In recent years, some research work have been using co-inductive and co-algebraic for defining dynamic behaviors of emerging interactive, reactive and cooperative computation in order to escape the “Turing Tarpit” of algorithmic computation [9].

Most state machines, like finite state automata, typically associate state transitions with input tokens so that they could change from one state to another as the input is processed step by step. Such algorithmic models can
not depict well the dynamic behaviors of some individual who is cooperating with other fellows inside the same teamwork, meanwhile interacting with observers from outside environment.

Finite State Transducers (FSTs) are a class of the simplest state machines that can model cooperative computation. Moore’s machine is such an example; it associates states with output characters which are distilled continually during the computation, meanwhile other partners can participates in the cooperative process. A Mealy machine is another model of an FST. But it associates output characters with transitions rather than with states, so as to form transition labels consisting of (input, output) pairs.

Both Mealy and Moore machines’ behaviors are characterized by a stream of pairs of characters. Furthermore, the input and output of FSTs are both handled dynamically in a computational process that needs no termination point. Thus, these machines can model very primitive cooperative computation. In contrast, as we know, FSAs just associate states with two values that is Boolean accept/reject, which are only observed finally by external environment once the computation terminates. Apparently, FSTs are more expressive than before.

If both the number of states and the number of transition labels are allowed to be infinite, the resulting computational model is known as a labelled transition system (LTS). This is a standard abstraction for many computing upon or beyond Turing Machine, including cooperative computation process modeling.

3. COOPERATIVE TURING MACHINE

In this section, we extend TMs by introducing cooperation context, and by ranging new generated open devices’ computational semantics from finite atomic strings to infinite streams comprising these strings, and by subdividing the input/output into two aspects: the interaction observable with external environment and the cooperation sensible within a team. The resulting device is an abstract machine model for cooperative computing.

3.1 Sharing a Teamwork Tape

Even to perform simple collaboration reliably, any individual has to ensure that the parties within teamwork agree on current cooperation context, such as what the team has experienced and where they desire to transit its state. This requires elements of teamwork through (1) transmitting cooperative tokens, (2) communicating cooperation context, and (3) co-operating some shared medium. In summary, it necessitates a common work space shared by all participants. That is to say, teammates need participate reading existing cooperation
sequences from this medium, or writing a new token to it. With respect to this paper, we take a common teamwork tape shared by participants as such cooperative medium.

**Definition 1 (Cooperative Turing Machine):** A Cooperative Turing Machine (CTM) is a multi-tape Turing Machine (TM) with a shared teamwork tape whose contents are co-operated by others between successive TM computations.

Besides a conventional private I/O tape that is completely occupied by itself meanwhile widely observed by external environment, the CTM has also a special additive tape and corresponding read/write head, which is only sensed by a smaller teamwork group instead of whole external world, as illustrated in Fig.1.

A CTM’s cooperative teamwork tape can be viewed as its shared memory, data storage, or tuple space; its content before and after (a single TM) computation is obviously a part of any CTM’s computation, so it is also called the CTM’s partial state.

Dissimilar to past machines, the state transition in a CTM is not always fired by the exogenetic environmental input and ended with the output observable to external world. Due to the existence of the shared teamwork tape that allow ingoing and outgoing cooperation tokens, the state transition in a CTM may be triggered by some endogenetic cause and issue some internal changes sensible only within the same team. In the context of distributed web service cooperation, for example, the trigger may be an outsourcing request that is a
part of e-business processes, and the issue may be some message delivery on a shared data space co-operated by allied service providers.

From the viewpoint of any individual machine, the exogenetic environmental input and the endogenetic causes could be seen as an integrated input to a transition, whereas the output observable and the internal changes sensible could be seen as an integrated output from a transition. On the other hand, in the view of an external observer, the endogenetic cause and internal changes are also a part of inside states about which they don’t concern. As for this paper, in order to explain it more clearly, we separate the cooperation context independent from either the external interaction or the internal state, that is, the cooperation is an explicit component distinguishing from the input, output and state in the automata expression.

3.2 Function-based CTM Model

Since the shared teamwork tape at the beginning of a CTM computation step is not always the same, the output string $O$ of a CTM $M$ at the end of the computation step depends not only on the input string $I$, and the internal state $S$ reflecting its autonomous status, but also on the shared teamwork tape content $C$ that is the cooperation context of the teamwork. As a result, $M$ defines a partial recursive function $f_M$ from (input, context, state) triple to (output, context, state) triple of strings. Thus a CTM over an alphabet $\Sigma$ can be represented by a triple $(f_M, \Sigma, \text{State})$, where $f_M$ comprises three components, viz. $\text{out}$, $\text{help}$ and $\text{next}$ corresponding to outputting, cooperating and transiting respectively.

More formally, CTMs are defined as infra:

**Definition 2 (CTM):** A CTM is a 7-tuple $(S, C, I, O, \text{out}, \text{help}, \text{next})$, where

(i) $S$ is the set of individual partial states;

(ii) $C$ is the set of teamwork partial states, or the set of cooperation tokens;

(iii) $I/O$ is the set of input/output tokens;

(iv) $\text{out/help/next}$ is the mapping from $S \times C \times I$ into $O/C/S$, i.e. $\text{out/help/next} : S \times C \times I \rightarrow O/C/S$, as illustrated in Fig.2.

If $S, C, I,$ and $O$ are finite, the CTM is finite. If there is a string-based encoding of the elements of $S, C, I,$ and $O$ such that $\text{out}, \text{help}$ and $\text{next}$ are Turing-computable, then the CTM is effective.

If $\text{out}, \text{help}$ and $\text{next}$ are functions, then the CTM is deterministic; else if they are relations, then the CTM is non-deterministic. In this paper, we explain most CTM notions in the deterministic form.
The \textit{out}, \textit{help} and \textit{next} are applied to sequences of input tokens and ingoing cooperation tokens. Then \textit{out} returns the resulting sequence of output tokens to the external environment; \textit{help} writes outgoing cooperation tokens to the shared teamwork tape; \textit{next} gives the device itself next new state. On the basis of these three, we also define an additional mapping \textit{finale} that gives us the resulting state if we start in \(M\)'s initial state and corresponding teamwork state at that time.

\textbf{Definition 3 (Functions of CTM):} Let \(M=(S, C, I, O, \text{out}, \text{help}, \text{next})\) be a CTM. For each \(s \in S, c \in C, i \in I, \) and \(x \in C^*, y \in I^*\)

\[
\begin{align*}
\text{out}(s, cx, iy) &= \text{out}(s, c, i)\text{out}(\text{next}(s, c, i), x, y); \\
\text{help}(s, cx, iy) &= \text{help}(s, c, i)\text{help}(\text{next}(s, c, i), x, y); \\
\text{next}(s, cx, iy) &= \text{next}(\text{next}(s, c, i), x, y); \\
\text{finale}(x, y) &= \text{next}(s_0, c_0, x, y), \text{where } s_0 \text{ is } M\text{'s initial state and } c_0 \text{ is the teamwork state during initiating } M.
\end{align*}
\]

As we can see, \textit{next} and \textit{finale} can give appropriate results finally only when the numbers of tokens in the input and cooperation sequences are finite, whereas even if the \(x\) and \(y\) are infinite streams \textit{out} and \textit{help} also return values continually.
3.3 Language of CTM

The language of an FSA or PDA is the set of (finite) input strings that leave it in a final state. Some languages associated with a Turing Machine include the sets of inputs that cause it to halt; those that cause it to halt in a particular state or with a particular output; its set of possible finite outputs; etc. The behavior of an FSA, DFA, PDA, or TM involves finite input, and these automata are associated with sets of finite sequences produced when the machine halts.

In contrast, the behavior of a CTM or the result of its computations does not depend on the assumption that the automaton ever needs to halt. Therefore, we must associate the machine with some sets of infinite sequences, formally called streams.

**Definition 4 (Environment Observable Interaction (I/O) Streams):** Let $M=(S, C, I, O, \text{out}, \text{help}, \text{next})$, $s \in S$, $c \in C$, and $i \in I$; without loss of generality, let $o \in O$ be $\text{out}(s, c, i)$, $c' \in C$ be $\text{help}(s, c, i)$, and $s' \in S$ be $\text{next}(s, c, i)$. If at the state $s'$ the input is $i'$, then output would be $o' \in O$, where $o' = \text{out}(s', c', i')$. Therefore, the interaction (I/O) stream observable at state $s$ in time series like $((i, o), (i', o'), \ldots)$, which is a pair whose first element is $(i, o)$ and whose second element is any interaction stream observable at next state $s'$.

**Definition 5 (Team Sensible cooperation Streams):** The cooperation stream at $s$ that can be sensed by all partners is a teamwork history in time series like $((c_1, c_1'), (c_2, c_2'), \ldots)$, that is to say, it is a pair whose first element is the current teamwork state $c$ and corresponding help behavior writing $c'$ to the teamwork tape, whose second element is any cooperation stream sensible at next state $s'$.

Note that the current ingoing teamwork state maybe not equal to the last outgoing teamwork state because some partner might change the content on this teamwork tape between such two internal state transitions.

On one hand, from an external observer’s perspective, input/output actions sequences on input/output tape take place in the middle of CTM computation, which appears similar to an interactive computing device. Accordingly, the interaction of CTMs with their environment is naturally described by interaction (I/O) streams, which interleave input and output streams.

On the other hand, in the view of the fellows within the same teamwork group, cooperation tokens sequences on the shared teamwork tape evolve during the CTM computation, which means that CTMs are continuing cooperative computing devices. So the cooperation of CTMs with their colleagues is through the shared work-tape co-operated by whole team, cooperation streams developed finally.
Because both interaction streams and cooperation streams are related with the state transition, we can interleaver them into a single integrated stream, called *behavioral stream*, like \(((i_1, c_1, o_1, c'_1)), ((i_2, c_2, o_2, c'_2)), ((i_3, c_3, o_3, c'_3)), \ldots\).

By stretching the terminology, these sets of streams are a useful analog for CTMs to the languages of past automaton such as DFAs and PDAs.

In general, a single computational step of a CTM \(M\) is modeled by a computable function \(f_M: I \times S \times C \rightarrow O \times S \times C\), where \(I, O, S,\) and \(C\) are subsets of \(\Sigma^*\). Eventually, we can consider the language-based model.

The language of a CTM \(M\) is the set of behavioral streams comprising all interaction streams and cooperation streams for \(M\).

**Definition 6 (Language of CTM):** The language \(L\) of a CTM \(M\) is the set of behavioral streams that may occur for it. In other words, the set of all behavioral streams for a CTM \(M\) constitutes its language, \(L(M)\), as illustrated in Fig.3. In particular, let \(s_0\) be the initial own state of \(M\), then the language of \(M\), \(L(M)\), is the set of all behavioral streams at \(s_0\).

Due to the infinite and dynamic behavioral streams, this definition is co-inductive; the sets of all interaction and cooperation streams exhibited at any state are obtained by applying greatest-fixpoint semantics to the definition.

![Fig.3. Language of Cooperative Turing Machine (CTM)](image-url)
3.4 Semantics of CTM Computation

The semantics of CTM computations are stream-based, where the stream tokens are strings over some alphabet. The input stream is generated by the (observer) environment, while the output stream is generated by the CTM and then observed by the external world. Besides, there would be bidirectional cooperation streams read from and written to the shared teamwork tape by other partners. All of these streams have dynamic evaluation semantics, where the next value is not produced until the previous one is consumed.

As for an individual CTM, its every computational step corresponds to a TM computation and is also string-based. However, the whole CTM computation consists of a (finite or infinite) sequence of Turing-computable steps, consuming input stream tokens from the external environment, and then considering the cooperation context reflecting the teamwork situation; as a result, generating output stream tokens to the outside observers, meanwhile, developing the cooperation context.

Given a CTM $M$ defining a function $f_M$ and an input stream $(i_1, i_2, i_3, \ldots)$, a computation of $M$ consists of a sequence of computational steps and produces an output stream $(o_1, o_2, o_3, \ldots)$. The integrated state of the CTM consisting of its own state and the teamwork state evolves during the computation starting with the initial own state $s_0$ and the initial teamwork state $c_0$:

\[
\begin{align*}
    f_M(i_1, s_0, c_0) &= (o_1, s_1, c_0') \\
    f_M(i_2, s_1, c_1) &= (o_2, s_2, c_1') \\
    f_M(i_3, s_2, c_2) &= (o_3, s_3, c_2') \\
    \vdots
\end{align*}
\]

Without loss of generality, we can assume that $s_0$ is an empty string, but at that time $c_0$ is not empty in general because the CTM computation is often awakened to instantiate according to some teamwork requirement in the initial ingoing cooperation token $c_0$.

As we can see, In the view of an individual CTM, the cooperation context is also a developing stream like $(c_0, c_0', c_1, c_1', c_2, c_2', c_3, c_3', \ldots)$, where the next ingoing $c_{i+1}$ maybe not equal the previous outgoing $c_i'$ because the cooperation context would possibly change from $c_i'$ to different $c_{i+1}$ between step $i-1$ and step $i$.

Traditionally, each computation of a Turing Machine begins from the same configuration. This condition is waived when we add cooperation to such multi-tape TM, allowing the machine’s common teamwork tape to vary, by sharing the cooperation context and recording other teammates’ operation in its content.
4. RECOGNITION OF CTM

4.1 Inheritance and Development of Tradition

Essentially, we view CTMs as effective infinite-state transducers over infinite sets of input, output and cooperation tokens. In other words, the CTM are also an extension of the FST. Above all, CTMs extends one-symbol input and output of FSTs to strings. Furthermore, CTMs subdivides the input and output of FSTs into two aspects, viz. interaction with outside world observing, and cooperation within a team comprising partners. Last but not least, the CTM extends an FST’s lookup table to a Turing-computable function.

CTMs can be modeled as LTSs likewise, but not vice versa because the general LTS is not always appropriate for modeling effective computing devices. In fact, CTMs are restrictions of LTSs that obey following assumptions:

1. Computing devices must be finitely specifiable;
2. There needs to be a distinction between every two of inputs generated by the (observer) environment, every two of outputs produced by the computing devices, two of ingoing cooperation tokens read from the shared teamwork tape, and two of outgoing ones written to the shared teamwork tape;
3. At every stage, the next state, output and cooperation tokens must be effectively computable.

The semantics of CTM computations are based on dynamic streams. It is assumed that each input token $i_k$ is generated after the previous output $o_{k-1}$ meanwhile each ingoing cooperation token $c_k$ is generated after the previous outgoing $c_{k-1}$‘, which is also called the dynamic evaluation semantics of CTM’s behavioral streams. As a result, later behavioral tokens (output and outgoing cooperation tokens) may depend on earlier behavioral ones, on happening external (exogenous) events input from the environment, and on present teamwork context. Furthermore, an output may depend not only on current corresponding input but also on all earlier inputs, which behavior is known as history dependent. The output may even depend on the teamwork situation, resulting in cooperation context sensitive behavior.

The history dependency of a CTM cannot be expressed by the TM whose output is a function of only current input. Moreover, the cooperation context sensitive behavior even can not be expressed by almost all of existing TM extensions including interactive computing devices.

4.2 Properties Distinguishing CTM

As a result, some properties distinguish the CTM computation from traditional ones:
Persistence or semi-persistence of the cooperation context: the cooperation information can be preserved for a long or short term so as to allow successive partners reading and modifying.

Late binding: the next fellow who would co-operate the shared teamwork tape may be unavailable until the previous co-operation has been completed.

History dependence: current output can depend on earlier interaction sequences and preserved cooperation context.

Team sensitive: the state transition is associated with the shared teamwork tape which reflects the cooperation status in the teamwork.

Compound driven coordination: invoking the I/O interfaces corresponds to the process-oriented coordination in the control-driven way, whereas accessing shared teamwork tape is similar to the data-driven coordination in a tuple space.

Boundless input, output and cooperation tokens: Semantics of computation are based on infinite streams; there are non-enumerable numbers of streams, in contrast to sets of finite strings, which are always countable.

No determination: Because of the existence of late binding and team sensitive properties, there may be some random decision and unpredictable output behavior. Even given the same current internal state and input, the output generated may still be different.

Highly situated: The cooperative computation can not be determined and planned completely in advance. Its dynamic behavior is ad hoc in some sense.

Dynamic evolution: The cooperation itself progresses step by step according to the situation at that time, associated with current input token, cooperation context and some historic memory.

In summary, the existence of the shared teamwork tape and corresponding behavioral (especially cooperation) streams characterize CTMs as span-new cooperative computing devices, not traditional TMs.

5. EXPRESSIVENESS OF CTMS

5.1 CTMs are More Expressive in Comparison with TMs

The input of any computing device is generated by an environment. However, a typical computing environment might have some constrains on the input streams or sequences that it could generate. For example, there is almost a limit to its life span for a real system which sometimes needs to halt, so that it would generate finite input sequences in general.

Here, we give the definitions of observations and environments for computing devices.
Definition 7 (Observation): Given an arbitrary computing device $M$, any prefix of some interaction stream of $M$ is an observation of $M$; its length is the number of pairs in it.

Definition 8 (Environment): A mapping $E$ from an arbitrary computing device $M$ to the set of its observations $O$, i.e. $E: M \rightarrow O$, constitutes an environment of $M$.

Definition 9 (Algorithmic Environment): When an environment admits only observations of length one, it is known as an algorithmic environment.

Algorithmic environments are finite environments with shortest life span, where the instantiation of the computing device is for a single interaction only. This view of algorithmic environment reflects also the nature of classical Turing Machines.

Then without lost of generality, a device can only get feasible input sequences that satisfy the constraints of its environment. Based on the feasible inputs, any two devices in an environment would be observational equivalent or distinguishable.

Definition 10 (Observational distinguishable and equivalent): Within a given environment, two devices would either always produce the same output sequences when they receive the same feasible input sequences, or produce different outputs even if they receive the same inputs. In the former, they are observational distinguishable. In the latter, they are observational equivalent.

Definition 11 (Richer and Poorer): If two devices are observational equivalent in an environment but distinguishable in another, then the latter environment is said to be richer than the former, or the former is said to be poorer than the latter.

If two devices, $M_1$ and $M_2$, are observational distinguishable in an environment, then $E (M_1) \neq E (M_2)$, that is to say, there exists differences between them.

Definition 12 (Observational differences): Observational differences are observations which are feasible in an environment for one of the machines but not the other.

The observational differences can be used to distinguish two non equivalent computing devices.
**Theorem 1:** Any two non observational equivalent computing devices are observational distinguishable by some finite-length observational differences.

**Proof:** Let $M_1$ and $M_2$ be two non observational equivalent computing devices, then there must exist an observations feasible for $M_1$ but not for $M_2$. Hence there must be some $k$ such that the prefix of length $k$ of an interaction stream that can be observed from $M_1$ but not from $M_2$. If we choose such $k$, then the prefix of length $k$ construct an observational difference for $M_1$ and $M_2$.

In a similar way, if two devices are distinguishable by some observation, there must be some finite environment where they are distinguishable. For TMs, algorithmic environments can be used to distinguish them.

Given a TM $M$, defining a function $f_M$ from input to output strings, and an input string $(i_1, i_2, i_3, ..., i_n)$, a computation of $M$ would produce an output string $(o_1, o_2, o_3, ..., o_n)$, resulting in an observation $\{(i_1, o_1), (i_2, o_2), (i_3, o_3), ..., (i_n, o_n)\}$ where $o_k = f_M(i_k)$ for any $k$.

**Theorem 2:** The shortest observational difference between any pair of observational distinguishable TMs has length 1.

**Proof:** Let $M_1$ and $M_2$ be two non observational equivalent TMs, respectively defining two different map functions $f_{M_1}$ and $f_{M_2}$. Next Let’s assume $\{(i_1, o_1), (i_2, o_2), (i_3, o_3), ..., (i_n, o_n)\}$ to be a feasible observation from $M_1$, that is to say, for all $j$, $o_j = f_{M_1}(i_j)$. Then there must exist $m$ and $1 \leq m \leq k$, such that $o_m \neq f_{M_2}(i_m)$. Therefore, $(i_m, o_m)$ would be an observational difference of length 1 for them.

In contrast, CTMs might be distinguished by an observational difference of length more than 1.

**Theorem 3:** For any $k$, there must exist a pair of observational distinguishable CTMs whose shortest observational difference has length $k+1$.

**Proof:** We can construct $M_1$ and $M_2$ as a pair of observational distinguishable CTMs with the help of their teamwork partners, such that $M_1$ ignores its inputs, outputting $k$ 0’s followed by all 1’s, where $M_2$ ignores its inputs, outputting $k+1$ 0’s followed by all 1’s. Then the $k+1$th output of $M_1$ would be 1, but the $k+1$th output of $M_2$ would be 0, whereas previous $k$ outputs of $M_1$ or $M_2$ are all 0. So the shortest observational difference between $M_1$ and $M_2$ has length $k+1$.

As we can see, the expressiveness of CTMs are higher than TMs.
**Corollary 1:** CTMs are more expressive than TMs.

**Proof:** Follows from Theorem 1, Theorem 2, and Theorem 3.

### 5.2 Infinite Expressiveness Hierarchy of CTM

Next, we define an infinite sequence of finite CTM environments as \( \mathcal{E} \).

**Definition 13** *(Sequence of CTM Environments):* \( \mathcal{E} = (E_1, E_2, E_3, \ldots) \), where for any \( k \) and any CTM \( M, E_k(M) \) is the set of \( M \)'s observations of length \( \leq k \).

\( E_k \) gives us a relative notion of the observational equivalence with regard to environments that can generate at most \( k \) input tokens. \( E_1 \) is an algorithmic environment, with the coarsest notion of observational equivalence. Moreover, CTMs observational equivalent in one poorer environment may be observational distinguishable in another richer environment, vice versa.

**Lemma 1:** As to the CTMs, for any \( k, E_{k+1} \) is richer than \( E_k \).

**Proof:** Let’s construct a sequence of CTMs \((M_1, M_2, M_3, \ldots)\), such that for any \( k \), \( M_k \) ignores its inputs, outputting \( k \) 0’s followed by all 1’s, with the help of its teamwork partners. Then the \( k+1 \)th output of \( M_k \) is 1, but the \( k+1 \)th output of \( M_{k+1} \) is 0, whereas previous \( k \) outputs of \( M_k \) or \( M_{k+1} \) are all 0. As we can see, the shortest observational difference between \( M_k \) and \( M_{k+1} \) has length \( k+1 \). In other words, \( M_k \) and \( M_{k+1} \) are observational equivalent in \( E_k \), but distinguishable in \( E_{k+1} \), so \( E_{k+1} \) is richer than \( E_k \).

This Lemma shows that the environments in this sequence are increasingly richer, without reach a limit.

**Theorem 4:** The environment in the sequence \( \mathcal{E} \) induces an infinite expressiveness hierarchy, with TM at the bottom, and the absolute models of CTMs as the limit point.

**Proof:** For any class of devices, any environment \( E_k \) in this sequence induces a partitioning of them into observational equivalent classes, where the members of each class are observational equivalent. In general, more devices are observational distinguishable meanwhile less are observational equivalent in the richer environment than in the poorer one. Then, according to the Lemma 1, as the environments are increasingly richer, the members of observational equivalent class are gradually less. As a rule, the observational equivalent class in a richer environment is more expressive than that in a poorer one. Therefore, the environments in this sequence are also increasingly more expressive, and the hierarchy is infinite.

Due to the shortest observational difference of length 1, TMs are at the bottom of this hierarchy.
All the environments in the sequences are finite, but the absolute models of CTMs defined in this paper are infinite. The observational equivalent classes induced by the absolute models can be regarded as the limit point for the infinite expressiveness hierarchy.

6. EXPERIENCING CTM IN PRACTICE

The work described in this paper has been performed in the context of our Open Network Computing Environment (ONCE) project, especially in the Process Integration (PI) middleware [10], so as to achieve flexible service oriented cooperation in some business process management (BPM) application scenes.

In particular, ONCE PI can visually model and observe the cooperation across services. At design time, we can use activity diagrams to model all cooperative activities explicitly, as illustrated in Fig.4. At runtime, we can trace and observe its execution, as illustrated in Fig.5.

![Fig.4. Visually modeling cooperation based on CTM in the ONCE PI](image)

The input, output and cooperation tokens correspond to process creation, destruction and rendezvous events, as well as events invoking and calling services with parameters; whereas cooperation tokens can either be time and deferred events, or operation on shared data storage, etc. Besides, variables viz. data fields and formal parameters are also allowed in the service cooperation model in order to improve its flexibility.
Furthermore, every transition may be associated with a trigger event (from the environment or teammates), a Boolean guard and a sequence of actions. A transition is enabled and can fire if and only if its source state is in the current configuration, its trigger is offered by the external environment or teammates, and the guard is satisfied. Then this transition would be taken by the engine. Finally, the source state is left, the actions associated with the transition are executed, and the target state is entered, that is, a new state is achieved, and then proper actions or activities associated with the new state would be performed if they exist.

In this work, we base all models on the CTMs notions, in order to testify the capabilities of CTMs. The experiences proven that the CTMs can depict current hottest workflow and BPM applications well naturally.

7. CONCLUSION

CTMs are a more powerful model expected to describe emerging cooperative behaviors of present-day computing, such as component composition, service cooperation, blackboard, tuple space, multi-agents, CSCW, e-business and so on, because CTMs have a richer set of behaviors and more expressive than existing automata, TMs, interactive devices and others that are not sufficient to model cooperation. As a conclusion, although the
final silver bullet to resolve all problems maybe not exists, we believe that the CTM could help to model and simulate today’s most computing problems at least.

In nature, the Chomsky hierarchy is obtained through relaxing the restriction on accessing tapes gradually. Finite automata require the tape to be read-only, pushdown automata permit an unbounded auxiliary pushdown tape, linear-bounded automata permit writing but place length bounds on the tape, while Turing machines permit unrestricted but noninteractive tape access. Interaction machines explore the consequences of relaxing the restriction that automata are controlled by noninteractive tapes, they provide a persistent work tape to allow interaction with environment. Intuitively, the next paradigm shift is to open a common work tape shared by partners within the same team, so as to introduce cooperation that obviously brings out stronger expressiveness than before.

ACKNOWLEDGMENTS

Many thanks to all today’s teammates within the same group, and many historical colleagues contributed into the ONCE PI. This research was supported by the National Grand Fundamental Research 973 Program of China under Grant No. 2002CB312005, the National Natural Science Foundation of China under Grant No.60203029, 60173023, the National ‘863’ High-Tech R&D Plan of China (No. 2001AA113010, 2001AA414330, 2002AA413610, 2003AA413010, 2003AA115440).

REFERENCES
