philosophies

MDPI

Article

Symmetric Instruction Machines and Symmetric Turing Machines

Mark Burgin [1,†] and Marcin J. Schroeder [2,*]

[1] Department of Computer Science, University of California, Los Angeles (UCLA), Los Angeles, CA 90095, USA
[2] Faculty of International Liberal Arts, Akita International University, 193-2 Okutsubakidai, Aza Tsubakigawa, Yuwa, Akita 010-1211, Japan
* Correspondence: mjs@gl.aiu.ac.jp
† Deceased author.

Abstract: Symmetric instruction machines (SIAs) and symmetric Turing machines (STMs) are models of computation involving concepts derived from those of classical Turing machines such as tape (memory) and head (processor), but with different functional and structural characteristics. The former model (SIAs) introduced in this paper and preferred by Mark Burgin is a result of a reformulation of the latter model (STMs) published in several articles by the second author in the past. The properties of both models are analyzed and compared. The word "symmetric" in both cases represents the feature of the design which is distinct from classical Turing machines where only cells on the tape change under the action of the head. In both models, symmetric computing involves changes of the tape and parallel ("symmetric") changes of instructions listed in the head. The key difference between SIAs and STMs is in the dynamic of the changes, which in the former model has the form of compound one-way actions and in the latter model, it has the form of uniform mutual interactions, which only in specific realizations can be separated into a pair of actions. Because of the untimely passing of Mark Burgin, the discussion of the two models and cooperation on the paper has never been finished. For this reason, the arguments of both authors are reported even though, in some cases, they are mutually inconsistent or even contradictory.

Keywords: philosophy of computation; Turing machines; unconventional computation; symmetric Turing machines; interaction-based computation; computing power

Academic Editors: Jordi Vallverdú and Diane Proudfoot

Received: 18 August 2024
Revised: 28 December 2024
Accepted: 23 January 2025
Published: 27 January 2025

Citation: Burgin, M.; Schroeder, M.J. Symmetric Instruction Machines and Symmetric Turing Machines. *Philosophies* **2025**, *10*, 16. https://doi.org/10.3390/philosophies10010016

1. Introduction

Scientific terminology is not free from ambiguity. There are many examples of identical terms understood in very different ways in inquiries of different subjects. If the subjects are not related, a misunderstanding is unlikely. The more related subjects are, the higher the risk is. This relatively close relationship has a place in the case of the term "symmetric Turing machine", which makes it necessary to begin with disambiguation of this term. In this paper, a symmetric Turing machine, which is a generalization and modification of the usual Turing machine, is completely unrelated to the special case of Turing machines distinguished by the condition that their configuration graphs are undirected, that is, one configuration yields another configuration whenever yielding is reversed, as defined by Harry R. Lewis and Christos H. Papadimitriou [1].

To avoid further potential misunderstandings and to make this paper self-contained, a summary of the idea of symmetric Turing machines, its motivation, and its relationship to other ideas of unconventional computing is presented in Section 2, titled "Symmetric Turing

Machines", as a report on the earlier publications of the second author. In a nutshell (the details will follow in Section 2), in a symmetric Turing machine (with its name abbreviated to s-machine following the tradition initiated by Turing who called his original model of computation an a-machine, followed by c-machine for choice machine, u-machine for unorganized machines, etc.), each step of computation modifies (or possibly retains) both the present (active) cell on the tape and, at the same time, the present (active) instruction in the head of the machine. The description or identification of the s-machine is based on the configuration of the instructions in the head but on the dynamic of the mutual interaction between the cells and the instructions. In short, the one-way action of the classical Turing machine is replaced by mutual interactions in the s-machine. It is this characteristic of s-machines that makes their inquiry of special importance for the philosophy of computation, particularly in the context of natural computing systems.

This paper was written in unusual circumstances. The first author, Mark Burgin, passed away before the work on the paper was finished. Mark Burgin proposed, after discussions with the second author, a preliminary version of the paper, which was reproduced without any essential changes (except for editing of the section numbering, references, typos, etc.) in the present final version as an extensive Section 3, titled "Symmetric Instruction Machines" (the title originally proposed by Mark for the entire original paper by both authors). He asked the second author for revisions, corrections, or comments on the draft, overtly expecting additional input from the second author. Since the second author had objections to the preliminary version (mainly because the presented model did not fully involve a departure from the classical Turing machine), he communicated his suggestions to Mark for changes, specifically, some revisions of the claims about symmetric Turing machines. Mark responded to the suggestions of revisions with a short acknowledgment "It looks reasonable". No other response arrived and after some time, there was an announcement about his illness and passing.

It would be unethical to revise Mark's preliminary version of the paper with the second author guessing what exactly Mark had endorsed in such a short response. Instead, Section 4, titled "Interactive Processing of Information in Symmetric Turing Machines", presents what the second author considered necessary to add or revise. It contains a reformulation of the statements, theorems, and examples that are marked with asterisks in Section 3. In short, all of them become true if the term "symmetric Turing machine" is replaced by "symmetric instruction machine". The former term represents a very general model of computation, which, in the draft, was reduced to its very special case of a symmetric instruction machine.

The symmetric instruction machine is a model for the realization of the idea of the symmetric Turing machine (s-machine) within a framework closer to the traditional conceptualization of computation and as such is of great interest. In our discussions, while expressing his appreciation for the theoretical model of s-machines to which he referred to in his earlier publication [2], Mark was always concerned about the feasibility of their construction or physical realization. He applied the idea of symmetry of such machines in his model of the symmetric inductive Turing machine presented at the 2019 IS4SI Summit at the University of California, Berkeley [2]. In our discussion after his presentation, I challenged his statement that symmetric processing (such as in s-machines) cannot increase computing power beyond that of the Turing machine as he claimed in his talk and published in his proceedings paper: "Automata that perform transformations with their programs, such as reflexive Turing machines, were explored in [3]. It was proved that these machines have the same computing power as Turing machines. This result disproved Kleene's conjecture [4], which suggested that algorithms that change their programs while computing would be more powerful than Turing machines" [2]. My argument was that

the fact that one particular type of computing with changing programs (or with changing instructions in the case of Turing machines) does not have increased computing power does not disprove the conjecture that some models of this type do. Mark agreed but remained skeptical about the practicality of such a general symmetric model. He believed that symmetric Turing machines are of interest because of their efficiency rather than computing power. We planned our cooperation on further inquiry into symmetric Turing machines, which only became feasible two years later due to our busy schedules. This paper reports the results of this cooperation that was tragically and sadly disrupted by Mark Burgin's untimely passing.

2. Symmetric Turing Machines

2.1. Motivations for Symmetric Turing Machines

The ideas behind symmetric Turing machines published by the second author in a series of articles starting in 2013 had their sources in inquiries into unconventional, natural computing, and philosophical questions related to this subject [5–8]. These inquiries were originally motivated by the objective of finding a description of information dynamics that can be associated with computation in natural processes, for instance, in living organisms, their populations, and ultimately in general complex systems considered intelligent or conscious.

On the other hand, these inquiries could help in answering the frequently asked but never definitely answered question about what computation is. This question is non-trivial as the computer has become a paradigmatic symbol for any artifact or natural system involving information. As always for any non-trivial concepts, while everyone accepts the work of Turing's theoretical model of computation (classical Turing machine) as a paradigmatic instance of computation for which multiple equivalent models have been developed, the consensus ends with the question about the possibility of non-equivalent, more computationally powerful forms of computation.

Turing himself provided examples of such non-equivalent models (e.g., oracle machines); however, they had only theoretical meaning due to the status of oracles. The quest for more powerful computation models stimulated multiple attempts, resulting in some breakthrough developments, such as quantum computing. In most cases, such alternative models of computation increased the efficiency of computation, breaking the barriers to the practical feasibility of some computations but did not increase computational power (capacity to compute what at the lower power is non-computable). Thus, the challenge of the central issue of a generalization beyond the original model introduced by Turing has not been overcome, which hindered the attempts to define computation in a way that satisfies everyone.

Whatever way a definition of computing is formulated, it requires a *genus* (a more general concept that is already defined) that is close enough to the defined concept to simplify its *differentia* (the description of how all other concepts within the same *genus* differ from it). The attempts to use the intuitive but very general (highly removed in abstraction) concepts such as "information processing" as a *genus* led to either overgeneralization (such that every process involving information is considered a computation) or the circular definition of information processing through the hidden use of the concept of computation in Turing machines.

There was another source of problems in the context of natural computing. While it was relatively easy to translate the original expressions "read" or "print" in the works of Turing who illustrated his a-machine as a work of a human "computor" into the work of technological devices for the physical realizations of a-machines as electronic computers, such translation is much more difficult when we consider the realization of computing in

natural systems such as living cells. In such a case, we do not have actions but interactions of several components of a cell at the same or similar scale. The terms, such as "head" or "tape", used for Turing machines lose their intuitive meaning.

Thus, in the description of a symmetric Turing machine (s-machine), we will only use terms such as "tape" and "head" metaphorically to make it easier to understand the way s-machines generalize the a-machines of Turing. Someone who knows Turing machines can easily imagine the structure of an s-machine by analogy to an a-machine. However, it is important to remember that the objects that these terms represent may have different functional and structural characteristics, and functions such as "reading" and "printing" completely lose any meaning.

2.2. Conceptual Foundations for Symmetric Turing Machines

Before the description of s-machines is given, the assumptions underlying their conceptualization (explained in detail elsewhere) are as follows [5–8]:

- Since, in all natural instances of dynamics (not only in physics), instead of action, we always have interaction, and in the model of symmetric Turing machines, all dynamical processes are interactions. Actions are idealizations of specific instances of interaction in which the change of state of one side of the interaction is so small in comparison to the change of the state of the other side that it can be neglected in approximations;
- There are two mutually associated manifestations of information called selective and structural. The distinction is the differences in the way in which information manifests itself, not in the type of information. One manifestation is always associated with the other. In this particular context, an example of selective information is the choice of a character in a cell of the tape or of an instruction assigned to a particular instruction list position while structural information is the configuration of all characters on the tape or the configuration of all instructions in the list of instructions. Of course, in the former case, we can consider the structural information of the configuration of points from which a character is formed, and in the latter case, we can consider the selective information associated with the selection of one of many configurations of characters;
- The model of the symmetric Turing machine presented here uses the traditional terminology of Turing machines such as "head" and "tape" in their entirely new meaning. The purpose of retaining the terms is to support the understanding of the construction as a generalization of Turing machines. In particular, "instructions" in the head are just local states of the components of the head. They do not instruct any action.
- Since the dynamic of symmetric Turing machines is based on the interaction between the states of components of the two global information systems but not necessarily the values observable from the outside (e.g., reading of characters representing these states), symmetric Turing machines can be considered analog generalizations of digital Turing machines. In this case, the analog–digital distinction is borrowed from quantum theory distinctions between the description of the system (here, the information system) in terms of the states (analog) or observables (digital). In the case where we can uniquely identify the states with observables (such as in Turing machines where we assume that what the head reads in a cell on a tape is the actual state of the cell and the process of reading does not change this state), the analog description can be replaced by a digital one.

2.3. Description of Symmetric Turing Machines

Now, we can proceed to the description of the model of the symmetric Turing machine in the form of a list of main points of its theoretical construction, illustrated below in Figure 1. The description summarizes the content of the publications by the second author preceding his cooperation with Mark [5–8]. However, the discussions with Mark and his questions were very helpful later in writing this summary by pointing at the parts of the original publications that required additional clarification.

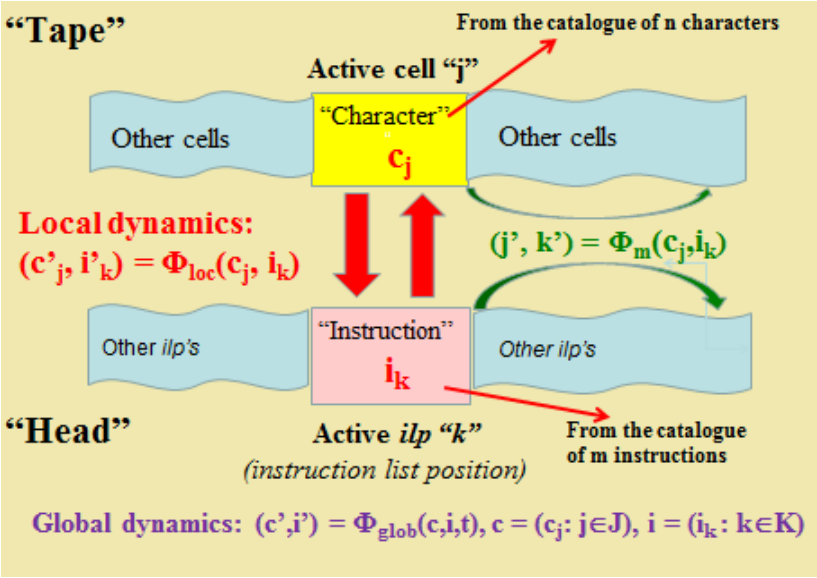

Figure 1. Diagram of a symmetric Turing machine [8].

1. Computation is understood as an interaction of two compound global information systems (called the "head" and "tape"), each consisting of an infinite but countable sequence of components, which themselves are information systems (the former is called "instruction list positions" (*ilp*) and the latter is called "*cells*"). We assume that each component can have an associated address indexed by an integer.
2. For each of the global information systems (head and tape), there is a separate, possibly but not necessarily different alphabet or catalogue of the possible states of their components (local information systems). The state of such component (*ilp* or *cell*) is a selection of an element from the appropriate alphabet/catalog. Every particular choice of a state for a cell or of an instruction for an instruction list position (*ilp*) is local selective information. Notice that the choice of a character for the cell or instruction for *ilp* does not identify the position of the cell or instruction in their configurations.
3. Each state of either of the global information systems (head or tape) is associated with the configuration of the states of their components (choices of elements from the corresponding alphabet for all their components). We can say that states of the global information systems carry structural information of the configuration, but this is equivalent to compound selective information as long as the components have assigned addresses.
4. It is important to note that the elements in the alphabet/catalogue for the head are functionally identical to the elements of the alphabet/catalogue for the tape. The state of an *ilp* (which, here, to maintain the analogy with Turing machines, is called an *instruction*), actually itself does not instruct anything or have any function in determining the process of computation but it is a variable entering the functions describing the dynamic of the computation. Possibly confusingly, the determination of the process of computing is not in the *instruction* but in the dynamic of the s-machine.

The *characters* (representing states of *cells*) and *instructions* (representing states of *ilp's*) do not have any fundamentally different features.

5. The process of computing involves more than one level of an information system. At the local level, the interaction is between a single pair of local component information systems called an *active cell* and *active ilp*. It is possible to consider further generalization, allowing for the involvement of a greater number of active components, but at this stage of inquiry, we will restrict local interactions to one pair of active components. This local dynamic expressing the interaction of the pair of active components transforms (or possibly retains) their states. This can be described as follows. At the local level, each step of the computation at the local state (c_j,i_k) of the active machine components (j,k) (where it is understood that the j-th *cell* is active and has *character* c_j and the k-th *ilp* is active and has *instruction* i_k) is described by the function $(c'_j,i'_k) = \Phi_{loc}(c_j,i_k)$, where c_j and c'_j belong to the catalogue of n *cell* states (*characters*), and i_k with i'_k belong to the catalogue of m states of *ilp* (*instructions*). The function Φ_{loc} does not depend explicitly on j and k, but exclusively on the values of c_j and i_k.
6. The second level of the dynamic, called the intermediate dynamic here, links the local and global levels of the information systems. It describes how the states of active components (at the local level) determine the selection of the addresses of the next pair of active components (including the possibility that either of the two addresses remains unchanged). Addresses link the lower, local level of the component information systems (more exactly, their states) with the upper, global level in which the global configuration is involved. In this case, we have another function $(j',k') = \Phi_m(c_j,i_k)$ (with index m indicating its "middle" status between the two levels), which, in this case, is (in principle) a compound function of j and k with the internal component function assigning the value (c_j,i_k) to (j,k), but its dependence on j and k may be eliminated (by defining each j' as j+s, where s is an integer function of the values of c_j and i_k, and similarly defining each k' as k+r, where r is an integer function of the values of c_j and i_k; then, Φ_m depends exclusively on the values of c_j, i_k, and the parameters s and r).
7. Finally, we have the third function Φ_{glob} describing the dynamic of the s-machine exclusively at the global level, which can be derived from the other two functions Φ_{loc} and Φ_m so that it does not have to be included in the identification of the s-machine. We have here the function $(\underline{c}',\underline{i}') = \Phi_{glob}(\underline{c},\underline{i},t)$ describing the change in the global states of the two interacting systems which form the machine after t iterations starting from the global states $\underline{c} = (c_j: j\in J)$ and $\underline{i} = (i_k: k\in K)$, where J and K are sets indexing the components of the global information systems and in this context, both can be identified as the sets of all integers.
8. The description of an s-machine includes the assumptions that both catalogues include a neutral or void element (neutral element for cells and neutral element for *ilp's*) and that in the initial state of the computation, all but a finite number of *cells* and *ilp's* are in this neutral state. We have also a normalization condition identifying the *initial active cell* and *initial active ilp*. For simplicity, we may assume that this initial state for (j,k) = (0,0).
9. It is obvious that the s-machine in which the state of each instruction list position (*ilp*) always remains the same, allowing for the identification of *ilp*'s with the respective instructions, is a usual a-machine. The disassociation of the change in the local state of the tape from the head and describing it as an interaction between the active local components of the compound systems is just a reinterpretation.

10. The concept of a universal symmetric Turing machine loses its classical meaning. Symmetric Turing machines are identified by their dynamic (more specifically, by the two functions Φ_{loc} and Φ_m). Thus, the universality would require a consideration of all possible choices of these functions. While in the case of the former function, we have a finite number of choices for new local states, the number of choices for the second function is infinite. Moreover, both functions (or at least one of them) can be non-computable (throughout this paper, non-computable means Turing non-computable), for instance, due to the use of non-computable sets in their description.
11. It can be easily observed that as long as the two functions Φ_{loc} and Φ_m are computable, there is a possibility to emulate the work of the s-machine using an a-machine. However, there is nothing that requires computability.

The diagram in Figure 1 provides a visual representation of the structural and functional description of symmetric Turing machines. The diagram in Figure 2 provides such a representation for the classical Turing machine model. The symmetry is reflected in the fact that we can exchange the names "head" and "tape" and the description will be still accurate after interchanging "cell"–"*ilp*" and "character"–"instruction" pairs. It is important that we have two global information systems, each of them compound, consisting of multiple local, component information systems. An interaction is described by the dynamic described by two functions removed from the "head".

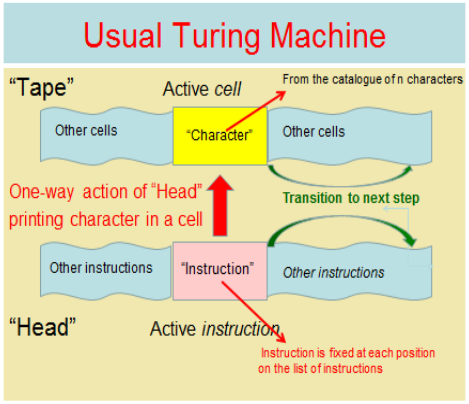

Figure 2. Diagram of a classical Turing machine.

One of the most important consequences of the generalization of Turing machines to symmetric Turing machines (actually, one of the main motivations for the way this generalization was designed by the second author) is the possibility of designing hierarchic multilevel computing systems which can be found in natural computation, particularly in living objects. Because the components of the global information systems do not have essentially different functional or structural characteristics, we can consider a multilevel, compound computing machine in which each pair of consecutive global information systems is an s-machine. A global information system can be a head for one s-machine and a tape for another at the same time, as described in Figure 3.

The apparent similarity of multilevel compound s-machines and neural networks designed for deep learning does not go far as the component s-machines may have very different dynamics (including non-computable dynamics) while all layers of deep learning neural networks have the same traditional dynamic.

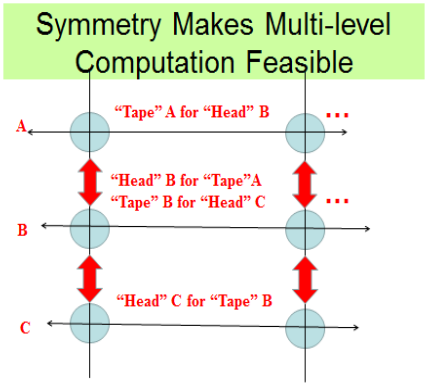

Figure 3. Diagram of a multilevel compound S-machine.

2.4. Difference Between Symmetric Turing Machines and Other Models of Computation Involving Changes in Instructions or Interactions

There is a natural question about whether and how s-machines are related (different or similar) to other computing system designs, particularly machines involving interactions. The answer is that to the best knowledge of the second author, the attempts to involve the changes in instructions or interactions were focused on configurations of several (at least more than one) Turing machines operating in a classical way or their equivalents or alternatively, on the external computing resources, but not on the internal structure or function of machines. Of course, Turing's oracle machines (o-machines) do not require additional Turing machines and actually, by definition, reject a-machines as oracles, but their design requires the engagement of an external component of an oracle.

Similarly, choice machines (c-machines) require choices made by an external agent. Choice machines were excluded from consideration by Turing as they assumed the finite number of choices made throughout every computation which can be realized by a single a-machine in which all possible choices (of a finite number) are considered. In non-deterministic Turing machines in which in all or some steps of computing the transition to the next state of either a cell or instruction (traditionally, the latter is called the state of the machine) is not determined by the present state can be reduced to the case of c-machines and therefore does not increase the computational power if the number of choices is finite. However, neither c-machines nor non-deterministic machines are comparable with general s-machines, as they do not admit changes in instructions.

There are several directions of inquiry in computational systems involving changes in instructions or interactions, but all such systems known to the second author are configurations of Turing machines. For instance, the direction of inquiry called Amorphous Computing overtly makes such an assumption: "An amorphous computing medium is a system of irregularly placed, asynchronous, locally interacting computing elements. We can model this medium as a collection of "computational particles" sprinkled irregularly on a surface or mixed throughout a volume. Each particle has modest computing power and a modest amount of memory. The particles are not synchronized, although we assume they compute at similar speeds, since they are all fabricated by the same process [...]. Thus, the entire amorphous medium can be regarded as a massively parallel computing system, ..." [9].

A more structured system consisting of a family of Turing machines indexed by the parameter t (related in some sense to time) is considered in evolutionary computing: "An Evolutionary Turing Machine (ETM) is a series of (possibly infinite) Turing Machines TM[t] working on population x[t] in generations t = 0, 1, 2, ..." [10]. In this case, the compound system of Turing machines is not just their sequence but the computation involves an

algorithmic transition between the components. However, the structure is built not within computing Turing machine or its generalization but between Turing machines.

The closest to the idea of s-machines is interactive computing, which was designed by Peter Wegner [11]. However, here too, the interaction is not within a component Turing machine or its generalization, but between a collection of orthodox Turing machines. This makes this form of interactive computing system of Turing machines incomparable to s-machines. However, in Wegner's analysis of the involvement of interaction, we can find a parallel to the functional analysis of s-machines. He writes, "Turing machines cannot model interaction machines (which extend Turing machines with interactive input/output) because interaction is not expressible by a finite initial input string. Interaction machines extend the Chomsky hierarchy, are modeled by interaction grammars, and precisely capture fuzzy concepts like open systems and empirical computer science. Computable functions cannot model real-world behavior because functions are too strong an abstraction, sacrificing the ability to model time and other real-world properties to realize formal tractability" [11]. This claim can be criticized as an overstatement as it is possible to find simple systems with interactions between Turing machines such that the interactions can be expressed in a sufficiently complex classical Turing machine. However, it is a well-justified claim that some interactions cannot be expressed.

Finally, we can consider some similarities between s-machines and computing systems for so-called physical reservoir computing. This is a subject of the recent extensive research in the context of neural networks and a variety of learning systems [12], but the interest in the idea of the involvement of a physical component in a computing system working as a black box making one or more steps of computation dates back further in time [13].

Actually, we can find the idea of using physical components (not yet called reservoirs) in the folklore of computing studies from the earliest time before the development of neural network theory, for instance, as a solution to the problem of the non-computability of randomness. If, in the process of computing, we need a truly random sequence of characters (which is not computable), we can delegate it to a physical system such as the process of tossing a die (which we now know is non-random but in the past, it was considered a paradigm of randomness). In cryptography, the use of physical external reservoirs allowed cryptographers to avoid the back door for decryption through the bias of pseudo-randomness. For us, in this paper, it is relevant that physical interactions are considered as a tool for computing; however, here too, such physical reservoirs are external resources. They are not essential for the work of the computing system and are only tools that can enhance computing.

3. Symmetric Instruction Machines (The Original Content of the Draft of This Paper Proposed by Mark Burgin with the Statements, Theorems, and Examples Marked with Asterisks Challenged by the Second Author in Section 4)

Algorithms and abstract automata (abstract machines) are used to describe and model computers, cell phones, computer networks, such as the Internet, and the processes in them. There are different forms of process descriptions: systems of instructions, functions, relations, or logical formulas. These forms determine the different types of programming languages, algorithms, and abstract automata. Here, we formalize classes of algorithms and abstract automata based on instructions, introduce new categories of these algorithms and abstract automata, such as instruction machines, finite automata, pushdown automata, register machines, Kolmogorov algorithms, random access machines (RAMs), and Turing machines.

All these automata satisfy the conditions of the Church–Turing Thesis, which asserts that Turing machines are the most powerful class of algorithms. Several researchers challenged this Thesis. For instance, in his talk at the International Congress of Mathematicians in 1958, Kleene formulated a conjecture that it might be possible that algorithms that change their programs while computing would be more powerful than Turing machines [4]. The first theoretical model of algorithms that changed their programs while computing was reflexive Turing machines [3]. However, it was proved that the class of reflexive Turing machines is equivalent to the class of Turing machines, i.e., these machines have the same computing power. In such a way, Kleene's conjecture was disproved but it was proved that reflexive Turing machines are more efficient than Turing machines. Namely, a relevant reflexive Turing machine can outperform any Turing machine that computes the same function [3].

Reflexive Turing machines are important special cases of symmetric Turing machines introduced by Schroeder as a tool for decreasing the complexity of information processing [5]. In this paper, we extend the concepts of machine symmetry and reflexivity in building symmetric instruction machines, which include, as special cases, symmetric Turing machines*, symmetric finite automata, inductive symmetric Turing machines, and inductive symmetric instruction machines, and we study their properties.

3.1. Instruction Machines

Many kinds of algorithms and abstract automata, such as finite automata, pushdown automata, register machines, Kolmogorov algorithms, random access machines (RAMs), and Turing machines, use instructions, for example, in the form of transition rules, to determine computational processes. All these classes of algorithms and abstract automata are unified by the comprehensive concept of an instruction machine.

Definition 3.1. *An instruction machine or instruction automaton M has three components:*

- A control device C_M which is a finite automaton and represents the states of a machine (automaton) M;
- *A memory* W_M which stores data;
- *A processor* P_M which transforms (processes) information (data) from memory W_M.

Memory W_M consists of cells and the connections between them. Each cell can be empty or contain a symbol from alphabet A_M of machine (automaton) M.

At each step of the computation, processor P_M observes one cell from memory W_M at a time and can change the symbol in this cell and go to another cell using connections in memory W_M. These operations are performed according to instructions R_M for processor P_M. Instructions R_M can be stored in processor P_M or in memory W_M.

Example 3.1. *A finite automaton G is an instruction machine in the following representation.*

Example 3.2. *A Turing machine T.*

Example 3.3 ([14,15])**.** *An inductive Turing machine K.*

There are three classes of instructions:

- *Straightforward or prescriptive instructions* directly state what the necessary action is.
- *Descriptive instructions describe* what result must be obtained.
- *Implicit instructions have a form of data that can be interpreted as instructions.*

Example 3.4. *The transition function of a finite automaton G is a straightforward instruction.*

Example 3.5. *A function in functional programming is a descriptive instruction.*

Example 3.6. *The weights of artificial neurons in artificial neural networks are implicit instructions.*

Symmetric instruction machines have a more advanced computational architecture than instruction machines.

Definition 3.2. *A symmetric instruction machine or symmetric instruction automaton H has five components:*

- *A control device C_H that is a finite automaton and represents the states of machine (automaton) H;*
- *A data memory W_H that stores data;*
- *An instruction memory V_H* that stores instructions;
- *A data processor P_M* that transforms (processes) information (data) from memory W_M;
- *An instruction processor D_M that transforms (processes) information (instructions) from memory V_M.*

Memory W_M consists of cells and the connections between them. Each cell can be empty or contain a symbol from alphabet A_M of machine (automaton) M.

At each step of the computation, processor P_M can observe one cell from memory W_M at a time, change the symbol in this cell, and go to another cell using the connections in memory W_M. The performed operation is defined by the instruction observed by processor D_M or in memory V_M.

At each step of the computation, processor D_M can observe one cell from memory V_M at a time, change the instruction in this cell, and go to another cell using the connections in memory V_M.

Example 3.7 * ([5]). *Symmetric Turing machine**

The transition function depends on two parameters—the data symbols and instructions. It is possible to interpret such transitions as caused by interactions between data and instructions.

There are two memory devices:

- A tape for data that keeps the processed data;
- A tape for instructions that keeps the instruction list.

Theorem 3.1. ** The functioning of a symmetric Turing machine can be simulated by a conventional Turing machine with two tapes and two heads.**

Corollary 3.1. ** Symmetric Turing machines are functionally equivalent to Turing machines.**

Corollary 3.2. ** Symmetric Turing machines are linguistically equivalent to Turing machines.**

Example 3.8. *Symmetric finite automaton.*

Example 3.9. *An inductive symmetric Turing machine K.*

Theorem 3.2. ** The functioning of a symmetric Turing machine can be simulated by a conventional Turing machine with three tapes and three heads.**

Theorem 3.3. ** There is a universal symmetric Turing machine.**

Theorem 3.4. *The functioning of an inductive symmetric Turing machine can be simulated by an inductive Turing machine with additional tapes and heads.*

Theorem 3.5. *There is a universal inductive symmetric Turing machine.*

3.2. Modes of Functioning and Results of Computation of Instruction Machines

There are various modes of information processing in abstract automata, material computers, and networks. For instance, there can be eight internal modes of abstract automata, computer, and network functioning [16]. Computational and networking practice shows that taking into account modes of information processing is important for the efficient design of distributed systems.

Here, we consider basic internal modes of automaton functioning.

- The reactive mode is when an automaton given some input directly gives the corresponding output;
- The recursive mode is built on recursion in processing input and giving the corresponding output;
- The inductive mode is built on induction in processing input and giving the corresponding output.

Examples of automata working in the inductive mode are deterministic finite automata, logical gates, and nondeterministic finite automata.

Examples of automata and algorithms working in the recursive mode are Turing machines, Kolmogorov algorithms, Minsky machines, Storage Modification Machines, Random Access Machines (RAMs), Petri nets, cellular automata, and partial recursive functions.

Examples of automata working in the inductive mode are inductive Turing machines [3], evolutionary inductive Turing machines [17], evolutionary finite automata working in the inductive mode [17], inductive cellular automata [15], and inductive register machines.

There are different techniques to organize the program formation in symmetric instruction machines. The basic types of them are as follows:

- *Prior program formation* (ppf) implies that the program, i.e., a set of instructions, is prepared before the main computation;
- *Continuous program formation* (cpf) implies that the program, i.e., a set of instructions, changes at each step of the main computation;
- *Interval program formation* (ipf) implies that the program, i.e., a set of instructions, does not change during definite intervals in the main computation.

There are three sorts of continuous program formation (cpf):

- *Formation parallel* to the main computation is when instructions are formed and performed at the same step of the computation;
- *Formation a priori concurrent* to the main computation is when instructions are formed and then performed after some time spent performing the computation, while the machine can wait until the necessary instruction is formed;
- *Formation precedes* the main computation: instructions are formed and then performed at the next step of the computation.

There are three sorts of interval program formation (cpf):

- *Formation parallel* to the main computation (data processing) is when instructions are formed and performed at the same time in a parallel way—two operations at each step—one with data and another with instructions;

- *Concurrent formation* means that at each interval, instructions are formed at the same time but independently of the main computation (data processing);
- *Separate formation* means that at each interval, either instructions are formed or data are processed.

Example 3.10 ([3])**.** *Depending on the organization of their functions, reflexive Turing machines perform either continuous or interval program formation.*

Example 3.11. *Computers perform prior program formation when they use compilers.*

Example 3.12. *Computers perform continuous program formation when they use interpreters.*

4. Interactive Processing of Information in Symmetric Turing Machines

This section is intended as a clarification of the reasons why the statements in the original draft of the paper written by Mark Burgin and sent to the second author, presented here as Section 3, were challenged by the second author and what revisions were suggested by the second author.

4.1. Reasons for Challenging the Statements in the Preceding Section Marked with Asterisks

The main reason for the need for revisions was the misstatement of the description of symmetric Turing machines from the earlier publications of the second author [5–8]. The most important feature of this model is a more general dynamic of information in the process of computation. In Turing machines, this dynamic of information is only in the form of one-way action of the head on tape, which is separate from the one-way action of the combined information of the content of a cell with information in the present instruction controlling the position of the head; in symmetric Turing machines, in the general case, it is a mutual interaction that characterizes all the physical processes, where the term "physical" is understood not just in epistemic way of the methodology of physics but in ontological way as a characteristic of reality or nature. Of course, two one-way actions can be reinterpreted as interactions in some circumstances, but the distinction is non-trivial. Thus, in Turing machines, we have an action as the most fundamental concept which allows for a combination of actions into a secondary concept of interaction. In natural phenomena, in many cases, interactions cannot be reduced to disentangled independent actions.

Here, we arrive at the key difference between symmetric Turing machines with their interactive dynamic and symmetric instruction machines. By definition, instruction machines are based on the dynamic of action. Any system in which change is based on the realization of instructions (i.e., prescription of action) has an action dynamic. Of course, this is also the case with symmetric instruction machines in which we have more than one action. Since, in some specific cases (e.g., in technological compound systems in which multiple interactions are designed to simulate one-way actions), we have instances of interactions that can be considered paired actions, symmetric instruction machines can be considered special cases of symmetric Turing machines, but not the other way around as is claimed in Section 3.

Theorem 3.1 stating that the functioning of a symmetric instruction machine can be simulated by a conventional Turing machine with two tapes and two heads, and is therefore equivalent to a Turing machine, is not a surprise. However, it is not true (i.e., not a theorem) for general symmetric Turing machines, unless we have a very special case of such a machine with functions Φ_{loc} and Φ_{m} that are computable, which is not required.

In our discussions with Mark, he expressed doubt about the existence of such functions. However, we have examples of such dynamical systems, and some of them are very simple, for instance, many-body mechanical systems.

Even the simple dynamics of the interaction of three identical colliding balls with trajectories crossing under equal angles is nondeterministic. Another example could be in chaotic dynamical systems that are strongly dependent on the initial conditions, which, although deterministic, may be considered non-computable. In this case, we may have systems in which the differences between the initial states of the system always produces much larger and growing differences in the final states. No matter how high the precision level used in the detection of the differences between the states is, the differences between the final ones is greater than the initial ones. This means that for any discretization of the states of the system, there will be more than one interval of the final states corresponding to one interval of the initial states. Thus, such physically deterministic systems will appear as non-deterministic after arbitrarily fine discretization. This justifies the claim that non-computability at a discrete level is not a practical effect that can be eliminated by increased precision but that it is inherent in discrete description.

More generally, any multicomponent system characterized by an emergent dynamic understood as irreducibility to the dynamics of components could serve as a potential implementation of non-computability. The fact that such systems are non-computable comes not from the way computability is understood, but from the very idea of emergence. The identification of multiple emergent phenomena in physics and biology recently led to the redirection towards non-reductionist scientific methodologies. Without referring to physics or biology, we can easily provide examples of non-computable functions defined by characteristic functions of non-computable sets. This does not mean that the choice of such non-computable sets can be easily justified by any practical reasons or be given any specific meaning. They may be considered as counterexamples to the claim that non-computable dynamics are theoretically impossible. Thus, the condition that the dynamic of a symmetric Turing machine may be described by non-computable functions Φ_{loc} and/or Φ_{m} is not speculative at all. This opens the possibility that such functions may have (in some cases) a computational power higher than that of Turing machines. The possibility of a non-computable dynamic for general symmetric machines is the main objective for the distinction between them and symmetric instruction machines. This section does not claim that symmetric Turing machines can implement more powerful computing than Turing machines but that Mark Burgin's claim in Section 3 that they cannot do it and that having efficient computing is the only advantage (true for symmetric instruction machines) is not justified.

4.2. Proposed Revisions

Now, we can proceed to the proposed revisions by the second author of the statements in Section 3 that are marked with asterisks.

The first one changes the last sentence before Sub-Section 3.1 to the following: "In this paper, we extend the concepts of machine symmetry and reflexivity in building machines based on the principles of symmetry, which include, as special cases, symmetric instruction machines, symmetric finite automata, inductive symmetric Turing machines, and inductive symmetric instruction machines, and we study their properties."

The next revision is in the example which refers to the symmetric Turing machine from an earlier publication of the second author [5]; its description is inconsistent with the original model since it changes it to a much more limited case in which the dynamic of computation is based on the traditional model of Turing machines. The example can be revised as follows.

Example 3.7. *Symmetric instruction machine.*

The transition function depends on two parameters—data symbols and instructions. It is possible to interpret such transitions as special cases of interactions of data with instructions, as is the case in a symmetric Turing machine [5].

There are two memory devices:

- A tape for data that keeps the processed data;
- A tape for instructions that keeps the instruction list.

The important restriction is that in this case, the term "instruction" has the same meaning as it does in usual Turing machines, while in symmetric Turing machines, this term is retained exclusively for illustrative purposes.

Due to the revision of the definition in Example 3.7, the series of theorems that follow have to be revised by replacing the expression "symmetric Turing machine" with "symmetric instruction machine".

Theorem 3.1. *The functioning of a symmetric instruction machine can be simulated by a conventional Turing machine with two tapes and two heads.*

Corollary 3.1. *Symmetric instruction machines are functionally equivalent to Turing machines.*

Corollary 3.2. *Symmetric instruction machines are linguistically equivalent to Turing machines.*

Theorem 3.2. *The functioning of a symmetric instruction machine can be simulated by a conventional Turing machine with three tapes and three heads.*

Theorem 3.3. *There is a universal symmetric instruction machine.*

Theorem 3.4. *The functioning of an inductive symmetric Turing machine can be simulated by an inductive Turing machine with additional tapes and heads.*

Theorem 3.5. *There is a universal inductive symmetric Turing machine.*

With these revisions, the content of Section 3 becomes true and fully consistent with the earlier publications of Mark Burgin and the second author. The revisions are placed in a separate section as the second author cannot be sure that Mark Burgin's short response to them ("It looks reasonable") can be interpreted as an endorsement of all of them, and if not all, it is not clear which ones.

Author Contributions: Writing—original draft, M.B.; Writing—review & editing, M.J.S. This paper was written in unusual circumstances. The first author, M.B., passed away before the work on the paper was finished. M.B. proposed a preliminary version of the paper based on the earlier discussions, which was reproduced in the final version as the section "Symmetric Instruction Machines" without any changes (except for editing references and eliminating typos). He asked the second author for revisions, corrections, or comments, which indicates his openness to the further evolution of the paper. Since the second author had objections to the preliminary version, he communicated his suggestions for changes, particularly some additions. M.B. responded with the short acknowledgment "It looks reasonable". No other response arrived and after some time, there was an announcement about his illness and passing. Thus, all other parts of the paper were written by the second author. They are consistent with the outcomes of the discussions between the two authors but the details of these sections were never endorsed by M.B. and the second author is solely responsible for their content. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: The study is purely theoretical-philosophical and does not require Institutional Review to protect any rights.

Informed Consent Statement: Not applicable as no human subjects involved.

Data Availability Statement: Not applicable as no data involved.

Acknowledgments: Even in writing the sections of this paper complementing Section 3 (written by Mark Burgin) after his passing, the second author benefited from the opportunity to discuss many relevant subjects on the philosophy of computation with him. The second author benefited from the comments received from the anonymous reviewers, which helped in improving the text of the paper.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Lewis, H.R.; Papadimitriou, C.H. Symmetric space-bounded computation. *Theor. Comput. Sci.* **1982**, *19*, 161–187. [CrossRef]
2. Burgin, M. Information Processing by Symmetric Inductive Turing Machines. *Proceedings* **2020**, *47*, 28. [CrossRef]
3. Burgin, M. Reflexive Calculi and Logic of Expert Systems. In *Creative Processes Modeling by the Means of Knowledge Bases*; Institute of Mathematics: Sofia, Bulgaria, 1992; pp. 139–160. (In Russian)
4. Kleene, S.C. Mathematical Logic: Constructive and Non-Constructive Operations. In Proceedings of the International Congress of Mathematicians, Edinburg, UK, 14–21 August 1958; Cambridge University Press: Cambridge, UK, 1960; pp. 137–153.
5. Schroeder, M.J. From Proactive to Interactive Theory of Computation. In Proceedings of the 6th AISB Symposium on Computing and Philosophy: The Scandal of Computation—What is Computation? Exeter, UK, 2–5 April 2013; pp. 47–51.
6. Schroeder, M.J. Dualism of Selective and Structural Manifestations of Information in Modelling of Information Dynamics. In *Computing Nature*; SAPERE 7; Springer: Berlin/Heidelberg, Germany, 2013; pp. 125–137.
7. Schroeder, M.J. Towards Autonomous Computation: Geometric Methods of Computing. *APA Newsl. Philos. Comput.* **2015**, *15*, 9–27. Available online: https://cdn.ymaws.com/www.apaonline.org/resource/collection/EADE8D52-8D02-4136-9A2A-729368501E43/ComputersV15n1.pdf (accessed on 20 July 2024).
8. Schroeder, M.J. Hierarchic Information Systems in a Search for Methods to Transcend Limitations of Complexity. *Philosophies* **2016**, *1*, 1–14. [CrossRef]
9. Abelson, H.; Allen, D.; Coore, D.; Hanson, C.; Homsy, G.; Knight Jr, T.F.; Nagpal, R.; Rauch, E.; Sussman, G.J.; Weiss, R. Amorphous Computing. *Commun. ACM* **2000**, *43*, 74–82. Available online: https://dl.acm.org/doi/10.1145/332833.332842 (accessed on 20 July 2024). [CrossRef]
10. Eberbach, E. Toward a theory of evolutionary computation. *BioSystems* **2005**, *82*, 1–19. [CrossRef] [PubMed]
11. Wegner, P. Interactive foundations of computing. *Theor. Comput. Sci.* **1998**, *192*, 315–351. [CrossRef]
12. Tanaka, G.; Yamane, T.; Héroux, J.B.; Nakane, R.; Kanazawa, N.; Takeda, S.; Numata, H.; Nakano, D.; Hirose, A. Recent advances in physical reservoir computing: A review. *Neural Netw.* **2019**, *115*, 100–123. [CrossRef]
13. Siegelmann, H.T. Computation Beyond the Turing Limit. *Science* **1995**, *268*, 545–548. [CrossRef]
14. Burgin, M. Nonlinear Phenomena in Spaces of Algorithms. *Int. J. Comput. Math.* **2003**, *80*, 1449–1476. [CrossRef]
15. Burgin, M. *Superrecursive Algorithms*; Springer: New York, NY, USA, 2005.
16. Burgin, M. Super-recursive Algorithms and Modes of Computation. In Proceedings of the 2015 European Conference on Software Architecture Workshops, Dubrovnik, Croatia, 7–11 September 2015; ACM: New York, NY, USA, 2015; pp. 1–5.
17. Burgin, M.; Eberbach, E. Universality for Turing Machines, Inductive Turing Machines and Evolutionary Algorithms. *Fundam. Informaticae* **2009**, *91*, 53–77. [CrossRef]