Prolegomena to an operator theory of computation

Citation for the original published paper (version of record):

Burgin, M., Dodig Crnkovic, G. (2020)
Prolegomena to an operator theory of computation
Information (Switzerland), 11(11)
http://dx.doi.org/10.3390/info11070349

information

MDPI

Article

Prolegomena to an Operator Theory of Computation

Mark Burgin [1,*] **and Gordana Dodig-Crnkovic** [2]

[1] Department of Mathematics, University of California, Los Angeles, 520 Portola Plaza, Los Angeles, CA 90095, USA

[2] Department of Computer Science and Engineering, Chalmers University of Technology, 41296 Gothenburg, Sweden; gordana.dodig-crnkovic@chalmers.se

* Correspondence: mburgin@math.ucla.edu

Received: 20 May 2020; Accepted: 29 June 2020; Published: 4 July 2020

Abstract: Defining computation as information processing (information dynamics) with information as a relational property of data structures (the difference in one system that makes a difference in another system) makes it very suitable to use operator formulation, with similarities to category theory. The concept of the operator is exceedingly important in many knowledge areas as a tool of theoretical studies and practical applications. Here we introduce the operator theory of computing, opening new opportunities for the exploration of computing devices, processes, and their networks.

Keywords: computation; information; info-computation; computing device; operator; network; computer architecture; composition

1. Introduction

The operator approach in computer science is, as we explain below, in essence similar to the category methodology in mathematics, providing innovative tools for the exploration of computing systems and computation.

The concept of a category was introduced by Eilenberg and McLane [1]. It was stimulated by the discovery and study of relations between abstract algebra and topology. The main idea of the category methodology is that the notion of structure-preserving mapping, such as a homomorphism in algebra or a continuous mapping in topology, is in many situations more important than the inner construction of mathematical objects when they are built from separate elements.

Traditionally, systems, in general, and mathematical systems, in particular, have been described by their inner structures, i.e., elements and relations between them [2,3]. For instance, a group is defined as a set G of elements with a binary operation over its elements, which satisfies axioms of closure, associativity, and has an identity element and inverse element.

In contrast to group theory or topology, category theory suggested describing mathematical systems by their *structure-preserving mappings* (see Goldblatt [4]), i.e., by their *outer structures* in the sense of the general theory of structures developed by Burgin [5].

In a similar way, computing systems have traditionally been described by their inner structures in the form of elements, components, and basic operations. For instance, a Turing machine contains three basic components: the control device, read/write head, and memory (tape). The control device governs the functioning of the head while the head performs transformations in the memory of the machine.

In contrast to this, operator theory suggested describing computing systems by their functioning, i.e., by their *pure external structures* in the sense of the general theory of structures [5]. Namely, a computing system generates a process of transformation of the input objects into the output objects. This can be described within the framework of info-computation [6], which models information as a structure and computation as information transformation, i.e., the dynamics of information.

Each such transformation can be a process that consists of many steps of elementary transformations of information structures. The data in this context are atoms of information, or the most elementary units of information, which are building increasingly complex structures through the transformations, that is, the process of computation.

Category theory of mathematical systems changes their descriptions by inner structures to their descriptions by outer structures. In a similar way, operator theory of computing systems changes their descriptions by inner structures to their descriptions by external structures.

In terms of operator theory, the system of all processes generated by a computing system Q is described as an operator A_Q, while each such process (transformation) is an application of operator A_Q to the input objects. For instance, a Turing machine operator describes transformation of the input words into the output results by a Turing machine.

Operators can be used to represent important *ontological features* of reality reflecting theirdynamical nature through a process when some objects act on other objects. Our universe exists in interactions of its components and elements, while every interaction can be decomposed in separate actions, each involving an operator.

At the same time, *epistemic* operators express the basic mechanisms of cognition, that is, the process of knowing about the world and its ontology. Cognitive processes, such as search, selection or recognition and learning, are performed by *physical* processes which can be represented by abstract operators decomposable into simpler operators. In the context of epistemic processes, any cognizing (knowledge-generating) physical system, such as a scientist, a measuring device, or a computer, is a *physical operator*.

Given their broad applicability, operators are one of the most important tools in sciences. In theoretical physics, both classical physics and quantum physics have operator formulations (cf., for example [7,8]). In mathematics, there is operator theory, which studies operators in Hilbert or Banach spaces (cf., for example [9–11]). Operators are used in chemistry as well as in computer science. In addition, operators are also becoming an important tool in information theory (cf., for example [5,12–14]).

Computing devices are information processors, transformers, transmitters, and generators of information. That is why in this work, we develop operator models of computing devices and study their properties based on the ontological operator theory originated by Burgin and Brenner [15].

The operator approach has many advantages. For instance, why do physicists call functions from one vector space to another by the new name *operator*? The reason is that in physics operators act mostly in infinite dimensional vector spaces, e.g., Hilbert spaces. Such spaces have many advanced properties. These properties are reflected in properties of operators, which become much more complex and sophisticated than conventional numerical functions. Even more, new properties which numerical functions do not have, such as the spectrum, were discovered in operators and proved very useful.

The same situation has become apparent in computer science. At first, operations of abstract and physical automata were represented by functions, such as the transition function of an automaton. However, processed data started becoming more and more complex and sophisticated bringing the necessity of introducing operators to describe functioning of abstract and physical automata.

In quantum computation, transition to operators happened automatically because quantum processes are traditionally described in physics by operators. For instance, a quantum circuit is a system of quantum gates and each quantum gate is a unitary operator, which acts on states of quantum systems.

However, even before emergence of quantum computation, operations in many programming languages are called operators. For instance, in computer programs, one of the most familiar sets of operators, the Boolean operators, is used to work with true/false values. Boolean operators include AND, OR, NOT (or AND NOT), and NEAR. These (and variations, such as XOR) are used in logic gates. Another class used in computer programming is formed by arithmetical operators, which contain +,

−, × and ÷. Other types of operators used in computer programming include *assignment operators*, which assign a specified value to another value and *relational operators*, which compare two values.

The reason for treating AND, OR, ADD, and other similar programming objects as operators and not only as operations is that an operator is what performs data transformations and an operation is the transformation itself. This distinction is explicitly exposed in the general model of an operator introduced in the ontological operator theory [15].

Another example of operator utilization comes from parallel programming presented by Pingali et al. [16]. A further practical area that brings forth utilization of operators for modeling, exploration, and application is in biological and chemical computers and other natural computers addressed by Rozenberg et al. [17] and Adamatzky [18]

One more theoretical area that brings forth utilization of operators for modeling, exploration and application is the theory of structural machines of Burgin and Adamatzky [19,20].

Morphological computing is a model of physical computation that especially involves transformations of a physical system, typically on a hierarchy of levels of organization, as argued by Dodig-Crnkovic [6]. It involves transformations of information structures (including elementary information structures—data structures), which are efficiently modeled by operators representing information dynamics.

2. The Concept of Operator

In physics, an operator is a function over a space of initial physical states to the space of final states. In classical mechanics, the movement of a particle (or a system of particles) is completely determined by the Lagrangian or, equivalently, the Hamiltonian operator of a system.

Operators in classical mechanics are related to symmetries which reflect invariance of motion with respect to a coordinate (Noether′s theorem). Thus, translational, rotational, Galilean transformation, parity and T-symmetry, each is connected with a specific classical mechanic operator. Operators in quantum mechanics are integral part of the formulation of QM. Thus, position, momentum, kinetic energy, angular momentum, spin, and Hamiltonian are expressed as operators in QM.

An example from quantum physics is S-matrix (scattering matrix), which denotes an operator that describes the process of transfer of a quantum-mechanical system from the initial state to the final one as a result of a scattering. Taking the set of quantum numbers describing the initial and final states, the scattering amplitudes form a table, which is called the scattering matrix S.

In quantum chemistry according to Levine [21], an operator is defined as "a rule that transforms a given function into another function". The differentiation operator d/dx is an example of operator that transforms a differentiable function $f(x)$ into another function $f'(x)$. Other examples include integration, the square root, and so forth. Numbers can also be considered as operators (they multiply a function). McQuarrie [22] gives an even more general definition for an operator: "An operator is a symbol that tells you to do something with whatever follows the symbol".

Operators are widely used in computer programming as well. For example, the Boolean operators, AND, OR, NOT (or AND NOT), and NEAR, with variations such as XOR, are used in logic gates. Furthermore, assignment operators, which assign a specified value to another value and relational operators, which compare two values are widely used in computer programming.

According to the Techopedia [23], an operator in computer programming is a symbol that usually represents an action or a process. An operator is used for manipulating a certain value or operator. For example, in "1 + 2", the "1" and "2" are the operands and the plus symbol is the operator. Common operators in programming languages are =, ==, +, ++, −, /, *, >, <, etc.

The unified operator theory of Burgin and Brenner [15] provides the most encompassing definitions of operators and related concepts. Here we use definitions from this theory.

Definition 1. *An operator is an object (system) that operates, i.e., performs operations on, some objects, systems or processes, which are called operands of this operator.*

This brings us to the following definition:

Definition 2. *An operand is an object, system or process operated by an operator.*

These definitions show that being an operator or an operand is a role and a characteristic of a system/object. One and the same system/object can be an operator in some situations and an operand in other situations. In a similar way, a system/object can be an operator with respect to some systems and not an operator with respect to other systems.

Definitions 1 and 2 form the foundation of the unified operator theory [1], which can be specified for a diversity of specialized operator theories, such as operator theories in physics and chemistry, or the theory of programming operators.

Definitions 1 and 2 also express the fundamental dyadic relation between operators and their operands, which is actualized in the form of the operator triad:

$$\text{Operator} \overset{\text{Operation/function}}{\longrightarrow} \text{Operand}$$

This diagram presents an operation as a component of an operator triad.

The operator triad is a special case of the basic fundamental triad [5,24]. In the symbolic representation, it has the form:

$$(\text{Op}, \text{on}, \text{Od})$$

where Op is an operator, on is an operation, and Od is an operand.

To construct a general mathematical operator theory in some domain, for example, in the realm of computations, it is necessary to organize the multiplicity of relevant operands in the form of an operating space, i.e., the space that is transformed by an operator.

In this context, the key formal model of an operator Op also has the form of the basic fundamental triad:

$$\text{Op} = (\text{D}, \text{on}, \text{C})$$

where D = D(Op) is the domain of the operator Op, i.e., a space that contains all objects that are operands of this operator on is the operation that the operator Op performs.

C = D(Op) is the codomain of the operator Op, i.e., a space that contains all objects that are results of this operator.

Together the domain D and codomain C form the operating space of the operator Op.

An arbitrary operator A is not necessarily defined for all elements from its domain D(A). The subspace (subset) of D(A) where A is defined is called the definability domain and denoted by DD(A). For instance, taking a Turing machine $T_\emptyset$ that works with words in the alphabet {0, 1} but never halts independently of its input, we see the domain $D(T_\emptyset)$ is the set of all words in the alphabet {0, 1} while the definability domain $DD(T_\emptyset)$ is the empty set $\emptyset$.

In a similar way, the range R(A) of an operator A, i.e., the set of all elements that are values of A, can be only a part of its codomain C(A). For instance, the codomain $C(T_\emptyset)$ is the set of all words in the alphabet {0, 1} while the range $R(T_\emptyset)$ is the empty set $\emptyset$.

3. The Concept of the Information Operator

Different types of operators function in distinct operating spaces. For instance, operators of quantum mechanics operate on Hilbert spaces.

Information operators work in information spaces. As there are diverse types of information, operator representation demands different types of information spaces. According to the multiscale taxonomy of information, Burgin and Dodig-Crnkovic [25] differentiate among syntactic, semantic, and pragmatic information; algorithmic and descriptive information; and cognitive, effective, and emotional information. In this context, each type of information has the corresponding type of information spaces. Here we should add that, according to recent results from cognitive science and neuroscience, cognitive and emotional phenomena cannot be treated separately [26]. With modern understanding of embodied, embedded, and enacted (EEE) cognition, emotions are an integral part of a real-life process of cognition [26], which is based on information processing.

There are three basic types of information operators:

- *Substantial information operators* transform physical objects into structural objects, i.e., their domain consists of physical objects while structural objects shape their range.
- *Co-substantial information operators* transform structural objects into physical objects, i.e., their domain consists of structural objects while physical objects compose their range.
- *Pure information operators* transform structural objects into structural objects, i.e., their domain and range consist of structural objects.

In all cases, structural objects are interrelated forming information spaces with various operations acting on these objects. In what follows we mostly concentrate on pure information operators, the domain and range of which are some information spaces.

While modeling computing devices by information operators, we treat operators as tools for transformation and generation of distinct kinds of information, and call them *computational information operators*. In the case of computing over information spaces, the most important classes of involved information are syntactic, semantic, and pragmatic information. This brings us to syntactic, semantic, and pragmatic information spaces, which are the most used operating spaces for computational information operators.

3.1. Syntactic Information Spaces

Digital computing devices process information in symbolic form, transforming words of some language. Consequently, it is usual to represent both physical and abstract, *syntactic information spaces* as systems of formal, artificial or natural languages. This implies that treating these devices as operators, we encounter operands of five types: separate symbols, words, texts, languages, and families of languages.

In the case of natural computing that refers to physical computations (morphological, chemical, cognitive, quantum, and other kinds of physical computation) we have *physical objects* that perform computations. Molecules, for example, on which our brains perform computations, are not only symbols as words of an electro-chemical language, but they are concrete physical objects as argued by Alcami and El Hady [27] and Silver [28]. That is why emotions are part of cognitive process, since cognition is not only electro-chemical *symbol* manipulation, but also embodied electro-chemical *material object* manipulation [26]. There is a difference between the symbol/word and the object which it represents. In our case, it is the difference between the classical Turing model of logical computing machine type of computation and natural computation in the real physical world, elaborated by Rozenberg et al. in [17].

Taking an abstract computing device such as a Turing machine, we come to the *syntactic operating space*, which consists of all formal *languages* with the alphabet of a particular Turing machine. At the same time, it is possible to take the space of all *strings* in some alphabet as the syntactic operating space of a Turing machine. The property of the Turing machine is that *it identifies symbolic representation of a machine with a machine itself* as an object in the symbolic world or symbolosphere [29]. That is the basis of programmable computing.

Note that while the Turing machine is not a concrete/physical object, but a symbolic model of computation, in our approach, the grounding of the concept of computation is achieved by allowing the most fundamental model of computation being the physical process itself. It is similar to Rodney Brooks' idea of AI without representation [30]. Philosophically, this points back to the symbol grounding problem, which is resolved in AI through grounding symbols in embodied cognition of agents, based on signals obtained directly from the physical world through sensors/senses.

In the case of finite automata, it is possible to utilize syntactic information spaces of four types. A syntactic information space of the first type consists of all *symbols* from the alphabet of the finite automaton. A syntactic information space of the second type consists of all symbols denoting *states* of the finite automaton. A syntactic information space of the third type consists of all *words* from the formal languages with the alphabet of the finite automaton. A syntactic information space of the fourth type consists of all *formal languages* with the alphabet of the finite automaton.

The majority of abstract automata (computing devices) work with linear (i.e., one-dimensional) languages. However, there are also abstract automata (computing devices) that work with more complex structures. For instance, Kolmogorov algorithms work with arbitrary graphs [31], Turing machines with two-dimensional tapes and two-dimensional cellular automata work with two-dimensional structures while structural machines work with arbitrary structures, according to Burgin and Adamatzky [19,20].

As the result, operators representing different abstract automata (computing devices) have different syntactic operating spaces.

Operators that represent Kolmogorov algorithms are Kolmogorov computation operators [31], the syntactic operating space of which is the collection of formal graph languages, i.e., languages the words of which are graphs, while the definability domain consists of all enumerable (recursively computable) graph languages.

Operators that represent two-dimensional cellular automata, as shown by Codd [32] are two-dimensional cellular computation operators, the syntactic operating space of which is the collection of two-dimensional array languages, i.e., languages the words of which are two-dimensional arrays, while the definability domain consists of all enumerable (recursively computable) two-dimensional array languages.

Operators that represent structural machines are structural computation operators, the syntactic operating space of which is the collection of structural languages, i.e., languages the words of which are structures, while the definability domain consists of all enumerable (recursively computable) structural languages.

Operators that represent Turing machines with one-dimensional tapes are one-dimensional Turing computation operators, the syntactic operating space of which is the collection of formal languages while the definability domain consists of all recursively enumerable (recursively computable) languages.

Operators that represent Turing machines with two-dimensional tapes are two-dimensional Turing computation operators, the syntactic operating space of which is the collection of two-dimensional formal languages while the definability domain consists of all recursively enumerable (recursively computable) two-dimensional languages.

3.2. Semantic Information Spaces

Semantic information spaces can also be distinguished based on different kinds of the semantic theory of information. For instance, in the semantic information theory of Bar-Hillel and Carnap, a semantic information space consists of possible worlds according to Bar-Hillel and Carnap [33]. Often a semantic information space is a conceptual space, as studied by Gärdenfors [34–36]. In the theory of epistemic information, a semantic information space is a conceptual space [5,13]. In the semantic information theory of Shreider [37], a semantic information space is a thesaurus as a system of texts and semantic relations between these texts. Conceptual spaces studied by Burgin and Díaz-Nafríagive one more example of semantic information spaces [38]. Utilization of semantic information spaces in modeling of computing devices by information operators allows studying semantic aspects of computation, computing systems and networks.

Natural computation, as it is presented by Dodig-Crnkovic in [39,40], involves different information spaces. Indeed, if we talk of physical computation, we must start with ontology-epistemology relationship with focus on the material properties of objects, instead of their logical properties. We do not logically derive physics, chemistry and biology as computational phenomena, we observe what there is in nature—that is, not a closed logical system, as Burgin and Dodig-Crnkovic [41] argued in the article describing the shift from the closed classical algorithmic universe to the open world of algorithmic constellations, where algorithms are physical/chemical/biological mechanisms as well. Especially taking into account the whole loop from ontology to epistemology and back as implemented in robotics, we notice that through the phenomenon of morphological computing, computational control of the central controlling mechanism in the robot can be replaced by natural physical behavior of a material of a robots body. For instance, passive dynamic walker robot walks down the slope without any other control, but the physical properties of its body, anticipated through the knowledge of its direct physical interactions with the environment as presented by Pfeifer and Bongard [42].

In case of natural computing/physical computing such as found in living organisms and robots, information semantics defines the relationship between the physical world and its symbolic representation or a behavior of a computational system. When talking about programming languages semantics (and Turing machines is a programming language equivalent), semantics is evaluation of the meaning of syntactically valid strings of symbols defined by a programming language, to the description of the computation involved, that is computational behavior—so it is language mapping.

In the similar way as syntactic and semantic information spaces, it is also possible to introduce pragmatic information spaces and pragmatic information operators. For instance, it is possible to consider the space of goals with corresponding operations as a pragmatic information space. The space of intentions is another kind of pragmatic information spaces.

4. The Concept of Computational Information Operator

Computation can be understood as information processing. Nevertheless, information transmission or communication is typically not seen as computation, although it may be taken as an element or part of computation. Computation is typically considered as having some input and producing some output. However, Alcami and El Hady [27] describe axonal information processing (where transmission of information proceedsfrom the cell body to the nerve terminal through an axon) as computation. Shannon defined his concept of information based on a technical model of human communication. Both computation and communication imply the transformation of information (where transformation can be identity). Bohan Broderick [43] compares notions of communication and computation and concludes that computation and communication are not conceptually distinguishable. They may be distinguished with respect to a given system, so that computation is limited to a process *within* a system (such as in the Turing Machine) and communication is an interaction *between* systems or *between* a system and its environment (such as in interactive computing and natural computing).

Dynamics of information is defined as a general form of computation. If the physical universe is an information structure, natural computation is a process governing the change/dynamics of information. Information and computation are two mutually defining concepts as argued by Dodig-Crnkovic [44], which are conceptually combined based on their complementarity, in the concept of "*info-computation*" [25].

Thus, to specify computational information operator within classical model of computation, it is necessary to delineate computation accordingly.

In the same way as there are varieties of concepts and frameworks for information, there are many approaches to descriptions and definitions of computation, demonstrating that defining computation is still an unsolved problem, as argued by Burgin and Dodig-Crnkovic [45].

There are three levels of generality in understanding the phenomenon of computation:

1. On the top (most general) level, computation is perceived as any transformation of information and/or information representation.
2. On the middle level, computation is distinguished as a *discrete* process of transformation of information and/or information representation.
3. On the bottom level, computation is defined as a *discrete* process of *symbolic transformation* of information and/or symbolic information representation in case of classical computation models. Alternatively, in case of natural and unconventional computing, physical/chemical/biological/cognitive processes

that are interpreted as computation or the basis for computational behaviors of physical systems under consideration are at the bottom level.

It is necessary to remark that if we do not go beyond the bottom level and if we insist on discreteness, we would lose continuous time computation realized by general dynamical systems of Bournez [46], hybrid systems of Gupta et al [47], and special computing devices, such as the differential analyzer of Shannon [48]; Moore [49]).

Computation is traditionally defined as transformation of information representation, see, e.g., Kelemen [50], where transformations can be discrete, continuous, or a mixture.

This definition of computation results in separation of substantial types of computation, as explained by Burgin and Dodig-Crnkovic in [45]:

1. *Symbolic computation* when information is represented by physically- or mentally-given symbols.
2. *Material computation* when information is represented by material objects, such as atoms, or molecules of a biological cell, and can be continuous as there are continuous phenomena in many branches of physics.

Typical artificial devices, such as conventional computers and calculators, perform material computations, which represent symbolic computation that is in focus. The same process of symbolic computation can be realized by different material computations, e.g., on different computers. That is the case of universal or general-purpose computers, which are substrate-independent.

There is another type of computer, the analog computer, which is a model for a certain problem that can then be used to solve analog problem by means of simulating it. In the analog computer there is no stored program controlling its operation. Instead, it is programmed by changing the interconnections between computing elements. This type of computation/information processing is similar to the information processing in the human brain.

Quantum computation is either a kind of symbolic computation embodied in material computation where symbols are represented by quantum states or an analog computation called quantum annealing, which is an optimization of the cost or energy functions of complex systems utilizing quantum fluctuations. This approach is used by D-wave computers, who recently claimed attaining quantum supremacy with a 53-qubit superconducting processor [51].

It is sometimes considered an open question whether symbolic computation is possible without material computation, even though the majority of researchers believe that there is no information without physical representation and there is no computation without information as argued by Szilard, Landauer, Swenson, and Lloyd as quoted by Karnani, Pääkkönen, and Annila [52]. Material computation is possible not only in computers as technological artifacts but in a computing nature [38] as a whole. An example of material computation is all physical computation that goes on in all kinds of physical objects, including living cells and living organisms. Neurons organized in neural networks are the only living cells capable of symbolic computation. Ehresmann [53] presents an info-computational model for (neuro-)cognitive systems capable of creativity built on several levels of organization/abstraction.

At the middle level of abstraction, computation is a discrete process of transformation of information and/or information representation reflected by results in three operational types of computation as presented by Burgin [54] and in the taxonomy of computation and information architecture by Burgin and Dodig-Crnkovic [55]:

1. *Discrete computation* with digital operations performed in elementary separate steps.
2. *Continuous computation* when operation goes without breaks in time.
3. *Piecewise continuous* computation, combining discrete and continuous computation.

In addition, we have three temporal types of computation [54,55]:

1. *Sequential computation*, which is performed in linear time.

2. *Parallel or branching* computation, in which separate steps are synchronized in time.
3. *Concurrent computation*, which does not have synchronization in time.

While parallel computation is completely synchronized, branching computation is not completely synchronized because separate branches acquire their own time and become synchronized only in interactions.

Existence of various types and kinds of computation, as well as a variety of approaches to the concept of computation, shows complexity of understanding of computation in a holistic picture.

In what follows, we consider computational information operators that represent algorithms or computing automata. Analog computing, which is not covered here, is presented in the section on natural computing.

The concept of computation stratifies the system of operands for computational information operators. Namely, if A is a computational information operator, its system of operands consists of three components:

1. Input, or initial, operands
2. Processed operands
3. Output, or resulting, operands

In turn, output (resulting) operands are divided into intermediate, final and analytic outputs of the operator in the form of an algorithm or computing automaton. Final outputs are the results of the computation produced by the algorithm or computing automaton. Analytic outputs are results of the computation, which are not produced by the algorithm or computing automaton but are determined theoretically. As an example of this situation, we consider limit Turing machines, which were defined by Burgin [54,56]. All other outputs are called intermediate.

With regard to outputs, computational information operators have three types:

1. *Explicit computational information operators* represent algorithms or computing automata, which function so that the last output is final and/or it is identified by the algorithm or computing automaton.
2. *Implicit computational information operators* represent algorithms or computing automata, which function so that the final output is not always identified by the algorithm or computing automaton.
3. *Analytic computational information operators* produce analytic outputs.

For instance, Turing machines are represented by explicit computational information operators, inductive Turing machines are represented by implicit computational information operators, and limit Turing machines are represented by analytic computational information operators.

There is a variety of techniques for composition of algorithms and computing devices [55]. These compositions induce corresponding compositions of computational information operators. The most popular of them is sequential composition, definition of which is given in Section 7.

Proposition 1. *The sequential composition of explicit computational information operators is an explicit computational information operator.*

Indeed, taking two explicit computational information operators, we see that if the first operator has a final output identified by the algorithm or computing automaton because it is an explicit computational information operator then this output goes to the second operator as its input. As the second operator is also explicit, in the case of producing the final output, this output is identified by the algorithm or computing automaton. It means that the sequential composition of these operators is an explicit computational information operator.

Proposition 1 means that that the class of explicit computational information operators is closed with respect of sequential composition.

At the same time, examples show that for explicit and analytic computational information operators, this statement is not valid in a general case.

According to the results of their application, there are three categories of computational information operators, which are defined by their results on valid inputs, that is, inputs such that application of the operator gives results:

1. A single-valued computational information operator produces at most one result for any valid input.
2. A finite-valued computational information operator can produce a finite number of results for any valid input.
3. An infinite-valued computational information operator can produce an infinite number of results for some valid input.

Deterministic computing automata and algorithms are represented by single-valued computational information operators, while nondeterministic computing automata and algorithms are usually represented by finite-valued or infinite-valued computational information operators. However, nondeterministic accepting automata and algorithms are also represented by single-valued computational information operators.

Finite-valued computational information operators are also represented by multiple computations studied by Burgin in [57].

Proposition 2. *The sequential composition of single-valued computational information operators is a single-valued computational information operator.*

Indeed, if the first operator is single valued, it produces, at most, one result, which serves as the input to the second operator in the composition, which also produces, at most, one result because it is also single-valued.

Proposition 2 means that that the class of single-valued computational information operators is closed with respect to sequential composition.

Proposition 3. *The sequential composition of finite-valued computational information operators is a finite-valued computational information operator.*

Proof is similar to the proof of Proposition 2.

Proposition 3 means that that the class of finite-valued computational information operators is closed with respect of sequential composition.

At the same time, examples show that, for infinite-valued computational information operators, this statement is not valid in a general case.

Note that a valid input to two operators can be invalid for their composition.

5. Computational Information Operators and Natural Computation

An operator formulation of Info-computational framework and its application on natural computation:

$$\text{(Information State Final)} = \text{Operator} \times \text{(Information State Initial)}$$

In this case, the general structure (Operator, Operation, Operand) takes the form of the triad:

$$\text{(Information StateInitial, Computation, Information StateFinal)}$$

Burgin and Dodig-Crnkovic [25] argue that information in the world appears on a multiple scales or levels of organization or levels of abstraction as well as in multiple dimensions.

In natural computing or computing nature [39], the whole of the nature is seen as a network of networks of computational processes on different levels of organization. Similarly, Zenil [58] presents the idea of a computable universe with both ambition to understand computation and exploring nature as computation.

The idea of natural info-computation [59] as a dynamics of information systems successively connects and relates their information structures (states) and makes it possible for a cognizing agent to get an idea about the information content of the world that gets revealed in the agent after sequential processes of information transformation from the source to the receiver. In [26], Dodig-Crnkovic, presents a framework of computing nature [39], where nature is regarded as a network of networks of morphological info-computational processes unfolding through information processing for cognitive agents. "Morphological" stands for a computing model based on information about the form or structure of material system performing computation [6].

Natural computation is a process studied within a field of natural computing or computing nature [39]. As elaborated in the *Handbook of Natural Computing* [17], it consists of three classes of methods:

(1) inspired by nature for the development of novel problem-solving techniques (e.g., cellular automata, neural computation, evolutionary computation, swarm intelligence, artificial immune systems, membrane computing, amorphous computing, cellular computing, molecular computing)
(2) based on the use of computers to synthesize/simulate natural phenomena (e.g., artificial life, artificial chemistry); and
(3) using natural materials (e.g., molecules) to compute (e.g., molecular computing or quantum computing).

Nature is increasingly modeled as computational i.e., information processing system in many research fields, such as systems biology, synthetic biology, cellular computing, cognitive computing, social computing, and morphological computing.

In two research fields of morphological computing—within robotics and a more general one that Turing started with his morphogenesis paper, the relational character of information structures and their dynamics makes it suitable for the application of operator formalism. Fields of application include neuroscience, neurobiology, information processes in neurons and neural systems, bioinformatics, computational biology, learning, memory, neuron, synapse, and biological information systems.

As mentioned, quantum computing is one of the important fields of natural computing. Digital quantum computing uses quantum logic gates to perform computation. It has the advantages of universality, scalability, and quantum error correction, but physical resource requirements to implement error-corrected quantum algorithms are huge. Analog quantum computing (quantum simulation, quantum annealing, and adiabatic quantum computation) is used to avoid the complexity of classical simulations of many-body quantum systems which grows exponentially with the dimension of the system. Feynman suggested the simulation of these problems by another fully-controllable quantum system with a similar encoded dynamics. Utilization of different models of quantum computing involving dissimilar quantum theories makes it an important problem to develop a unified operator theory of quantum computing.

6. Computational Information Operators as an Efficient Tool in Computer Science

Let us consider advantages and possibilities opened by operator representation of *computing devices*.

First possibility: Operator representation of computing devices allows formulating and solving many problems about these of computing devices in a more general context of operating spaces of operators.

An example of such a problem is the *Definability Problem*, which is called the Halting Problem for Turing machines because definability for a Turing machine is equivalent to halting. In particular, according to the results of Church and Turing, λ-definable functions are functions that are "effectively" (by mechanical methods) computable. Turing showed that the class of all Turing machines is equivalent to the class of all λ-definable functions. This was an important step in recognition of Turing machine as the supreme model of algorithm.

Let us consider the Definability Problem for operators.

Definability Problem. Given a class **K** of operators, is there an operator B in **K** such that for any element x from the domain of operators from **K** and any operator A from **K,** B determines whether x belongs to the definability domain of A or does not belong.

Second possibility: Operator representation of computing devices allows constructing a variety of operator compositions (operations) and developing new schemas of computation as well as new network and computer architectures using operations with (a composition of) operators.

Let us look at two examples of such compositions.

Example 1. *Sequential composition*

Given two operators A and B, their sequential composition is the operator C such that C(x) is equal to B(A(x)) when:

(1) A(x) is defined and belongs to the domain of B;
(2) B(A(x)) is defined.

Otherwise, C gives no result being applied to x.

Example 2. *Disjunctive parallel composition*

Given two operators A and B, their disjunctive parallel composition is the operator H such that the result of application of D to any operand u is performed so that A and B are applied to u at the same time and D(u) = A(u) if A gets its result at the same time or earlier than B; otherwise, D(u) = B(u).

Third possibility: Operator representation of computing devices allows efficient application of the axiomatic technique for investigation of computing devices, algorithms and computations.

Let us consider a class **K** of algorithms (computing devices) and the corresponding class $\mathbf{O_K}$of operators, which model algorithms (computing devices) from **K**.Here are some examples of axioms, which characterize the class $\mathbf{O_K}$.

Totality axiom: For any operator A from $\mathbf{O_K}$, $D(A) = DD(A)$.
Domain stability axiom: For any operators A and B from $\mathbf{O_K}$, $D(A) = D(B)$.
Domain loop axiom: For any operator A from $\mathbf{O_K}$, $D(A) = C(A)$.

This allows obtaining similar axioms for the class **K**.

Totality axiom: For any computing device (algorithm) R from **K**, $D(A) = DD(A)$.
Domain stability axiom: For any computing devices (algorithms) R and Q from **K**, $D(A) = D(B)$.
Domain loop axiom: For any computing device (algorithm) R from **K**, $D(A) = C(A)$.

The axiomatic theory of algorithms and computations has been created and developed in the context of functions [60–62]. The transition from functions to operators allows essentially expand the axiomatic theory of algorithms including quantum and natural computations as well as algorithmic functioning of structural machines [63].

Note that it is possible to apply these axioms to algebras of operators such as von Neumann algebras [64] or Kleene algebras [65].

This was sufficient for computations with simple structures, e.g., symbols or words, with which finite automata and Turing machines work. When computational structures become more sophisticated, as in the case of quantum computers, morphological computations and structural machines, it is necessary to utilize operators to represent computational media, computing devices, and computations.

7. Conclusions

Operator representation of computing processes—both abstract and physical—allows the application of mathematical operator theory to automata, algorithms, and computations. Even though all computation is always performed on some physical substrate, including Turing computation that presupposes a human computer with a pencil and rubber and a piece of paper, classical theory of computing does not investigate its physical substrate and it is developed exactly to *abstract from the detail of physical implementation*. That is how we can run programs on a variety of computational devices, from smallest sensors equipped with some control to supercomputers and various networks.

Classical computing machinery that we have today is energy consuming, non-resilient, and it is not very well suited for simulations of complex quantum systems or representation of the information processing in the brain or in large and interconnected economic or social systems. Thus, we are interested in not only general-purpose, substrate-independent abstract types of computation, but also such computational devices that are dedicated to specific computational problems, such as quantum computers or cognitive computational devices.

An interesting problem is to build operator spaces and to study operators in these different spaces, which represent different classes of systems with different computational characteristics.

The present study demonstrates how to build a variety of novel operations with operators. At the same time, operator algebras studied in mathematics employ only classical (standard) operations, such as sequential composition.

This brings us to one more interesting problem of construction and explorations of operator algebras with nonstandard operations, which, at the same time, may be a contribution to the research in mathematics and in the theory of computing.

Author Contributions: Both authors have equally contributed to the article, its conceptualization, methodology, investigation, writing—original draft preparation, review, and editing. Both authors have read and agreed to the published version of the manuscript. All authors have read and agreed to the published version of the manuscript.

Funding: The research of the one co-author, GD-C, was funded by Swedish Research Council grant MORCOM@COGS.

Acknowledgments: We would like to acknowledge the contribution from the colleagues at IS4SI 2019 conference Theoretical Information Studies in Berkeley where the first version of this work was presented for interesting discussions. Remarks of the reviewers were also useful for the improvement of the paper.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Eilenberg, S.; MacLane, S. Relations between homology and homotopy groups of spaces. *Ann. Math.* **1945**, *46*, 480–509. [CrossRef]
2. Bourbaki, N. *Elements de Mathematique. Theorie des Ensembles*; Hermann: Paris, France, 1960.
3. Robinson, A. *Introduction to Model Theory and Metamathematics of Algebra*; North-Holland Publishing: Amsterdam, The Netherlands, 1963.
4. Goldblatt, R. *Topoi: The Categorical Analysis of Logic*; North Holland Publishing: Amsterdam, The Netherlands, 1979.
5. Burgin, M. Epistemic Information in Stratified M-Spaces. *Information* **2011**, *2*, 697–726. [CrossRef]
6. Dodig-Crnkovic, G. Nature as a Network of Morphological Infocomputational Processes for Cognitive Agents. *Eur. Phys. J. Spec. Top.* **2017**, *226*, 181–195. [CrossRef]
7. Von Neumann, J. *Mathematical Foundations of Quantum Mechanics*; Princeton University Press: Princeton, NJ, USA, 1955.
8. Exner, P.; Havlíček, M. *Hilbert Space Operators in Quantum Physics*; Springer: New York, NY, USA, 2008.
9. Brown, A.; Pearcy, C. *Introduction to Operator Theory I: Elements of Functional Analysis*; Springer-Verlag: New York, NY, USA, 1977.

10. Ball, J.A.; Bolotnikov, V.; Helton, J.W.; Rodman, L. (Eds.) *Topics in Operator Theory (Operator Theory: Advances and Applications)*; BirkhäuserVerlag: Basel, Switzerland, 2010.
11. Burgin, M. *Semitopological Vector Spaces: Hypernorms, Hyperseminorms and Operators*; Apple Academic Press: Toronto, ON, Canada, 2017.
12. Burgin, M. Weighted E-Spaces and Epistemic Information Operators. *Information* **2014**, *5*, 357–388. [CrossRef]
13. Harris, Z. *A Theory of Language and Information: A Mathematical Approach*; Oxford University Press: Oxford, UK, 1991.
14. Brenner, J.; Burgin, M. Information as a Natural and Social Operator. *Inform. Theor. Appl.* **2011**, *18*, 33–49.
15. Burgin, M.; Brenner, J. Operators in Nature, Science, Technology, and Society: Mathematical, Logical, and Philosophical Issues. *Philosophies* **2017**, *2*, 21. [CrossRef]
16. Pingali, K.; Nguyen, D.; Kulkarni, M.; Burtscher, M.; Hassaan, M.A.; Kaleem, R.; Lee, T.H.; Lenharth, A.; Manevich, R.; Mendez-Lojo, M.; et al. The Tao of Parallelism in Algorithms. In *PLDI '11: Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation*; Association for Computing Machinery: New York, NY, USA, 2011; pp. 12–25. [CrossRef]
17. Rozenberg, G.; Back, T.; Kok, J. (Eds.) *Handbook of Natural Computing*; Springer Verlag: Berlin, Germany, 2012.
18. Adamatzky, A. (Ed.) *Unconventional Computing. A Volume in Encyclopedia of Complexity and Systems Science*, 2nd ed.; Springer Nature: Basel, Switzerland, 2018.
19. Burgin, M.; Adamatzky, A. Structural Machines as a Mathematical Model of Biological and Chemical Computers. *Theor. Appl. Math. Comput. Sci.* **2017**, *7*, 1–30.
20. Burgin, M.; Adamatzky, A. Structural machines and slime mold computation. *Int. J. Gen. Syst.* **2017**, *45*, 201–224. [CrossRef]
21. Levine, I.N. *Quantum Chemistry*, 4th ed.; Prentice Hall: Englewood Cliffs, NJ, USA, 1991.
22. McQuarrie, D.A. *Quantum Chemistry*; University Science Books: Mill Valey, CA, USA, 1983.
23. What Does Operator Mean (in Computer Programming)? *Techopedia.* Available online: https://www.techopedia.com/definition/3485/operator-programming (accessed on 8 May 2020).
24. Burgin, M. Triadic Structures in Interpersonal Communication. *Information* **2018**, *9*, 283. [CrossRef]
25. Burgin, M.; Dodig-Crnkovic, G. A Multiscale Taxonomy of Information in the World. In *Theoretical Information Studies. Information in the World*; World Scientific Series in Information Studies; Burgin, M., Dodig-Crnković, G., Eds.; World Scientific: Singapore, 2019; Volume 11.
26. Dodig-Crnkovic, G. Cognition as Embodied Morphological Computation. In *Philosophy and Theory of Artificial Intelligence 2017*; PT-AI 2017. Studies in Applied Philosophy, Epistemology and Rational Ethics; Müller, V., Ed.; Springer: Cham, Switzerland, 2018; Volume 44.
27. Alcami, P.; El Hady, A. Axonal Computations. *Front. Cell. Neurosci.* **2019**, *13*, 413. [CrossRef]
28. Silver, A.R. Neuronal arithmetic. *Nat. Rev. Neurosci.* **2010**, *11*, 474–489. [CrossRef] [PubMed]
29. Burgin, M.; Schumann, J. Three Levels of the Symbolosphere. *Semiotica* **2006**, *160*, 185–202. [CrossRef]
30. Brooks, R.A. Intelligence without representation. *Artif. Intell.* **1991**, *47*, 139–159. [CrossRef]
31. Kolmogorov, A.N. On the Concept of Algorithm. *Rus. Math. Surv.* **1953**, *8*, 175–176.
32. Codd, E.F. *Cellular Automata*; Academic Press: New York, NY, USA, 1968.
33. Bar-Hillel, Y.; Carnap, R. Semantic Information. *Br. J. Philos. Sci.* **1958**, *4*, 147–157. [CrossRef]
34. Gärdenfors, P. *Conceptual Spaces: The Geometry of Thought*; MIT Press: Cambridge, MA, USA, 2000.
35. Gärdenfors, P. Conceptual Spaces as a Framework for Knowledge Representation. *Mind Matter* **2004**, *2*, 9–27.
36. Gärdenfors, P. Cognitive semantics and image schemas with embodied forces. In *Embodiment in Cognition and Culture*; Krois, J.M., Rosengren, M., Steidele, A., Westerkamp, D., Eds.; Benjamins: Amsterdam, The Netherlands, 2007; pp. 57–76.
37. Shreider, Y.A. On Semantic Aspects of Information Theory. *Inform. Cybern.* **1967**, 15–47. (In Russian)
38. Burgin, M.; Díaz-Nafría, J.M. Introduction to the Mathematical Theory of Knowledge Conceptualization: Conceptual Systems and Structures. In Proceedings of the Second International Conference on Applied Informatics (ICAI 2019), Communications in Computer and Information Science Book Series (CCIS, Volume 1051), Madrid, Spain, 7–9 November 2019; pp. 469–482.
39. Dodig-Crnkovic, G.; Giovagnoli, R. Computing Nature—A Network of Networks of Concurrent Information Processes. In *Computing Nature*; Springer: Heidelberg, Germany, 2013; pp. 1–22.
40. Dodig-Crnkovic, G. Dynamics of Information as Natural Computation. *Information* **2011**, *2*, 460–477. [CrossRef]

41. Burgin, M.; Dodig-Crnkovic, G. From the Closed Classical Algorithmic Universe to an Open World of Algorithmic Constellations. In *Computing Nature*; SAPERE Book Series; Springer: Heidelberg, Germany, 2013; pp. 241–253, arXiv:1211.4547.
42. Pfeifer, R.; Bongard, J. *How the Body Shapes the Way We Think: A New View of Intelligence*; MIT Press: Cambridge, UK, 2006.
43. Bohan Broderick, P. On Communication and Computation. *Minds Mach.* **2004**, *14*, 1–19. [CrossRef]
44. Dodig-Crnkovic, G. *Investigations into Information Semantics and Ethics of Computing*; Mälardalen University Press: Västerås, Sweden, 2006.
45. Burgin, M.; Dodig-Crnkovic, G. Information and Computation—Omnipresent and Pervasive. In *Information and Computation*; World Scientific: New York, NY, USA, 2011; pp. vii–xxxii.
46. Bournez, O. Achilles and the tortoise climbing up the hyper-arithmetical hierarchy. *Theor. Comput. Sci.* **1999**, *210*, 21–71. [CrossRef]
47. Gupta, V.; Jagadeesan, R.; Saraswat, V.A. Computing with Continuous Change. *Sci. Comput. Program.* **1999**, *30*, 3–49. [CrossRef]
48. Shannon, C. Mathematical Theory of the Differential Analyzer. *J. Math. Phys. MIT* **1941**, *20*, 337–354. [CrossRef]
49. Moore, C. Recursion Theory on the Reals and Continuous-time Computation: Real numbers and computers. *Theor. Comput. Sci.* **1996**, *162*, 23–44. [CrossRef]
50. Kelemen, J. On a Possible Future of Computationalism. In Proceedings of the 7th International Symposium of Hungarian Researchers on Computational Intelligence, HUCI'06, Budapest, Hungary, 24–25 November 2006; pp. 51–56.
51. Bera, R.K. *The Amazing World of Quantum Computing*; Springer: Singapore, 2020.
52. Karnani, M.; Pääkkönen, K.; Annila, A. The physical character of information. *Proc. R. Soc. A* **2009**, *465*, 2155–2175. [CrossRef]
53. Ehresmann, A.C. MENS, an Info-Computational Model for (Neuro-)cognitive Systems Capable of Creativity. *Entropy* **2012**, *14*, 1703–1716. [CrossRef]
54. Burgin, M. *Super-Recursive Algorithms*; Springer: New York, NY, USA, 2005.
55. Burgin, M.; Dodig-Crnkovic, G. A Taxonomy of Computation and Information Architecture. In Proceedings of the 2015 European Conference on Software Architecture Workshops, Dubrovnik/Cavtat, Croatia, 7–11 September 2015; ACM: New York, NY, USA. [CrossRef]
56. Burgin, M. Universal Limit Turing Machines. *Not. Russ. Acad. Sci.* **1992**, *325*, 654–658.
57. Burgin, M. Multiple computations and Kolmogorov complexity for such processes. *Not. Acad. Sci. USSR* **1983**, *27*, 793–797.
58. Zenil, H. *A Computable Universe. Understanding Computation & Exploring Nature as Computation*; World Scientific Publishing Company/Imperial College Press: Singapore, 2012.
59. Dodig-Crnkovic, G. Physical Computation as Dynamics of Form that Glues Everything Together. *Information* **2012**, *3*, 204–218, Special Issue on Information: Its Different Modes and Its Relation to Meaning. [CrossRef]
60. Burgin, M. *Measuring Power of Algorithms, Computer Programs, and Information Automata*; Nova Science Publishers: New York, NY, USA, 2010.
61. Burgin, M. Decidability and Universality in the Axiomatic Theory of Computability and Algorithms. *Int. J. Found. Comput. Sci.* **2012**, *23*, 1465–1480. [CrossRef]
62. Dodig-Crnkovic, G.; Burgin, M. Axiomatic Tools versus Constructive approach to Unconventional Algorithms. In Proceedings of the Symposium on Natural Computing/Unconventional Computing and its Philosophical Significance, AISB/IACAP World Congress, Birmingham, UK, 2–6 July 2012.
63. Burgin, M. Information Processing by Structural Machines. In *Theoretical Information Studies: Information in the World*; World Scientific: Singapore, 2020; pp. 323–371.
64. Blackadar, B. *Operator Algebras*; Springer: New York, NY, USA, 2005.
65. Kozen, D. A Completeness Theorem for Kleene Algebras and the Algebra of Regular Events. *Inform. Comput.* **1994**, *110*, 366–390. [CrossRef]