Cellular Engineering

M. Burgin

Department of Mathematics
University of California, Los Angeles
405 Hilgard Avenue
Los Angeles, CA 90095

The main goal of this paper is to develop tools for constructing different kinds of abstract automata based on cellular automata. We call this engineering problem cellular engineering. Different levels of computing systems and models are considered. The emphasis is made on the top-level model called a grid automaton. Our goal is to construct grid automata using cellular automata. To achieve this, we develop a specific technology based on multilevel finite automata. It is proved that two-dimensional cellular automata allow the construction of some types of grid automata, as well as Turing machines and pushdown automata.

1. Introduction

The goal of this work is to study relations between uniform and non-uniform computational systems and develop tools for representing, modeling, and building different computational schemas and devices (from simple to more advanced, such as grid automata) by means of cellular automata. This is an engineering problem that we call *cellular engineering*. Here are the three main types of cellular engineering.

- *Process cellular engineering* is aimed at building a cellular automaton to reproduce, organize, model, or simulate some process.
- *Function cellular engineering* is aimed at building a cellular automaton to reproduce, organize, model, or simulate some function.
- *System cellular engineering* is aimed at building a cellular automaton to reproduce or model some system with its subsystems, components, and elements.

Traditional engineering problems solved with cellular automata are mostly related to cellular process organization or reproduction, that is, how to get a process with all necessary characteristics represented by a cellular automaton. Sometimes functions are modeled, like when cellular automata are used to model the functioning of a Turing machine. System cellular engineering reproduces (models) a system with some level of detail. For instance, it is possible to represent a system at the level of its elements or at the level of its components.

The area of cellular automata can be divided into three big subareas: CA science, CA computation, and CA engineering. CA science

https://doi.org/10.25088/ComplexSystems.18.1.103

studies properties of cellular automata and, in particular, their dynamics, or how they function. CA computation uses cellular automata for computation, simulation, optimization, and generation of evolving processes. CA engineering is aimed at constructing different devices from cellular automata. All three areas are complementary to one another.

Cellular automata are the simplest uniform models of distributed computations and concurrent processes. Grid automata are the most advanced and powerful models of distributed computations and concurrent processes, which synthesizes different approaches to modeling and simulating such processes [1, 2].

A *grid automaton* is a system of automata that are situated in a grid and called *nodes*. Some of these automata are connected and interact with one another.

Cellular automata are special cases of grid automata; although, in general, grid automata are nonuniform. Our goal is not to substitute cellular automata by grid automata, but to use cellular automata as the basic level for building hierarchies of grid automata. The reason for doing this is to reduce the complexity of the description of the system and its processes. For instance, a computer has several levels of hierarchy: from the lowest logic gate level to the highest level of functional units, such as system memory, CPU, keyboard, monitor, and printer. In addition, as Clark writes (cf. [3]), "all good computer scientists worship the god of modularity, since modularity brings many benefits, including the all-powerful benefit of not having to understand all parts of a problem at the same time in order to solve it." That is why another goal of this paper is to introduce modularity into the realm of cellular automata. This allows a better understanding and more flexible construction tools without going into detailed exposition of the lower system levels. As a result, we develop a computing hierarchy based on cellular automata.

Cellular engineering is a complimentary approach to evolutionary simulation and optimization. Evolutionary simulation is aimed at modeling complex behavior by simple systems, such as cellular automata. Evolutionary optimization is directed toward improving systems by simple automata, such as cellular automata, which imitate natural evolutionary processes. Cellular engineering is aimed at constructing complex systems using simple systems, such as cellular automata. In evolutionary processes, systems are evolving subject to definite rules. In engineering, systems are purposefully constructed according to a designed plan.

Function cellular engineering is the weakest form, while system cellular engineering is the strongest form of cellular engineering. Indeed, building a system with necessary properties solves the problem of creating a process with necessary features, while the latter solves the problem of constructing a function with necessary characteristics. Usually only function cellular engineering has been considered, for example, when cellular automata computed the same function as a Turing machine [4]. Here we emphasize system cellular engineering. Results

https://doi.org/10.25088/ComplexSystems.18.1.103

obtained in this paper support the conjecture of Wolfram [4] that a diversity of complex systems can be obtained from cellular automata. In addition, as von Neumann demonstrated [5], construction of complex systems using cellular automata allows one to essentially increase reliability of these systems.

It is necessary to remark that different authors studied hierarchical cellular automata, for example, [6, 7]. We expand hierarchical computational structures beyond the level of cellular automata, taking cellular structure as the base.

1.1 Denotations

If X is a set, then $|X|$ is the number of elements in X.

If X and Y are sets, then $X \times Y$ is the direct product of these sets.

2. A Hierarchy of Computational Levels

The hierarchy of computational levels studied in this paper is similar to the hierarchy in the physical, biological, and social levels of nature. As we know, the latter consists of these levels: (1) subatomic particles; (2) atoms; (3) molecules; (4) rigid bodies, liquids, and gases; (5) living cells; (6) organs; (7) organisms; (8) living beings; and (9) societies.

It is possible to build a similar hierarchy in the world of computing to give us the following material computational-universe hierarchy.

1. Examples of computational subatomic particles are gates in computing devices.
2. Systems of gates that realize computing operations, such as addition or multiplication, are atoms.
3. Chips are molecules.
4. The hardware of computers represent rigid bodies, while embedded devices are liquids and gases.
5. Systems of gates with programs that realize some computing operations, such as addition or multiplication, are living cells.
6. Computer components, such as a CPU, memory, display, CD, mouse, or notepad, are organs of the organisms.
7. Computers are organisms.
8. Computer networks are societies.

Here is the abstract computational hierarchy.

1. Examples of abstract computational subatomic particles are Boolean circuits and binary neurons of McCulloch and Pitts [8]. For instance, Minsky [9] demonstrated how to build Turing machines that are computational "rigid bodies" from such "particles".
2. Finite automata and general artificial neurons are abstract computational atoms.

https://doi.org/10.25088/ComplexSystems.18.1.103

3. Cellular automata, Petri nets, and artificial neural networks are three different types of abstract computational molecules.
4. Turing machines, inductive Turing machines, random access machines, alternating Turing machines, and many other recursive algorithms are rigid bodies, liquids, and gases of the abstract computational universe.
5. Grid automata and other network models are systems of rigid bodies in the abstract computational universe.

The goal of this paper is to build different rigid bodies and their systems from computational molecules such as cellular automata.

3. Grid Automata

3.1 General Constructions

In comparison with cellular automata, a grid automaton can contain different kinds of automata as its nodes. For example, finite automata, Turing machines, and inductive Turing machines can belong to one and the same grid. In comparison with systolic arrays, connections between different nodes in a grid automaton can be arbitrary, like connections in neural networks. In comparison with neural networks and Petri nets, a grid automaton contains, as its nodes, more powerful machines than finite automata. Consequently, neural networks, cellular automata, systolic arrays, and Petri nets are special kinds of grid automata. An important property of grid automata is the possibility of realizing hierarchical structures; that is, a node can be also a grid automaton. In grid automata, interaction and communication becomes as important as computation. This peculiarity results in a variety of automata types, their functioning modes, and space organization.

Forming the highest structural level in the computational hierarchy, grid automata formalize a rather simple idea.

Definition 1. A *grid automaton* is a system of automata, which are situated in a grid, are called nodes, are optionally connected, and interact with one another.

Making this semiformal definition mathematical, we come to two types of grid automata: basic grid automata and grid automata with ports. The latter are simply called grid automata.

The basic idea of interacting processes is for a transmitting process to send a message to a port and for the receiving process to get the message from another port. To formalize this structure, we assume, as it is often true in reality, that connections are attached to automata by means of ports. Ports are specific automaton elements through which data come into (*input ports* or *inlets*) and send outside the automaton (*output ports* or *outlets*). Thus, any system $\boldsymbol{P}$ of ports is the union of its two (possibly) disjoint subsets $\boldsymbol{P} = \boldsymbol{P}_{\text{in}} \cup \boldsymbol{P}_{\text{out}}$ where $\boldsymbol{P}_{\text{in}}$ consists of all inlets from $\boldsymbol{P}$ and $\boldsymbol{P}_{\text{out}}$ consists of all outlets from $\boldsymbol{P}$. If there are

https://doi.org/10.25088/ComplexSystems.18.1.103

ports in the real system that are both inlets and outlets, in the model, we split them into pairs consisting of an input port and an output port. There are other different types of ports. For example, contemporary computers have parallel and serial ports. Ports can have an inner structure, but in this first approximation, it is possible to consider them as elementary units.

We also assume that each connection is directed; that is, it has a beginning and an end. It is possible to build bidirectional connections from directed connections.

Definition 2. A *grid automaton* G is the following system that consists of three sets and three mappings:

$$G = \left(\boldsymbol{A}_G, \boldsymbol{P}_G, \boldsymbol{C}_G, p_{IG}, c_G, p_{EG}\right).$$

Here the set $\boldsymbol{A}_G$ is the set of all automata from G. $\boldsymbol{C}_G$ is the set of all links from G. The set $\boldsymbol{P}_G = \boldsymbol{P}_{IG} \bigcup \boldsymbol{P}_{EG}$ (with $\boldsymbol{P}_{IG} \bigcap \boldsymbol{P}_{EG} = \emptyset$) is the set of all ports of G, $\boldsymbol{P}_{IG}$ is the set of *internal ports* of the automata from $\boldsymbol{A}_G$, and $\boldsymbol{P}_{EG}$ is the set of *external ports* of G, which are used for the interaction of G with different external systems. $p_{IG} : \boldsymbol{P}_{IG} \to \boldsymbol{A}_G$ is a total function, called the *internal port assignment function*, that assigns ports to automata. $c_G : \boldsymbol{C}_G \to \left(\boldsymbol{P}_{IG\text{out}} \times \boldsymbol{P}_{IG\text{in}}\right) \bigcup \boldsymbol{P}'_{IG\text{in}} \bigcup P''_{IG\text{out}}$ is a (eventually, partial) function, called the *port-link adjacency function*, that assigns connections to ports where $\boldsymbol{P}'_{IG\text{in}}$ and $P''_{IG\text{out}}$ are disjunctive copies of $\boldsymbol{P}_{IG\text{in}}$. $p_{EG} : \boldsymbol{P}_{EG} \to \boldsymbol{A}_G \bigcup \boldsymbol{P}_{IG} \bigcup \boldsymbol{C}_G$ is a function, called the *external port assignment function*, that assigns ports to different elements from G.

If l is a link that belongs to the set $\boldsymbol{C}_G$ and $c_G(l)$ belongs to $\boldsymbol{P}_{\text{Gin}} \times \boldsymbol{P}_{\text{Gout}}$; that is, $c_G(l) = (p_1, p_2)$, it means that the beginning of l is attached to p_1, while the end of l is attached to p_2. Such a link is called *closed*. If l is a link from $\boldsymbol{C}_G$ and $c_G(l)$ belongs to $\boldsymbol{P}_{\text{Gin}}$ (or $\boldsymbol{P}_{\text{Gout}}$); that is, $c_G(l) = p_1 \in \boldsymbol{P}_{\text{Gin}}$ (correspondingly, $c_A(l) = p_2 \in \boldsymbol{P}_{\text{Gout}}$), it means that the beginning of l is attached to p_1 (correspondingly, the end of l is attached to p_2). Such a link is called *open*.

The automata from A_G are also called *nodes* of G, and connections/links from C_G are also called *edges* of G. Like ports, nodes and edges can be of different types. For instance, nodes in a grid automaton can be neurons, neural networks, finite automata, Turing machines, port automata [10], vector machines, array machines, random access machines (RAM), inductive Turing machines [2], fuzzy Turing machines [11], or others. Some of the nodes can also be grid automata.

As a result, elements from the set $\boldsymbol{A}_G$ have an inner structure. Besides, elements from the sets $\boldsymbol{P}_G$ and $\boldsymbol{C}_A$ can also have an inner structure. For example, a link or a port can be an automaton. If we con-

https://doi.org/10.25088/ComplexSystems.18.1.103

sider the Internet as a grid automaton with computers as nodes, then links include modems, servers, routers, and eventually some other devices. A network adapter is an example of a port with an inner structure.

Remark 1. To have meaningful assignments of ports, the port assignment functions p_{IG} and p_{EG} have to satisfy some additional conditions. For instance, it is necessary not to assign input ports of the automaton G to the end of any link in G. In the case of a neural network as a node of G, inner ports of G assigned to this network are usually connected to open links going to and from neurons. At the same time, it is possible to have such ports connected to neurons directly, as well as free ports that are not connected to any element of the network. Free ports might be useful for increasing the reliability of network connections to the environment. When some port fails, it would be possible by dynamically changing the assignment function to switch from the damaged port to a free port.

Taking the nervous system of a human being and representing it as a grid automaton with neurons as its nodes, it is natural to consider dendrites and axons as links: dendrites are input links and axons are output links. Then synaptic membranes are ports of this automaton: presynaptic membranes are outlets and postsynaptic membranes are inlets. Presynaptic membranes are axon terminals, that is, output ports are adjusted only to output links, while postsynaptic membranes are parts of dendrites and bodies of neurons, that is, input ports are adjusted both to nodes (automata) and to input links. Cell membranes in general and neuron membranes, in particular, are examples of ports with a complex inner structure.

Remark 2. Representing a grid automata without ports is the first approximation to a general network model [1], while representing a grid automata with ports is the second (more exact) approximation. In some cases, it is sufficient to use grid automata without ports, while in other situations to build an adequate, flexible, and efficient model, we need automata with ports as nodes of a grid automaton.

We now give a formal description of a grid automaton without ports.

Definition 3. A *basic grid automaton* A is the following system that consists of two sets and one mapping:

$$R = (\boldsymbol{A}_A,\ \boldsymbol{C}_A,\ c_A).$$

Here $\boldsymbol{A}_A$ is the set of all automata from A, $\boldsymbol{C}_R$ is the set of all connections/links from R, and $c_A : \boldsymbol{C}_A \to \boldsymbol{A}_A \times \boldsymbol{A}_A \bigcup \boldsymbol{A}'_A \bigcup \boldsymbol{A}''_A$ is a (variable) function, called the *node-link adjacency function*, that assigns connections to nodes where $\boldsymbol{A}'_A$ and $\boldsymbol{A}''_A$ are disjunctive copies of $\boldsymbol{A}_A$.

https://doi.org/10.25088/ComplexSystems.18.1.103

There are different types of connections. For instance, computer network links or connections are implemented on a variety of different physical media, including twisted pair, coaxial cable, optical fiber, and space (cf. [3]).

Grid automata are abstract models of grid arrays that are real (physical systems). These models are used to study properties of grid arrays, their functioning, and behavior.

What is possible to do with grid automata and how they function depends on their characteristics. A grid automaton B is described by three grid characteristics, three node characteristics, and three edge characteristics. Characteristics allow one to separate different classes of grid automata.

3.2 Grid Characteristics

1. The *space organization* or *structure* of the grid automaton B is the spatial structure in which nodes and connection of the automaton function. This space structure may be in physical space, reflecting where the corresponding information processing systems are situated, or it may be a mathematical structure defined by the node relations. We consider three levels of space structures in a schema: local, regional, and global. Sometimes these structures are the same, while in other cases they are different.

   The space structure of a grid automaton can be static or dynamic. The functioning of a grid consists of elementary operations, which can be discrete or continuous. In addition, these operations are organized so that they form definite cycles of computation and interaction. For instance, taking a finite automaton, we see that an elementary operation is the processing of a single symbol, while a cycle is the processing of a separate word. A cycle for a Turing machine is the process that goes from the start state to a final state of the machine. This gives us three kinds of space organization for a grid automaton: *static structure* that is always the same, *persistent dynamic structure* that may change between different cycles of computation, and *flexible dynamic structure* that may change at any time during a computation. Reflexive Turing machines [12] have flexible dynamic structure, while persistent Turing machines [13] and von Neumann automata [5], have persistent dynamic structure.

2. The *topology* of B is determined by the neighborhoods of the nodes. A neighborhood of a node is the set of those nodes with which this node directly interacts. In a physical grid these are often the nodes that are the closest to the node in question. For example, if each node has only two neighbors, one on its right and one on its left, the topology is either linear or circular. The topology of computer networks is an example of the grid automaton topology [14].

   Here are the three main types of grid automaton topology.

   - A *uniform topology*, in which neighborhoods of all nodes of the grid automaton have the structure.
   - A *regular topology*, in which the structure of different node neighborhoods is subjected to some regularity. For instance, the system

https://doi.org/10.25088/ComplexSystems.18.1.103

neighborhoods can be invariant with respect to gauge transformations similar to gauge transformations in physics (cf. [15]).

- An *irregular topology* where there is no regularity in the structure of different node neighborhoods.

Conventional cellular automata have a uniform topology. Cellular automata in the hyperbolic plane or on a Fibonacci tree [16] are examples of grid automata with a regular topology.

3. The *dynamics* of B determine the rules its nodes use to exchange information with each other and with the environment of B. For example, it is possible that there is an automaton A in B that determines when and how all automata in B interact. Then if the automaton A is equivalent to a Turing machine—that is, A is a recursive algorithm [2], and all other automata in the grid automaton B are also recursive—then B is equivalent to a Turing machine [1]. At the same time, when the interaction of Turing machines in a grid automaton B is random, then B is much more powerful than any Turing machine [1].

   Environmental interaction gives two classes of grid automata: *open grid automata* interact with the environment through definite connections, while *closed grid automata* have no interaction with the environment. For example, Turing machines are usually considered closed automata because they begin functioning from some start state and tape configuration, and finish functioning (if at all) in some final state and tape configuration, and do not interact with their environment.

   In turn, here are the three types of open grid automata.

   (a) Grid automata open only for receiving information from the environment are called *accepting grid automata* or acceptors.

   (b) Grid automata open only for sending their output to the environment are called *transmitting grid automata* or transmitters.

   (c) Grid automata open for both receiving information from and sending their output to the environment are called *transducing grid automata* or transducers.

   To be open, a grid automaton must have a definite topology. For instance, to be an acceptor, a grid automaton must have open input edges. Existence of free ports makes a closed grid automaton *potentially open* as it is possible to attach connections to these ports.

3.3 Node Characteristics

1. The *structure* of the node, including structures of its ports, reflects inner organization of this node and its external connections. For instance, a finite automaton as a node has the one node structure. The structure of a node that is a Turing machine can also be the one node structure if we do not separate different modules, for example, the head, tape, and control device, of the Turing machine. It is possible that nodes also have inner structure. For instance, dendrites as ports of a natural neuron have rather developed inner structure, which can be represented on different levels—from functional components to molecular and even atomic organization. The inner structure of a Turing machine includes such modules as the head, tape, and control device, as well as connec-

https://doi.org/10.25088/ComplexSystems.18.1.103

tions between these modules. In turn, the tape consists of cells and, thus, has its inner structure determined by connections between cells.

In particular, the structure of a node defines how ports are adjusted in the node. For instance, if a neural network is a node of the grid automaton A, inner ports of A are usually connected to links going to and from neurons. It is also possible to have ports connected to neurons directly, as well as free ports that are not connected to any element of the network. Free ports might be useful for the reliability of network connections to the environment.

In the case when a Turing machine T is a node of the grid automaton A, it is possible to connect inner ports of A to some cells of the tapes from T or to whole tapes. In the first case, external information coming to such input ports will be written in the adjusted cells, while output ports send the symbol written in those cells to another node. In the second case, external information coming to such input ports will be distributed on the corresponding tape by some rule, while an output port sends the word written on the tape to another node.

2. The *external dynamics* of the node determines the interactions of this node. According to this characteristic, there are three types of nodes: *accepting nodes* that only accept or reject their input; *generating nodes* that only produce some input; and *transducing nodes* that both accept some input and produce some input. Note that nodes with the same external dynamics can work in grids with various dynamics. Primitive ports do not change node dynamics. However, compound ports are able to influence processes not only in the node to which they belong but also in the whole grid automaton.

3. The *internal dynamics* of the node determines what processes go inside this node. For instance, the internal dynamics of a finite automaton are defined by its transition function, while the internal dynamics of a Turing machine are defined by its rules. Differences in internal dynamics of nodes are very important because, for example, a change in producing the output allows us to go from conventional Turing machines to much more powerful inductive Turing machines of the first order [2].

3.4 Edge Characteristics

1. The *external structure* of the edge reflects how this edge is connected in the grid automaton. According to this characteristic, there are three types of edges: a *closed edge*, both sides of which are connected to ports of the grid automaton; an *ingoing edge*, in which only the end side is connected to a port of the grid automaton; and an *outgoing edge*, in which only the beginning side is connected to a port of the grid automaton.

2. Properties and the *internal structure* of the edge reflect inner organization of this edge. According to the internal structure, there are three types of edges: a *simple channel* that only transmits data, a *channel with filtering* that separates a signal from noise, and a *channel with data correction*.

https://doi.org/10.25088/ComplexSystems.18.1.103

3. The *dynamics* of the edge determines edge functioning. For instance, two important dynamic characteristics of an edge are bandwidth, as the number of bits per second transmitted on the edge, and throughput, as the measured performance of the edge.

   Link properties separate all links into these three standard classes.

   (a) An *information link* is a channel for processed data transmission.

   (b) A *control link* is a channel for transmitting instructions.

   (c) A *process link* realizes control transfer and determines how the process goes by initiation of an automaton in the grid by another automaton (other automata) in the grid.

   Process links determine what to do, control links are used to instruct how to work, and information links supply automata with data in a process of grid automaton functioning.

Example 1. When a sequential composition of two finite automata A and B is built, these automata are connected by two links. One of them is an information link. Through this link, the result obtained by the first automaton A is transferred from the output port (open from the right edge) of A to the input port (open from the left edge) of B. In addition, A and B are connected by a control link. When the automaton A produces its result, it transfers control to B. However, this does not mean that A halts, it can immediately start a new cycle.

It is essential to remark that in some situations there are no control links between the automata in the composition and both automata are synchronized by data transfer.

Here are the three main categories of links with respect to a given grid automaton.

- External links connect other systems.
- Intermediate links connect nodes of this automaton to other systems.
- Internal links connect nodes of the given grid automaton.

Remark 3. Initiation of an automaton in the grid by a signal that comes through a control link is usually regulated by some condition(s). Examples of conditions are: (*a*) some automata in the grid have obtained their results, (*b*) the initiated automaton has enough data to start working, and (*c*) the number (level) of initiating signals is above a prescribed threshold. This is an event-driven functioning, which is usually contrasted with operating on a time-scale.

Example 2. Artificial neurons are initiated only when the combined effect of all their input signals is above the firing threshold. For a natural neuron, single excitatory postsynaptic potentials have amplitudes in the range of 1 millivolt (mV). The critical value for spike initiation is about 20 to 30 mV above the resting potential. In most neurons, four spikes are not sufficient to trigger an action potential. Instead, about 20 to 50 presynaptic spikes have to arrive within a short time window before postsynaptic action potentials are triggered.

https://doi.org/10.25088/ComplexSystems.18.1.103

Remark 4. Transmission of instructions from one automaton in the grid to another one can be realized by transmitting values of some control parameter.

3.5 Structures of Grid Automata

To represent structures of grid automata now and schemas later, we use oriented multigraphs and generalized oriented multigraphs.

Definition 4. [17]. An *oriented* or *directed multigraph* G has the following form:

$$G = (V, E, c).$$

Here V is the set of vertices or nodes of G, E is the set of edges of G, and $c : E \to V \times V$ is the edge-node *adjacency* or *incidence function*. This function assigns each edge to a pair of vertices so that the beginning of each edge is connected to the first element in the corresponding pair of vertices, and the end of the same edge is connected to the second element in the same pair of vertices.

A multigraph is a *graph* when c is an injection [17].

Open systems demand a more general construction.

Definition 5. A *generalized oriented* or *directed multigraph* G has the following form:

$$G = \big(V, E, c : E \to \big(V \times V \cup V_b \cup V_e\big)\big).$$

Here V is the set of vertices or nodes of G, E is the set of edges of G (with fixed beginnings and ends), $V_b \approx V_e \approx V$, and c is the edge-node adjacency function, which assigns each edge either to a pair of vertices or to one vertex. In the latter case, when the image $c(e)$ of an edge e belongs to V_b, it means that e is connected to the vertex $c(e)$ by its beginning. When the image $c(e)$ of an edge e belongs to V_e, it means that e is connected to the vertex $c(e)$ by its end. Edges that are mapped to the set $V_b \cup V_e$ are called *open*.

The difference between multigraphs and generalized oriented multigraphs is that in a multigraph each edge connects two vertices, while in a generalized multigraph an edge may be connected only to one vertex.

A grid automaton is realized on a grid. Here is an exact definition of this grid.

Definition 6. The *grid* $\mathrm{G}(A)$ of a grid automaton A is the generalized oriented multigraph that has exactly the same vertices and edges as A, while its adjacency function $c_{\mathrm{G}(A)}$ is a composition of functions p_{IA} and c_A, namely, $c_{\mathrm{G}(A)}(l) = p^*_{IA}(c_A(l))$ where l is an arbitrary link from $\boldsymbol{C}_A$, $\boldsymbol{A}'_A$ and $\boldsymbol{A}''_A$ are disjoint copies of $\boldsymbol{A}_A$, and $p^*_{IA} = (p_{IA} \times p_{IA}) * p_{IA} * p_{IA} : \big(\boldsymbol{P}_{IA\mathrm{in}} \times \boldsymbol{P}_{IA\mathrm{out}}\big) \cup \boldsymbol{P}_{IA\mathrm{in}} \cup \boldsymbol{P}_{IA\mathrm{out}} \to (\boldsymbol{A}_A \times \boldsymbol{A}_A) \cup \boldsymbol{A}'_A \cup \boldsymbol{A}''_A$.

https://doi.org/10.25088/ComplexSystems.18.1.103

Here $\times$ is the product and $*$ is the coproduct of mappings in the sense of category theory [18].

Example 3. The grid G(**GA**) of the grid automaton **GA** from [2], Chapter 4 is given in Figure 1.

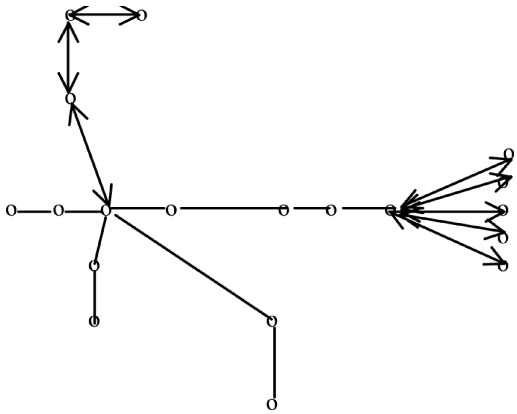

Figure 1. The grid of the grid automaton **GA** from [2], Chapter 4.

Grids of the grid automata allow one to characterize definite classes of grid automata.

Proposition 1. A grid automaton B is closed if and only if its grid G(B) satisfies the condition $\operatorname{Im} c \subseteq V \times V$, or in other words, the grid G(B) of B is a conventional multigraph.

Many classical models of computation for example, Turing machines, are closed grid automata.

Proposition 2. A grid automaton B is an acceptor only if it has external input ports or/and $\operatorname{Im} c \bigcap V_e \neq \emptyset$; that is, the grid G($B$) has edges connected by their end.

Proposition 3. A grid automaton B is a transmitter only if it has external output ports or/and $\operatorname{Im} c \bigcap V_b \neq \emptyset$; that is, the grid G($B$) has edges connected by their beginning.

Proposition 4. A grid automaton B is a transducer if and only if it has external input and output ports or/and $\operatorname{Im} c \bigcap V_b \neq \emptyset$ and $\operatorname{Im} c \bigcap V_e \neq \emptyset$; that is, the grid G($B$) has edges connected by their beginning and edges connected by their end.

Definition 7. The *connection grid* CG(A) of a grid automaton A is the generalized oriented multigraph nodes that bijectively correspond to internal ports of A, while edges and the adjacency function $c_{\mathrm{CG}(A)}$ are the same as in A.

https://doi.org/10.25088/ComplexSystems.18.1.103

Proposition 5. The grid G(B) of a grid automaton B is a homomorphic image of its connection grid CG(B).

Indeed, by the definition of a grid automaton, ports are uniquely assigned to nodes, and by the definition of the grid G(B) a grid automaton B, the adjacency function $c_{\mathrm{G}(B)}$ of the grid G(B) is a composition of the port assignment function p_B and the adjacency function c_B of the automaton B.

Grid automata as abstract information processing systems have different categories of resources: memory, interface (input and output) devices, control devices, operating devices, software, and data/knowledge bases.

Resource utilization modes yield this interdependence classification of automata in a grid.

1. *Autonomous automata* with independent resources.
2. *Automata with shared resources*, in which some resource, for example, memory, belongs to one node, but some other nodes from the grid can also use it.
3. *Automata with common resources*, for example, common memory or a database, which belong to two or more nodes from the grid.

Each type of automata implies specific styles of exchange in the grid. For example, here are three levels of exchange for autonomous automata.

1. *Data and program exchange* (distributed storage of information).
2. *Task and workspace exchange* (distributed computation and intelligent agent systems).
3. *System exchange* (data, knowledge, tasks, programs, and agents are specific systems in such an exchange).

4. Constructing with Cellular Automata

In this section, we develop techniques for cellular engineering.

Definition 8. It is possible to *model* an abstract automaton A by a cellular automaton C if there is a configuration W of cells from A and a system R of states of cells from W such that after initializing these states, the cellular automaton C works as the automaton A.

This is either process cellular engineering or function cellular engineering.

In some cases, individual cellular engineering allows us to perform cellular engineering for classes of automata.

Definition 9. It is possible to *model* a model of computation **M** in a class **C** of cellular automata if it is possible to model any automaton A from **M** by some cellular automaton C from **C**.

https://doi.org/10.25088/ComplexSystems.18.1.103

There are different types of modeling.

Definition 10. An abstract automaton A is called *programmable* in a cellular automaton C if there is a configuration W of cells from A and a system R of states of cells from W such that after initializing these states, the cellular automaton C works as the automaton A; that is, with the same input, C gives the same result as A.

This is function cellular engineering.

As in a general case, we can perform function cellular engineering for classes of automata.

Definition 11. A model of computation $\mathbf{M}$ is called *programmable* in a class $\mathbf{C}$ of cellular automata if any automaton A from $\mathbf{M}$ is programmable in some cellular automaton C from $\mathbf{C}$.

Let us consider a model of computation $\mathbf{M}$ that has a universal automaton U.

Theorem 1. A model of computation $\mathbf{M}$ is programmable in a class $\mathbf{C}$ of cellular automata if and only if a universal automaton U is programmable in some cellular automaton C from $\mathbf{M}$.

Corollary 1. A model of computation $\mathbf{M}$ is programmable in a cellular automaton C if the automaton U is programmable in C.

For illustration, we now give a well-known result in the theory of cellular automata.

Theorem 2. The class $\mathbf{T}$ of all Turing machines is programmable in the class $\mathbf{C}_1$ of one-dimensional cellular automata.

Definition 12. An abstract automaton A is called *constructible* in a cellular automaton C if there is a configuration W of cells from A and a system R of states of cells from W such that after initializing these states, the cellular automaton C works as the automaton A. And, to each structural component D of A, some part B of the automaton C is corresponded in such a way that B works as D.

This is system cellular engineering.

Definition 13. A model of computation $\mathbf{M}$ is called *constructible* in a class $\mathbf{C}$ of cellular automata if any automaton A from $\mathbf{M}$ is constructible in some cellular automaton C from $\mathbf{C}$.

To construct definite devices, we need elements from which to choose and algorithms to assemble them.

There are three main element types (in information typology), which correspond to the three main types of information operations described in [19].

- Computational elements or *transformers*.
- Transaction elements or *transmitters*.
- Storage elements or *memory cells*.

https://doi.org/10.25088/ComplexSystems.18.1.103

There are three element types (in dynamic typology), which correspond to their dynamics.

- Elements with a fixed structure.
- Reconfigurable elements.
- Switching elements.

Elements with a fixed structure have the same structure during the whole process. Reconfigurable elements can change their structure during the process. Switching elements tentatively change their structure in each operation.

There are three element types of memory cells: *read-only cells*, *write-only cells*, and *two-way cells*, which allow both reading and writing.

We use multilevel finite automata to realize all these types of elements in cellular automata.

The set Q_A of the states in an n-level finite automaton A is the direct product $Q_1 \times Q_2 \times Q_3 \times \cdots \times Q_n$ where Q_i is the set of states of the level i. All levels function in a parallel mode and their inputs and outputs have the same stratification, that is, a data element, or datum, with the form $(a_1,\ a_2,\ a_3,\ \ldots,\ a_n)$.

It is possible to use each level either for computation, information transmission, or as a memory. As a rule, we use the first level for computation; that is, Q_1 is a finite automaton that computes in the conventional way by changing its state and giving some (may be void) output. An intermediate range of levels (Q_2, ...) are used as a memory for the automaton Q_1. Each such level Q_j can store exactly $|Q_j|$ symbols or words when elements of Q_j are words. Upper levels are transmitters used for data transmission.

Each level i can also be stratified, that is, $Q_i = Q_{i1} \times Q_{i2} \times \cdots Q_{ik}$ where sets Q_{i2} are called layers of the level i. All transmitters Q_i in the automaton Q are stratified. One layer is used for data that are transmitted. Another layer is used for indicating the direction of transmission. One more layer is used for indicating when it is necessary to stop transmission. In addition, there is a layer that can be used for some changes of data on all other layers of this level.

For instance, it is necessary to transmit the symbol a five cells to the right and 10 cells up in rectangular grid of a two-dimensional cellular automaton. The symbol a is preserved in the first layer during the whole process of transmission. The word (5 r, 10 u) goes to the second layer. The third layer contains the word (0, 0). The automaton on the fourth layer sends data to prescribed neighbors and decreases the word on the second level, subtracting 1 after each move. When the move is to the right, 1 is subtracted from the first part of the word in the second layer. When the move is up, 1 is subtracted from the second part of the word in the second layer. When the word on the sec-

https://doi.org/10.25088/ComplexSystems.18.1.103

ond layer contains only zeros, the process of transmission stops, and the symbol a goes to a lower level of the automaton where the process stopped. The symbol can go to one of the levels in the memory or to the processor on the first level.

Building a two-dimensional cellular automaton CA from such multi-level finite automata, we can prove the following result.

Theorem 3. A two-dimensional cellular automaton can realize any finite grid of connections between nodes in a grid automaton G.

Corollary 2. If all nodes in a finite grid automaton G have a finite number of ports and are programmable (constructible) in one-dimensional cellular automata, then the automaton G is programmable (respectively, constructible) in a two-dimensional cellular automaton.

Note that not any finite configuration is a finite automaton. For instance, at each step, a Turing machine is a finite configuration, but it is not a finite automaton. Another example is when a node in a grid automaton can be an automaton that works with real numbers.

We now show how to construct a Turing machine in a two-dimensional cellular automaton.

Theorem 4. An arbitrary Turing machine is constructible in the class $\mathbf{C}_1$ of one-dimensional cellular automata.

Proof of this theorem is given in the Appendix.

Note that one-dimensional cellular automata that can emulate a one-dimensional Turing machine are not the standard result. The standard result says that an arbitrary Turing machine is programmable in the class $\mathbf{C}_1$ of one-dimensional cellular automata. Theorem 4 establishes that an arbitrary Turing machine is constructible in the class $\mathbf{C}_1$. Constructibility implies programmability, but the converse is not true. For instance, any Turing machine with a two-dimensional tape is programmable in the class of Turing machines with a one-dimensional tape, but it is not constructible in this class.

As the class $\mathbf{T}$ has universal Turing machines, Theorems 1 and 4 imply the following result.

Corollary 3. The class $\mathbf{T}$ of all Turing machines is constructible in the class $\mathbf{C}_1$ of one-dimensional cellular automata.

Global Turing machines or Internet machines, introduced by Van Leeuwen and Wiedermann in [20], form an important class of grid automata. An Internet machine is a finite grid automaton in which all nodes are Turing machines. The number of Turing machines and their connections may change in the process of functioning.

Theorems 3 and 4 imply the following result.

Corollary 4. An Internet machine IM is constructible in the class $\mathbf{C}_2$ of two-dimensional cellular automata.

This implies the following result.

https://doi.org/10.25088/ComplexSystems.18.1.103

Corollary 5. The class **IM** of all Internet machines is constructible in the class $\mathbf{C}_2$ of two-dimensional cellular automata.

Theorem 5. An arbitrary pushdown automaton is constructible in the class $\mathbf{C}_2$ of two-dimensional cellular automata.

The proof of this is similar to the proof of Theorem 4.

Corollary 6. The class **T** of all Turing machines is constructible in the class $\mathbf{C}_2$ of two-dimensional cellular automata.

In such a way, it is possible to program many other abstract automata (models of computation).

Appendix

A. Proof of Theorem 4

As all kinds of Turing machines are functionally equivalent to deterministic Turing machines with one head and one linear tape (cf. [2]), we can prove this result only for a deterministic Turing machine with one head and one linear tape. In addition, we can assume that the tape is one-directional.

Let us consider a Turing machine T. It has an alphabet X, a control device A that is a finite automaton and controls the performance of T, a head or operating device H, and a linear tape E used as a memory of T. The tape E consists of identical cells, which are enumerated by natural numbers. The head H can write a symbol from X to any cell or erase such a symbol from a cell. At the beginning, a finite number of cells may be filled with symbols from X. The functioning of T is determined by a system R of rules, which have the form

$$\langle q_h, a_i \rangle \rightarrow \langle a_j, q_k, e \rangle. \tag{A.1}$$

Here q_h and q_k are states of the automaton A, a_i and a_j are symbols of the alphabet of T, and c is either R, L, or N. Rule (A.1) means that if the state of the control device A of T is q_h and the head H observes the symbol a_i in the cell, then the state of A becomes q_k, the head H writes the symbol a_j in the cell where it is situated and moves to the next cell by a connection of the type c. That is, if $e = \mathrm{R}$, the head moves to the nearest right cell in E. When $e = \mathrm{L}$, the head moves to the nearest left cell in E. When $e = \mathrm{N}$, the head does not move. Each rule directs one computation step of the Turing machine T.

To model the machine T by a one-dimensional cellular automaton CA, we use a three-level deterministic finite automaton AT as the cell of the automaton CA. On the first level of AT, we have an identical copy of the automaton A from the Turing machine T. On the second level of AT, we have an automaton C that models the functioning of an arbitrary cell from the memory E of the Turing machine T. If X is the alphabet of the Turing machine T, then for each symbol from X,

https://doi.org/10.25088/ComplexSystems.18.1.103

the automaton C has a corresponding state or even several states. This allows CA to store data on this level. We denote the second level of the automaton AT by cAT. On the third level of the automaton AT, we have an automaton D that is used only for data transmission.

Each of the levels has a dead-end state d_j, $j = 1, 2, 3$; when the level comes to this state, it stops functioning. Other levels may continue to work, but this level halts. In addition, the second and third levels have closed states. When the memory cell level is in an open state, it is possible to write to this cell (i.e., to change its content) and to read from this cell (i.e., to take the symbol written in the cell to the transmission level of the same automaton). When the memory cell level is in a closed state, it is forbidden (impossible) to write into this cell, that is, to change its content, and it is also forbidden (impossible) to read from this cell. Each time the content is changed in a cell, the memory cell level comes to a closed state. Thus, the states of the second level have the form (a, c) and (a, f) where $a \in X \cup \{\Lambda\}$, Λ denotes the empty cell, c denotes the closed state of the cell, and f denotes the open state of the cell.

When the transmission level is in an open state, data can be transmitted in the prescribed direction; that is, this level gives output in this direction. When the memory cell level is in a closed state, it is forbidden (impossible) to transmit data; that is, this level does not give output in all directions.

Thus, one-dimensional cellular automaton CA is a sequence of deterministic finite automata $\{AT_0, AT_1, AT_2, \ldots, AT_n, \ldots\}$, each of which is an identical copy of the automaton AT. It is possible to extend this sequence infinitely to the left, using integer numbers for enumeration, but we do not need this.

When starting the functioning of the cellular automaton CA, we have to reproduce the initial state of the Turing machine T, which starts working with an input word w in the alphabet X. To do this, we put the memory cell level of the automaton AT_0 in the state d_2. As a result, this level never functions in AT_0. By the rules of Turing machine functioning (cf. [2] or [21]), the input w is written in cells of the tape of the Turing machine T. Usually cells with numbers $1, 2, 3, \ldots, k$ are used where k is the length of the word w. Correspondingly, if $w = a_1 a_2 a_3 \ldots a_k$, then we put the second level of the automata $AT_1, AT_2, AT_3, \ldots, AT_k$ in the states (a_1, f), (a_2, c), (a_3, c), $\ldots$, (a_k, c). We keep the memory level of the automaton AT_i open when the head of T is at the cell with number i. The head of a Turing machine is usually at the first cell when functioning begins (cf. [2] or [21]). So, we keep the memory level of the automaton AT_1 open.

Note that the cellular automaton CA models the structure of the Turing machine T. Indeed, it has the control device AT_0 and the sequence of cells $\{\text{c}AT_1, \text{c}AT_2, \ldots, \text{c}AT_n, \ldots\}$. This sequence of cells

https://doi.org/10.25088/ComplexSystems.18.1.103

corresponds to the one-directional linear tape of the Turing machine T. Besides, the head H of the Turing machine T is modeled by the symbol f, which denotes the open cell; that is, the cell of T observed by the head H corresponds to the open cell of CA.

We prove that the cellular automaton CA imitates the functioning of Turing machine T for the input word w by induction. To do this, we, at first, model the first step of the Turing machine T.

The head H reads the symbol a_1 written in the first cell of the tape. In a similar way, the cellular automaton AT_1 reads the symbol a_1 written in the first cell cAT_1 of the cellular automaton CA by the following rule of the automaton AT_1:

$$\{\langle(d_1),\ \epsilon\rangle,\ \langle(a_1, f),\ \epsilon\rangle,\ \langle(\Lambda,\ \mathrm{r}c,\ \mathrm{l}c),\ \epsilon\rangle\} \rightarrow \\ \left\{\langle(d_1),\ \epsilon\rangle,\ \langle(a_1,\ f),\ \epsilon\rangle,\ \left\langle\left(a_1,\ \mathrm{r}c,\ l_f\right),\ a_1\right\rangle\right\}.$$

Here ϵ denotes the empty input. Λ denotes the empty symbol. $\langle(d_1),\ \epsilon\rangle$ means that the first level of AT_1 is switched off, that is, gives no output and has no input. $\langle(a_1,\ f),\ \epsilon\rangle$ means that the second level of AT_1 is in the open state a_1, that is, gives no output and has no input. $\langle(\Lambda,\ \mathrm{r}c,\ \mathrm{l}c),\ \epsilon\rangle$ means that the third level of AT_1 contains no symbols from X, is in the closed state to the right and left, gives no output to the right or left, and has no input from the right or left. $\langle(a_1,\ \mathrm{r}c,\ \mathrm{l}f),\ a_1\rangle$ means that the third level of AT_1 contains the symbol a_1 from X, is in the closed state to the right, is in the open state to the left, gives no output to the right, gives a_1 as its output to the left, and has no input from the right or left.

Informally, this rule determines an operation that rewrites the symbol a_1 from the memory to the transmission cell, which gives this symbol a_1 as its output to the left. Consequently, this symbol a_1 comes as input to AT_0 from the right. To process it, the automaton AT_0 accepts this symbol a_1 to its third, transmission level by the rule

$$\left\{\left\langle\left(q_0\right),\ \epsilon\right\rangle,\ \langle(d_2,\ c),\ \epsilon\rangle,\ \langle(\Lambda,\ \mathrm{r}f,\ \mathrm{l}c),\ a_1\rangle\right\} \rightarrow \\ \left\{\left\langle\left(q_0\right),\ \epsilon\right\rangle,\ \langle(d_2,\ c),\ \epsilon\rangle,\ \langle(a_1,\ \mathrm{r}c,\ \mathrm{l}f),\ \epsilon\rangle\right\}.$$

Here $\left\langle\left(q_0\right),\ \epsilon\right\rangle$ means that the first level of AT_0 is in the state q_0, which is the start state of the Turing machine T, so it gives no output and has no input. $\langle(d_2,\ c),\ \epsilon\rangle$ means that the second level of AT_0 is in the dead-end state d_2, so it gives no output and accepts no input. $\langle(\Lambda,\ \mathrm{r}f,\ \mathrm{l}c),\ a_1\rangle$ means that the third level of AT_0 contains no symbols from X, is in the closed state to the left, is in the open state to the right, gives no output to the right or left and has the symbol a_1 as input from the right. $\langle(a_1,\ \mathrm{r}c,\ \mathrm{l}f),\ \epsilon\rangle$ means that the third level of AT_0 contains the symbol a_1 from X, is in the closed state to the right, is in the open state to the left, gives no output to the right, gives a_1 as its output to the left, and has no input from the right or left.

https://doi.org/10.25088/ComplexSystems.18.1.103

At the same time, the automaton AT_1 performs the rule

$$\{\langle(d_1), \varepsilon\rangle,\ \langle(a_1, f), \varepsilon\rangle,\ \langle(a_1, \mathrm{r}c, \mathrm{l}f), \varepsilon\rangle\} \to \\ \{\langle(d_1), \varepsilon\rangle,\ \langle(a_1, f), \varepsilon\rangle,\ \langle(\Lambda, \mathrm{r}c, \mathrm{l}f), \varepsilon\rangle\}.$$

Then the automaton AT_0 lowers the symbol a_1 to its first level by the rule

$$\{\langle(q_0), \varepsilon\rangle,\ \langle(d_2, c), \varepsilon\rangle,\ \langle(a_1, \mathrm{r}f, \mathrm{l}c), \varepsilon\rangle\} \to \\ \{\langle(q_0), a_1\rangle,\ \langle(d_2, c), \varepsilon\rangle,\ \langle(\Lambda, \mathrm{r}c, \mathrm{l}c), \varepsilon\rangle\}.$$

Here $\langle(q_0), a_1\rangle$ means that the first level of AT_0 is in the state q_0, which is the start state of the Turing machine T, so it gives no output and has the symbol a_1 as input. $\langle(d_2, c), \varepsilon\rangle$ means that the second level of AT_0 is in the dead-end state d_2, so it gives no output and accepts no input. $\langle(a_1, \mathrm{r}f, \mathrm{l}c), \varepsilon\rangle$ means that the third level of AT_0 contains symbol a_1 from X, is in the open state to the right, gives no output to the right or left, and has no input from the right or left.

The automaton AT_0 works as the control device A of the Turing machine T. Thus, AT_0 looks through the system R of rules of the Turing machine T. If there is no rule that has the form

$$\langle q_0, a_1\rangle \to \langle a_j, q_t, e\rangle \tag{A.2}$$

for some numbers j and t, then the automaton AT_0 and thus, the cellular automaton CA, stop because this state of AT_0 has no continuation, and all other automata AT_1, AT_2, AT_3, ..., AT_k, ... from CA are closed. At the same time, the absence of rule (A.2) in R means that the automaton A and thus, the Turing machine T, also stop due to the impossibility of making another move. Consequently, in this case, the cellular automaton CA behaves exactly as the Turing machine T.

When the system R has rule (A.2), for some numbers j and t, then the automaton AT_0 makes the following transition:

$$\{\langle(q_0), a_1\rangle,\ \langle(d_2, c), \varepsilon\rangle,\ \langle(\Lambda, \mathrm{r}c, \mathrm{l}c), \varepsilon\rangle\} \to \\ \{\langle(q_t), a_j\rangle,\ \langle(d_2, c), \varepsilon\rangle,\ \langle(a_j, \mathrm{r}c, \mathrm{l}c), \varepsilon\rangle\}.$$

The state of AT_0 changes to q_t and it gives the output (a_j, e). Then the automaton AT_0 elevates the pair (a_j, e) to its third level by the rule

$$\{\langle(q_t), a_j\rangle,\ \langle(d_2, c), \varepsilon\rangle,\ \langle(a_j, \mathrm{r}c, \mathrm{l}c), \varepsilon\rangle\} \to \\ \{\langle(q_t), \varepsilon\rangle,\ \langle(d_2, c), \varepsilon\rangle,\ \langle((a_j, e), \mathrm{r}c, \mathrm{l}c), \varepsilon\rangle\}.$$

https://doi.org/10.25088/ComplexSystems.18.1.103

After this move, the automaton AT_0 transmits the pair (a_j, e) to the third level of the second cell AT_1 in CA by these rules:

$$\{\langle(q_t), \varepsilon\rangle, \langle(d_2, c), \varepsilon\rangle, \langle((a_j, e), \mathrm{r}c, \mathrm{l}c), \varepsilon\rangle\} \to \{\langle(q_t), \varepsilon\rangle, \langle(d_2, c), \varepsilon\rangle, \langle(\Lambda, \mathrm{r}f, \mathrm{l}c), (a_j, e)\rangle\} \text{ and}$$

$$\{\langle(d_1), \varepsilon\rangle, \langle(a_1, f), \varepsilon\rangle, \langle(\Lambda, \mathrm{r}c, \mathrm{l}f), (a_j, e)\rangle\} \to \{\langle(d_1), \varepsilon\rangle, \langle(a_1, f), \varepsilon\rangle, \langle((a_j, e), \mathrm{r}c, \mathrm{l}f), \varepsilon\rangle\}.$$

Here the first rule determines operation in AT_0, while the second rule determines operation in AT_1. Then the symbol is written to the cell $\mathrm{c}AT_1$ by one of the following rules.

- When $e = \mathrm{N}$, we apply the rule

$$\{\langle(d_1), \varepsilon\rangle, \langle(a_1, f), \varepsilon\rangle, \langle((a_j, e), \mathrm{r}c, \mathrm{l}f), \varepsilon\rangle\} \to \{\langle(d_1), \varepsilon\rangle, \langle(a_j, f), \varepsilon\rangle, \langle(\Lambda, \mathrm{r}c, \mathrm{l}f), \varepsilon\rangle\}.$$

- When $e = \mathrm{R}$, we apply the rule

$$\{\langle(d_1), \varepsilon\rangle, \langle(a_1, f), \varepsilon\rangle, \langle((a_j, e), \mathrm{r}c, \mathrm{l}f), \varepsilon\rangle\} \to \{\langle(d_1), \varepsilon\rangle, \langle(a_j, c), \varepsilon\rangle, \langle(\Lambda, \mathrm{r}f, \mathrm{l}c), \mathrm{R}\rangle\}.$$

- When $e = \mathrm{L}$, we apply the rule

$$\{\langle(d_1), \varepsilon\rangle, \langle(a_1, f), \varepsilon\rangle, \langle((a_j, e), \mathrm{r}c, \mathrm{l}f), \varepsilon\rangle\} \to \{\langle(d_1), \varepsilon\rangle, \langle(a_j, c), \varepsilon\rangle, \langle(\Lambda, \mathrm{r}c, \mathrm{l}f), \mathrm{L}\rangle\}.$$

Here $\langle(\Lambda, \mathrm{r}f, \mathrm{l}c), \mathrm{R}\rangle$ means that the third level of AT_1 contains no symbols from X, is in the closed state to the left, is in the open state to the right, gives output R to the right, and no output to the left. $\langle(\Lambda, \mathrm{r}c, \mathrm{l}f), \mathrm{L}\rangle$ means that the third level of AT_1 contains no symbols from X, is in the closed state to the right, is in the open state to the left, gives no output to the right, gives L as its output to the left, and has no input from the right or left.

The first case means that the same cell of CA stays open. The corresponding rule of the Turing machine T means that the head of T does not move from the initial cell. Consequently, in this case, the cellular automaton CA behaves exactly as the Turing machine T.

The second case means that the cell $\mathrm{c}AT_2$ of CA becomes open by the rule

$$\{\langle(d_1), \varepsilon\rangle, \langle(a_v, c), \varepsilon\rangle, \langle(\Lambda, \mathrm{r}c, \mathrm{l}c), \mathrm{R}\rangle\} \to \{\langle(d_1), \varepsilon\rangle, \langle(a_v, f), \varepsilon\rangle, \langle(\Lambda, \mathrm{r}c, \mathrm{l}c), \varepsilon\rangle\},$$

or, when this cell is empty, by the rule

$$\{\langle\emptyset, \varepsilon\rangle, \langle\emptyset, \varepsilon\rangle, \langle\emptyset, \mathrm{R}\rangle\} \to \{\langle(d_1), \varepsilon\rangle, \langle(\Lambda, f), \varepsilon\rangle, \langle(\Lambda, \mathrm{r}c, \mathrm{l}c), \varepsilon\rangle\}.$$

https://doi.org/10.25088/ComplexSystems.18.1.103

Rule (A.2), in this case, means that the Turing machine T moves the head to the right of the first memory cell. Consequently, in this case, the cellular automaton CA behaves exactly as the Turing machine T.

The third case makes the cellular automaton CA stop because there are no memory cells to the left of $\mathrm{c}AT_1$. This also means that the machine T stops because its rule demands moving the head to the left of the first memory cell.

Thus, we have demonstrated that the automaton CA exactly simulates the first move of the Turing machine T.

Now we describe how the cellular automaton CA imitates one step of the Turing machine T. Let us assume that the machine T is in the state q_j and the head H observes a cell with number i in which the symbol a_l is written, meaning that the control device of T is in the state q_j. By the induction assumption, the automaton AT_0 is in the state

$$\{\langle(q_j), \varepsilon\rangle,\ \langle(d_2, c), \varepsilon\rangle,\ \langle(\Lambda, \mathrm{r}c, \mathrm{l}c), \varepsilon\rangle\},$$

and the automaton AT_i is in the state

$$\{\langle d_1, \varepsilon\rangle,\ \langle(a_l, f), \varepsilon\rangle,\ \langle(\Lambda, \mathrm{r}c, \mathrm{l}c), \varepsilon\rangle\}.$$

Then the head H reads the symbol a_l written in the cell. In a similar way, the cellular automaton AT_i reads the symbol a_l written in the cell $\mathrm{c}AT_i$ of the cellular automaton CA by this rule of the automaton AT_i:

$$\{\langle(d_1), \varepsilon\rangle,\ \langle(a_l, f), \varepsilon\rangle,\ \langle(\Lambda, \mathrm{r}c, \mathrm{l}c), \varepsilon\rangle\} \to \\ \{\langle(d_1), \varepsilon\rangle,\ \langle(a_l, f), \varepsilon\rangle,\ \langle(a_l, \mathrm{r}c, \mathrm{l}f), a_l\rangle\}.$$

Here ε denotes the empty input. Λ denotes the empty symbol. $\langle(d_1), \varepsilon\rangle$ means that the first level of AT_i is switched off, so it gives no output and has no input. $\langle(a_l, f), \varepsilon\rangle$ means that the second level of AT_i is in the open state a_1, so it gives no output and has no input. $\langle(\Lambda, \mathrm{r}c, \mathrm{l}c), \varepsilon\rangle$ means that the third level of AT_i contains no symbols from X, is in the closed state to the right and left, gives no output to the right or left, and has no input from the right or left. $\langle(a_l, \mathrm{r}c, \mathrm{l}f), a_l\rangle$ means that the third level of AT_i contains the symbol a_l from X, is in the closed state to the right, is in the open state to the left, gives no output to the right, gives a_l as its output to the left, and has no input from the right or left.

https://doi.org/10.25088/ComplexSystems.18.1.103

Informally, this rule determines an operation that rewrites the symbol a_l from the memory of AT_i to the transmission cell of AT_i, which gives this symbol a_l as its output to left. Consequently, this symbol a_l comes as input to the automaton AT_{i-1} from the right. The automaton AT_{i-1} processes this symbol by the rule

$$\{\langle(d_1), \varepsilon\rangle,\ \langle(a_l, c), \varepsilon\rangle,\ \langle(\Lambda, \mathrm{r}c, \mathrm{l}c), a_l\rangle\} \to \\ \{\langle(d_1), \varepsilon\rangle,\ \langle(a_l, f), \varepsilon\rangle,\ \langle(a_l, \mathrm{r}c, \mathrm{l}f), a_l\rangle\}.$$

This rule means that the symbol a_l comes as input from the right to the third level of this automaton. Then the symbol is written to this third level and goes as its output to the left, coming to the automaton AT_{i-2}.

At the same time, the automaton AT_i cleans it transmission level by the rule

$$\{\langle(d_1), \varepsilon\rangle,\ \langle(a_l, f), \varepsilon\rangle,\ \langle(a_l, \mathrm{r}c, \mathrm{l}c), \varepsilon\rangle\} \to \\ \{\langle(d_1), \varepsilon\rangle,\ \langle(a_l, f), \varepsilon\rangle,\ \langle(\Lambda, \mathrm{r}c, \mathrm{l}c), \varepsilon\rangle\}.$$

After i steps, this symbol a_l comes as input to the automaton AT_0 from the right. To process it, the automaton AT_0 accepts this symbol a_l to its third transmission level by the rule

$$\{\langle(q_0), \varepsilon\rangle,\ \langle(d_2, c), \varepsilon\rangle,\ \langle(\Lambda, \mathrm{r}f, \mathrm{l}c), a_l\rangle\} \to \\ \{\langle(q_0), \varepsilon\rangle,\ \langle(d_2, c), \varepsilon\rangle,\ \langle(a_l, \mathrm{r}c, \mathrm{l}c), \varepsilon\rangle\}.$$

Here $\langle(q_0), \varepsilon\rangle$ means that the first level of AT_0 is in the state q_0, which is the start state of the Turing machine T, so it gives no output and has no input. $\langle(d_2, c), \varepsilon\rangle$ means that the second level of AT_0 is in the dead-end state d_2, so it gives no output and accepts no input. $\langle(\Lambda, \mathrm{r}c, \mathrm{l}c), a_l\rangle$ means that the third level of AT_0 contains no symbols from X, is in the closed state to the left, is in the open state to the right, gives no output to the right or left, and has the symbol a_l as input from the right. $\langle(a_l, \mathrm{r}c, \mathrm{l}c), \varepsilon\rangle$ means that the third level of AT_0 contains the symbol a_l from X, is in the closed state to the right, is in the open state to the left, gives no output to the right, gives a_l as its output to the left, and has no input from the right or left.

At the same time, the automaton AT_1 performs the rule

$$\{\langle(d_1), \varepsilon\rangle,\ \langle(a_1, f), \varepsilon\rangle,\ \langle(a_l, \mathrm{r}c, \mathrm{l}f), \varepsilon\rangle\} \to \\ \{\langle(d_1), \varepsilon\rangle,\ \langle(a_l, f), \varepsilon\rangle,\ \langle(\Lambda, \mathrm{r}c, \mathrm{l}f), \varepsilon\rangle\}.$$

https://doi.org/10.25088/ComplexSystems.18.1.103

Then the automaton AT_0 lowers the symbol a_l to its first level by the rule

$$\{\langle(q_0), \varepsilon\rangle, \langle(d_2, c), \varepsilon\rangle, \langle(a_l, \mathrm{r}f, \mathrm{l}c), \varepsilon\rangle\} \to \{\langle(q_0), a_l\rangle, \langle(d_2, c), \varepsilon\rangle, \langle(\Lambda, \mathrm{r}c, \mathrm{l}c), \varepsilon\rangle\}.$$

Here $\langle(q_0), a_l\rangle$ means that the first level of AT_0 is in the state q_0, which is the start state of the Turing machine T, so it gives no output and has the symbol a_l as input. $\langle(d_2, c), \varepsilon\rangle$ means that the second level of AT_0 is in the dead-end state d_2, so it gives no output and accepts no input. $\langle(a_l, \mathrm{r}f, \mathrm{l}c), \varepsilon\rangle$ means that the third level of AT_0 contains symbol a_1 from X, is in the open state to the right, gives no output to the right or left, and has no input from the right and from the left.

The automaton AT_0 works as the control device A of the Turing machine T. Thus, AT_0 looks through the system R of rules of the Turing machine T. If there is no rule that has the form

$$\langle q_0, a_l\rangle \to \langle a_h, q_t, e\rangle \tag{A.3}$$

for some numbers j and t, then the automaton AT_0 and, thus, the cellular automaton CA, stop because this state of AT_0 has no continuation and all other automata AT_1, AT_2, AT_3, ..., AT_k, ... from CA are closed. At the same time, the absence of rule (A.2) in R means that the automaton A and, thus, the Turing machine T, also stop due to the impossibility of making another move. Consequently, in this case, the cellular automaton CA behaves exactly as the Turing machine T.

When the system R has rule (A.3) for some numbers j and t, then the automaton AT_0 makes the following transition:

$$\{\langle(q_0), a_l\rangle, \langle(d_2, c), \varepsilon\rangle, \langle(\Lambda, \mathrm{r}c, \mathrm{l}c), \varepsilon\rangle\} \to \{\langle(q_t), a_h\rangle, \langle(d_2, c), \varepsilon\rangle, \langle(a_h, \mathrm{r}c, \mathrm{l}c), \varepsilon\rangle\}.$$

The state of AT_0 changes to q_t, and it gives the output (a_j, e). Then the automaton AT_0 elevates the pair (a_h, e) to its third level by the rule

$$\{\langle(q_t), a_h\rangle, \langle(d_2, c), \varepsilon\rangle, \langle(a_h, \mathrm{r}c, \mathrm{l}c), \varepsilon\rangle\} \to \{\langle(q_t), \varepsilon\rangle, \langle(d_2, c), \varepsilon\rangle, \langle((a_h, e), \mathrm{r}c, \mathrm{l}c), \varepsilon\rangle\}.$$

After this move, the automaton AT_0 transmits the pair (a_h, e) to the third level of the second cell AT_1 in CA by these rules:

$$\{\langle(q_t), \varepsilon\rangle, \langle(d_2, c), \varepsilon\rangle, \langle((a_h, e), \mathrm{r}c, \mathrm{l}c), \varepsilon\rangle\} \to \{\langle(q_t), \varepsilon\rangle, \langle(d_2, c), \varepsilon\rangle, \langle(\Lambda, \mathrm{r}f, \mathrm{l}c), (a_h, e)\rangle\} \text{ and}$$

$$\{\langle(d_1), \varepsilon\rangle, \langle(a_1, c), \varepsilon\rangle, \langle(\Lambda, \mathrm{r}c, \mathrm{l}f), (a_h, e)\rangle\} \to \{\langle(d_1), \varepsilon\rangle, \langle(a_1, c), \varepsilon\rangle, \langle((a_h, e), \mathrm{r}c, \mathrm{l}f), \varepsilon\rangle\}.$$

https://doi.org/10.25088/ComplexSystems.18.1.103

Here the first rule determines operation in AT_0, while the second rule determines operation in AT_1.

Then the automaton AT_1 transmits the pair (a_b, e) to the third level of the second cell AT_2 by similar rules, and this process continues until the pair (a_b, e) reaches the third level of the automaton AT_k, in which the second level is open. However, the only automaton AT_k, in which the second level is open, is the automaton AT_i. Thus, the new symbol a_b reaches the third level of the automaton AT_i.

Then the symbol is written to the cell $\mathrm{c}AT_i$ by one of the following rules.

- When $e = \mathrm{N}$, we apply the rule

$$\{\langle(d_1), \varepsilon\rangle, \langle(a_l, f), \varepsilon\rangle, \langle((a_b, e), \mathrm{r}c, \mathrm{l}f), \varepsilon\rangle\} \to \{\langle(d_1), \varepsilon\rangle, \langle(a_b, f), \varepsilon\rangle, \langle(\Lambda, \mathrm{r}c, \mathrm{l}f), \varepsilon\rangle\}.$$

- When $e = \mathrm{R}$, we apply the rule

$$\{\langle(d_1), \varepsilon\rangle, \langle(a_l, f), \varepsilon\rangle, \langle((a_b, e), \mathrm{r}c, \mathrm{l}f), \varepsilon\rangle\} \to \{\langle(d_1), \varepsilon\rangle, \langle(a_b, c), \varepsilon\rangle, \langle(\Lambda, \mathrm{r}f, \mathrm{l}c), \mathrm{R}\rangle\}.$$

- When $e = \mathrm{L}$, we apply the rule

$$\{\langle(d_1), \varepsilon\rangle, \langle(a_l, f), \varepsilon\rangle, \langle((a_b, e), \mathrm{r}c, \mathrm{l}f), \varepsilon\rangle\} \to \{\langle(d_1), \varepsilon\rangle, \langle(a_b, c), \varepsilon\rangle, \langle(\Lambda, \mathrm{r}c, \mathrm{l}f), \mathrm{L}\rangle\}.$$

Here $\langle(\Lambda, \mathrm{r}f, \mathrm{l}c), \mathrm{R}\rangle$ means that the third level of AT_i contains no symbols from X, is in the closed state to the left, is in the open state to the right, gives output R to the right, and no output to the left. $\langle(\Lambda, \mathrm{r}c, \mathrm{l}f), \mathrm{L}\rangle$ means that the third level of AT_i contains no symbols from X, is in the closed state to the right, is in the open state to the left, gives no output to the right, gives L as its output to the left, and has no input from the right or left.

The first case means that the same cell of CA stays open. The corresponding rule of the Turing machine T means that the head of T does not move from the initial cell. Consequently, in this case, the cellular automaton CA behaves exactly as the Turing machine T.

The second case means that the cell $\mathrm{c}AT_{i+1}$ of CA becomes open by the rule

$$\{\langle(d_1), \varepsilon\rangle, \langle(a_v, c), \varepsilon\rangle, \langle(\Lambda, \mathrm{r}c, \mathrm{l}c), \mathrm{R}\rangle\} \to \{\langle(d_1), \varepsilon\rangle, \langle(a_v, f), \varepsilon\rangle, \langle(\Lambda, \mathrm{r}c, \mathrm{l}c), \varepsilon\rangle\}$$

or when this cell is empty, by the rule

$$\{\langle\emptyset, \varepsilon\rangle, \langle\emptyset, \varepsilon\rangle, \langle\emptyset, \mathrm{R}\rangle\} \to \{\langle(d_1), \varepsilon\rangle, \langle(\Lambda, f), \varepsilon\rangle, \langle(\Lambda, \mathrm{r}c, \mathrm{l}c), \varepsilon\rangle\}.$$

Rule (A.2), in this case, means that the Turing machine T moves the head to the right of the memory cell with number i. Consequently, in

https://doi.org/10.25088/ComplexSystems.18.1.103

this case, the cellular automaton CA behaves exactly as the Turing machine T.

When $i > 1$, then the third case means that the cell $\mathrm{c}AT_{i-1}$ of CA becomes open by the rule

$$\{\langle(d_1), \varepsilon\rangle,\ \langle(a_v, c), \varepsilon\rangle,\ \langle(\Lambda, \mathrm{r}c, \mathrm{l}c), \mathrm{R}\rangle\} \to \{\langle(d_1), \varepsilon\rangle,\ \langle(a_v, f), \varepsilon\rangle,\ \langle(\Lambda, \mathrm{r}c, \mathrm{l}c), \varepsilon\rangle\},$$

or when this cell is empty, by the rule

$$\{\langle\emptyset, \varepsilon\rangle,\ \langle\emptyset, \varepsilon\rangle,\ \langle\emptyset, \mathrm{R}\rangle\} \to \{\langle(d_1), \varepsilon\rangle,\ \langle(\Lambda, f), \varepsilon\rangle,\ \langle(\Lambda, \mathrm{r}c, \mathrm{l}c), \varepsilon\rangle\}.$$

Rule (A.2), in this case, means that the Turing machine T moves the head to the left of the memory cell with number i. Consequently, in this case, the cellular automaton CA behaves exactly as the Turing machine T.

When $i = 1$, then the third case makes the cellular automaton CA stop because there are no memory cells to the left of $\mathrm{c}AT_1$. This also means that the Turing machine T stops because its rule demands moving the head to the left of the first memory cell.

Thus, we have demonstrated that the cellular automaton CA exactly simulates an arbitrary move of the Turing machine T. Now we can apply the induction principle, which asserts that the cellular automaton CA exactly simulates any number of moves of the Turing machine T.

The theorem is proved.

Acknowledgments

The author is grateful to the anonymous reviewer for useful advice.

References

[1] M. Burgin, "Cluster Computers and Grid Automata," in *Proceedings of the Eighteenth ISCA International Conference on Computers and Their Applications (CATA'03)*, Honolulu (N. Debnath, ed.), Cary, NC: International Society for Computers and Their Applications, 2003 pp. 106-109.

[2] M. Burgin, *Super-recursive Algorithms*, New York: Springer-Verlag, 2005.

[3] L. L. Peterson and B. S. Davie, *Computer Networks: A System Approach*, San Francisco: Morgan Kaufmann Publishers, 2000.

[4] S. Wolfram, *A New Kind Of Science*, Champaign, IL: Wolfram Media, Inc., 2002.

[5] J. von Neumann, *John von Neumann Collected Works*, New York: Macmillan, 1963.

https://doi.org/10.25088/ComplexSystems.18.1.103

[6] E. D. Adamides, P. Tsalides, and A. Thanailakis, "Hierarchical Cellular Automata Structures," *Parallel Computing*, **8**(5), 1992 pp. 517-524.

[7] B. K. Sikdar, P. Majumder, M. Mukherjee, P. P. Chaudhuri, D. K. Das, and N. Ganguly, "Hierarchical Cellular Automata As an On-Chip Test Pattern Generator," in *Proceedings of the Fourteenth International Conference on VLSI Design (VLSID'01)*, Washington, DC: IEEE Computer Society, 2001 pp. 403-408. doi.ieeecomputersociety.org/10.1109/ICVD.2001.902692.

[8] W. S. McCulloch and W. Pitts, "A Logical Calculus of the Ideas Immanent in Nervous Activity," *Bulletin of Mathematical Biology*, **5**(4), 1943 pp. 115-133.

[9] M. L. Minsky, *Computation: Finite and Infinite Machines (Automatic Computation)*, New York: Prentice-Hall, 1967.

[10] M. Steenstrup, M. A. Arbib, and E. G. Manes, "Port Automata and the Algebra of Concurrent Processes," *Journal of Computer and System Sciences*, **27**(1), 1983 pp. 29-50.

[11] J. Wiedermann, "Characterizing the Super-Turing Computing Power and Efficiency of Classical Fuzzy Turing Machines," *Theoretical Computer Science*, **317**(1-3), 2004 pp. 61-69.

[12] M. Burgin, "Reflexive Turing Machines and Calculi," *Vychislitelnyye Systemy (Logical Methods in Computer Science)*, No. 148, 1993 pp. 94-116, 175-176 (in Russian).

[13] P. Wegner and D. Goldin, "Computation beyond Turing Machines," *Communications of the ACM*, **46**(4), 2003 pp. 100-102.

[14] V. P. Heuring and H. F. Jordan, *Computer Systems Design and Architecture*, Menlo Park, CA: Addison-Wesley Publishing Company, 1997.

[15] F. J. Yndurain, *Quantum Chromodynamics: An Introduction to the Theory of Quarks and Gluons (Texts and Monographs in Physics)*, New York: Springer-Verlag, 1983.

[16] M. Margenstern, "Cellular Automata in the Hyperbolic Plane: A Survey," *Romanian Journal of Information Science and Technology*, **5**(1-2), 2002 pp. 155-179.

[17] C. Berge, *Graphs and Hypergraphs*, New York: North Holland P. C., 1973.

[18] H. Herrlich and G. E. Strecker, *Category Theory: An Introduction (Allyn and Bacon Series in Advanced Mathematics)*, Boston: Allyn and Bacon, 1973.

[19] M. Burgin, "Information Algebras," *Control Systems and Machines*, No. 6, 1997 pp. 5-16 (in Russian).

[20] J. van Leeuwen and J. Wiedermann, "Breaking the Turing Barrier: The Case of the Internet," Technical Report, Prague: Institute of Computer Science, Academy of Sciences of the Czech Republic, 2000.

[21] H. Rogers, *Theory of Recursive Functions and Effective Computability*, Cambridge, MA: MIT Press, 1987.

https://doi.org/10.25088/ComplexSystems.18.1.103