Theory and Applications of Mathematics & Computer Science

(ISSN 2067-2764, EISSN 2247-6202)
http://www.uav.ro/applications/se/journal/index.php/tamcs

Theory and Applications of Mathematics & Computer Science 7 (2) (2017) 1 – 30

Structural Machines as a Mathematical Model of Biological and Chemical Computers

Mark Burgin[a,*], Andrew Adamatzky[b,c]

[a] *University of California at Los Angeles, Los Angeles, CA 90095, USA.*
[b] *University of the West of England, Bristol BS16 1QY, UK.*
[c] *Unconventional Computing Ltd, Bristol BS39 5RX, UK.*

Abstract

To rigorously describe and study the living morphological computers, we develop a formal model of algorithms and computations called a structural machine providing theoretical tools for exploration of possibilities of biological computations and extension of their applications. We study properties of structural machines demonstrating how they can model the most popular in computer science model of computation — Turing machine. We also prove that structural machines can solve some problems more efficiently than Turing machines. We also show how structural machines model a programmable amorphous biological computer called a Physarum machine, as well as an inductive Turing machine.

Keywords: model of computation, algorithm, structural machine, Turing machine, Physarum machine, bio-inspired computing, computing automaton, slime mould, unconventional computing, computational efficiency.
2010 MSC: 68Q05, 68Q10.

1. Introduction

To better understand and further develop information processing, researchers created different mathematical models of algorithms and computation building foundations of theoretical computer science. After development and proliferation of digital computers, Turing machine became the most popular of those models impersonating the paradigm of algorithmic computational devices according to the Church-Turing Thesis. In spite of this, creation of new mathematical models has continued. There were three reasons for this situation in theoretical computer science. First, many researchers tried to build a model more powerful than Turing machine. Regardless of many attempts, the first mathematical model of algorithms and computation, for which it was proved

*Corresponding author
Email addresses: mburgin@math.ucla.edu (Mark Burgin), andrew.adamatzky@uwe.ac.uk (Andrew Adamatzky)

that it was really more powerful than Turing machine, was inductive Turing machine constructed in (Burgin, 1983). Note that although Turing machines with an oracle are more powerful than conventional Turing machines, a Turing machine with an oracle is not a constructive theoretical device, while inductive Turing machines are virtually as constructive as conventional Turing machines. Second, by its construction, Turing machine is a very simple edifice well suited for theoretical research but reflecting only some essential features of digital computers while ignoring other features. At the same time, exactly these properties of a Turing machine made it very inefficient as a computing device. Therefore, researchers thrived to build more efficient than Turing machines mathematical models, representing more key features of ever-developing computers. Examples of such models are Kolmogorov algorithms (Kolmogorov, 1953), Random Access Machines (Shepherdson & Sturgis, 1963), Random Access Machines with the Stored Program (Elgot & Robinson, 1964) and some others. The third incentive for the development of new models has been necessity to address various kinds of computations that the conventional Turing machine was not able to properly model. This intent brought forth many novel models such as:

- cellular automata (Von Neumann *et al.*, 1966), Turing machines with many heads and tapes, vector machines and systolic arrays (Kung & Leiserson, 1978) for modeling and exploration of parallel computations;
- Petri nets (Petri, 1962) and grid automata (Burgin, 2003; Burgin & Eberbach, 2009*a*) for modeling and exploration of concurrent computations;
- formal grammars (Chomsky, 1956) for modeling and exploration of linguistic transformations; and some others.

Discovery of unconventional types of computation and research in building unconventional computers such as biological, chemical, or non-linear physical computers (Adamatzky, 2001; Adamatzky *et al.*, 2005; Calude & Dinneen, 2015; Adamatzky, 2016) demand innovative mathematical models of computation that reflect peculiarity of unconventional computations (Stepney *et al.*, 2005; Cooper, 2013; Kendon *et al.*, 2015) and original models of algorithms that represent control in such computations. This is the exact goal of the development and exploration of structural machines in this paper. The paper is organized as follows. In the second section, we describe structural machines and study their properties. In the third section, we study relations between structural machines and other popular computational models. In particular, we show how structural machines model Turing machines (Theorem 1) and inductive Turing machines (Theorem 3) demonstrating (Theorem 2) that structural machines are more efficient than Turing machines. In the fourth section, we briefly describe Physarum machines, which perform biological computations, and show how structural machines model Physarum machines. In the fifth section, we discuss the obtained results and suggest future directions for research.

2. Structural machines

There are structural machines of different orders. Here we study structural machines of the first order, which work with first-order structures, and structural machines of the second order, which work with first-order and second-order structures.

Definition 2.1. From (Burgin, 2012). A first-order structure is a triad of the form $\mathbf{A} = (A, r, \mathcal{R})$ In this expression, we have:

- the set A, which is called the substance of the structure $\mathbf{A}$ and consists of elements of the structure $\mathbf{A}$, which are called structure elements of the structure $\mathbf{A}$
- the set $\mathcal{R}$, which is called the arrangement of the structure $\mathbf{A}$ and consists of relations in the structure $\mathbf{A}$, which are called structure relations of the structure $\mathbf{A}$
- the incidence relation r, which connects groups of elements from A with relations from $\mathcal{R}$.

For instance, if $\mathbf{R}$ is an n-ary relation from $\mathcal{R}$ and $a_1, a_2, a_3, \ldots$, an are elements from A, then the expression $r(R; a_1, a_2, a_3, \ldots, a_n)$ means that the elements $a_1, a_2, a_3, \ldots, a_n$ belong to the relation $\mathbf{R}$ with the name R, i.e., r connects the elements $a_1, a_2, a_3, \ldots, a_n$ with the relation $\mathbf{R}$.

Describing structures, it is significant to distinguish relations and their names because when a structural machine functions, relations, as a rule, are changing, while their names can remain the same. For instance, when a structure $\mathbf{A}$ on a set A has a ternary relation $\mathbf{R}$ with the name R and in the process of computation, the machine additionally connects three elements from A by a link assigning it to the relation $\mathbf{R}$. As a result, $\mathbf{R}$ becomes larger but preserves the same name R.

However, it is also possible, that a structural machine changes names of the structure relations, for example by splitting one relation into two new relations of the same arity.

Lists, queues, words, texts, graphs, directed graphs, labeled graphs, mathematical and chemical formulas, tapes of Turing machines and Kolmogorov complexes are particular cases of structures that have only unary and binary relations. Note that labels, names, types and properties are unary relations.

In the case when the set $\mathcal{R}$ of relations consists of one binary and several unary relations, then the first order structure is a labeled (named) graph. When $\mathcal{R}$ contains only binary and unary relations, then the first order structure is a labeled (named) multigraph.

Example 2.1. Let us consider the first order structure $\mathbf{A} = (A, r, \mathcal{R})$ where $A = \{a, b, c, d, e, f, g, h\}$ and $\mathcal{R}$ consists of one binary relation P and one ternary relation Q, the graphical representation of which is given in Figure 1.

In the case of a Physarum machine, we can interpret elements of the relation Q in Figure 1 as two clusters of nutrients, colonized by a single slime mould. There are higher degrees of connections between growth zones and branches of the protoplasmic network inside clusters but there are only few links bringing these two clusters together.

Another possibility is when a first order structure $A = (A, r, \mathcal{R})$, in which the set $\mathcal{R}$ consists of a single binary relation R can faithfully represent the structure of a living slime mould established by blobs of slime mould and active zones. Namely, elements from the set A represent blobs of slime mould and active zones, while elements from the relation R represent connecting tubes.

However, a slime mould often has a more sophisticated structure. The network of blobs, active zones and protoplasmic tubes is not uniform but forms clusters. These clusters are also connected by thick protoplasmic tubes, which represent the incidence relation that connects groups of elements from A. To model a slime mould with clusters, we need second-order structures.

Definition 2.2. From (Burgin, 2012). A second-order structure is a triad of the form

$$\mathbf{A} = (A, r, \mathcal{R})$$

Here

- the set A, which is called the substance of the structure $\mathbf{A}$ and consists of elements of the structure $\mathbf{A}$, which are called structure elements of the structure $\mathbf{A}$
- the set $\mathcal{R}$, which is called the arrangement of the structure $\mathbf{A}$ and consists of relations in the structure $\mathbf{A}$, which are called structure relations of the structure $\mathbf{A}$
- r is the incidence relation that connects groups of elements from A with relations from $\mathcal{R}$
- $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$
- $\mathcal{R}_1$ is the set of relations in the set A
- $\mathcal{R}_2$ is the set of relations in the set $\mathcal{R}_1$, i.e., elements from $\mathcal{R}_2$ are relations between relations from $\mathcal{R}_1$

Second-order structures are used to represent data processed by structural machines of the second order.

Relations from the set $\mathcal{R}$ determine the intrinsic structure of the structure $\mathbf{A} = (A, r, \mathcal{R})$. However, efficient operation with, utilization and modeling of first-order and higher-order structures demands additional (extrinsic) structures (Burgin, 2012). One of these extrinsic structures is pretopology (ech, 1966) determined by neighborhoods in the set A.

Definition 2.3. If $\mathbf{R}$ is an n–ary relation from $\mathcal{R}$, then:

1. the substantial R–neighborhood of a structure element a in a structure $\mathbf{A} = (A, r, \mathcal{R})$ is a set of the form

$$O_{RS}a = \{a\} \cup \{d; \exists i, k \exists a_2, \ldots, a_n \in A((1 \leq k \leq n) \wedge (a_1, a_2, \ldots, a_i = a, \ldots, a_k = d, \ldots, a_n) \in \mathbf{R})\}$$

2. the link R–neighborhood of a structure element a in a structure $\mathbf{A} = (A, r, \mathcal{R})$ is a set of the form

$$O_{RL}a = \{(a_1, a_2, \ldots, a_n) \in \mathbf{R}; a_1, a_2, \ldots, a_n \in O_R a\}$$

3. the full R–neighborhood of a structure element a in a structure $\mathbf{A} = (A, r, \mathcal{R})$ is the set

$$O_{RF}a = O_{RS}a \cup O_{RL}a$$

Informally, the substantial R-neighborhood of a structure element a consists of all structure elements connected to a by the relation $\mathbf{R}$. The link R–neighborhood of a structure element a consists of all elements from the relation $\mathbf{R}$ that contain a. The full R–neighborhood of a structure element a is the union of the substantial R–neighborhood and link R–neighborhood of a.

Examples:

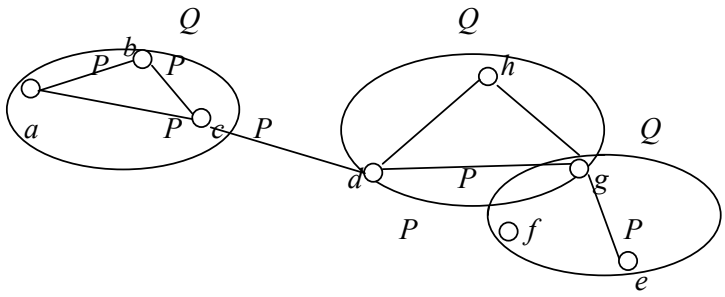

Figure 1. The graphical representation of a first order structure.

1. Taking the structure presented in Fig. 1, we see that $\{a, b, c, f\}$ is the substantial P–neighborhood of the structure element a determined by the relation P from the structure in Fig. 1.
2. Taking the structure presented in Fig. 1, we see that $\{a, b, c\}$ is the substantial Q–neighborhood of the structure element a determined by the relation Q from the structure in Fig. 1.
3. Taking the structure presented in Fig. 1, we see that $\{e, g\}$ is the substantial P–neighborhood of the structure element e determined by the relation P from the structure in Fig. 1.
4. Taking the structure presented in Fig. 1, we see that $\{f, g, e\}$ is the substantial Q–neighborhood of the structure element f determined by the relation Q from the structure in Fig. 1.
5. Taking the structure presented in Fig. 1, we see that $\{(a, b), (a, c)\}$ is the link P–neighborhood of the structure element a determined by the relation Q from the structure in Fig. 1.
6. Taking the structure presented in Fig. 1, we see that $\{(a, b, c)\}$ is the link Q–neighborhood of the structure element a and of the structure element b where both neighborhoods are determined by the relation Q from the structure in Fig. 1.

Definition 3 implies the following result.

Lemma 2.1. *Each structure element a in a structure* $\mathbf{A} = (A, r, \mathcal{R})$ *has the uniquely defined substantial R—neighborhood.*

Relations between relations from $\mathcal{R}$ imply relations between neighborhoods.

Proposition 2.1. *For any structure element a, the inclusion* $R \subseteq Q$ *of two relations* $R, Q \in \mathcal{R}$ *implies inclusions of neighborhoods*

$O_{RS}a \subseteq O_{QS}a$, $O_{RL}a \subseteq O_{QL}a$ *and* $O_{RF}a \subseteq O_{QF}a$.

Neighborhoods of elements allow us to build neighborhoods of sets of elements.

Definition 2.4. If R is an n–ary relation from $\mathcal{R}$ and $Z \subseteq A$, then:
(α) the substantial R–neighborhood of the set Z in a structure $\mathbf{A} = (A, r, \mathcal{R})$ is the set

$$O_{RS}Z = \cup_{a \in Z} O_{RS}a$$

(β) the link R–neighborhood of the set Z in a structure $\mathbf{A} = (A, r, \mathcal{R})$ is a set of the form

$$O_{RL}Z = \cup_{a \in Z} O_{RL}a$$

(γ) the full R–neighborhood of the set Z in a structure $\mathbf{A} = (A, r, \mathcal{R})$ is the set

$$O_{RF}Z = O_{RS}Z \cup O_{RL}Z$$

Lemma 1 implies the following result.

Lemma 2.2. *The substantial (link or full) R–neighborhood of any set Z of structure elements is uniquely defined.*

Proposition 1 implies the following result.

Proposition 2.2. *For any set Z of structure elements, the inclusion $R \subseteq Q$ of two relations $R, Q \in \mathcal{R}$ implies inclusions of neighborhoods $O_{RS}Z \subseteq O_{QS}Z, O_{RL}Z \subseteq O_{QL}Z$ and $O_{RF}Z \subseteq O_{QF}Z$.*

Substantial neighborhoods of sets of structure elements determine pretopology in the set A of all structure elements. We remind that a pretopological space is defined as a set X with a preclosure operator (ech closure operator) cl. Let 2^X be the power set of X. A preclosure operator on a set X is a mapping $cl : 2^X \to 2^X$ that satisfies the following axioms (ech, 1966):

- $cl(\emptyset) = \emptyset$
- $Z \subseteq cl(Z)$ for any $Z \subseteq X$
- $cl(Z \cup Y) \subseteq cl(Z) \cup cl(Y)$ for any $Z, Y \subseteq X$
- $Y \subseteq Z$ implies $cl(Y) \subseteq cl(Z)$ for any $Z, Y \subseteq X$

Properties of substantial R–neighborhoods allow us to prove the following result.

Proposition 2.3. *Substantial R–neighborhoods define a pretopology in the set A.*

Proof. Let us define the closure $cl(Z)$ of a set $Z \subseteq A$ equal to its substantial R–neighborhood O_{RSX} and check the axioms of pretopological spaces.

Axioms 1 and 2 are true by definition as $cl(\emptyset) = \emptyset$ and $Z \subseteq cl(Z)$ for any $Z \subseteq A$. In addition,

$cl(Z \cup Y) = O_{RS}(Z \cup Y) = \cup_{a \in Z \cup Y} O_{RS}a = (\cup_{a \in Z} O_{RS}a) \cup (\cup_{a \in Y} O_{RS}a) = O_{RS}Z \cup O_{RS}Y = cl(Z) \cup cl(Y)$

This gives us Axiom 3. Axiom 4 is implied by Proposition 2. □

Definition 2.5. • The substantial $\mathcal{R}$–neighborhood of a structure element a in a structure $\mathbf{A} = (A, r, \mathcal{R})$ is the set

$$O_{\mathcal{R}\mathbf{S}}a = \cup_{\mathbf{R} \in \mathcal{R}} O_{RS}a$$

- The link $\mathcal{R}$–neighborhood of a structure element a in a structure $\mathbf{A} = (A, r, \mathcal{R})$ is a set of the form

$$O_{\mathcal{R}\mathbf{L}}a = \cup_{\mathbf{R}\in\mathcal{R}} O_{RS}\, a$$

- The full $\mathcal{R}$–neighborhood of a structure element a in a structure $\mathbf{A} = (A, r, \mathcal{R})$ is the set

$$O_{\mathcal{R}\mathbf{F}a} = \cup_{\mathbf{R}\in\mathcal{R}} O_{RF}\, a$$

Definition 5 implies the following result.

Lemma 2.3. *The substantial (link or full) $\mathcal{R}$–neighborhood of a structure element a in a structure $\mathbf{A} = (A, r, \mathcal{R})$ is uniquely defined.*

Properties of union and Definitions 4 and 5 imply the following result.

Lemma 2.4. $O_{\mathcal{R}\mathbf{F}}a = O_{\mathbf{RS}}a \cup O_{\mathbf{RL}}a$.

Proposition 3 and Definition 4 imply the following result.

Corollary 2.1. *Substantial $\mathcal{R}$–neighborhoods define a pretopology in the set A.*

Neighborhoods of elements allow us to build neighborhoods of sets of elements.

Definition 2.6. If $Z \subseteq A$, then:
α : the substantial $\mathcal{R}$–neighborhood of the set Z in a structure $\mathbf{A} = (A, r, \mathcal{R})$ is the set

$$O_{\mathcal{R}\mathbf{S}}Z = \cup_{a\in Z} O_{\mathcal{R}\mathbf{S}a}$$

β: the link $\mathcal{R}$–neighborhood of the set Z in a structure $\mathbf{A} = (A, r, \mathcal{R})$ is a set of the form

$$O_{\mathcal{R}\mathbf{L}}Z = \cup_{a\in Z} O_{\mathcal{R}\mathbf{L}}a$$

γ: the full **R**–neighborhood of the set Z in a structure $\mathbf{A} = (A, r, \mathcal{R})$ is the set

$$O_{\mathbf{RF}}Z = O_{\mathbf{RS}}Z \cup O_{\mathbf{RL}}Z$$

Lemma 3 implies the following result.

Lemma 2.5. *The substantial (link or full) $\mathcal{R}$–neighborhood of any set Z of structure elements is uniquely defined.*

Let us study properties of neighborhoods in structures.

Definition 2.7. If R is an n-ary relation from $\mathcal{R}$, then the substantial R–neighborhood of a structure element a in a structure $\mathbf{A} = (A, r, \mathcal{R})$ is symmetric if for any elements $a_1, a_2, a_3, \ldots, a_n$ from A and any permutation $i_1, i_2, i_3, \ldots, i_n$ of the numbers $1, 2, 3, \ldots, n$, we have

$$(a_1, a_2, a_3, \ldots, a_n) \in R$$

if and only if

$$(a_{i_1}, a_{i_2}, a_{i_3}, \ldots, a_{i_n}) \in R$$

Lemma 2.6. *If the substantial R–neighborhood $O_{RS}a$ of a structure element a is symmetric, a structure element b belongs to the R–neighborhood $O_{RS}a$ if and only if a belongs to the substantial R–neighborhood $O_{RS}b$ of b.*

Definition 2.8. If $\mathbf{R}$ is an n-ary relation from $\mathcal{R}$, then the R–neighborhood of a structure element a in a structure $\mathbf{A} = (A, r, \mathcal{R})$ is transitive if for any elements $a_1, a_2, a_3, \ldots, a_n, d_1, d_2, d_3, \ldots, d_n$ from A and any numbers i and j from the set $\{1, 2, 3, \ldots, n\}$, we have

If $(a_1, a_2, a_3, \ldots, a_n) \in \mathbf{R}$ and $(d_1, \ldots, a_i, \ldots, d_n) \in \mathbf{R}$, then there are elements $b_1, b_2, b_3, \ldots, b_n$ from A such that $(b_1, \ldots, a_i, \ldots, d_j, \ldots, b_n) \in \mathbf{R}$

Proposition 2.4. *If a relation $\mathbf{R} \in \mathcal{R}$ is symmetric and transitive, then substantial R–neighborhoods of two structure elements either coincide or do not intersect.*

Corollary 2.2. *If all relations in $\mathcal{R}$ are symmetric and transitive, then substantial $\mathcal{R}$–neighborhoods of two structure elements either coincide or do not intersect.*

Corollary 2.3. *If $\mathcal{R}$ consists of one symmetric binary relation R, i.e., $\mathbf{A}$ is a graph and R is also transitive, then any substantial $\mathcal{R}$–neighborhood (R–neighborhood) is a complete graph*

Proposition 2.5. *If a relation $\mathbf{R} \in \mathcal{R}$ is symmetric and transitive, then the system of substantial R–neighborhoods forms a base of a topology in A.*

Corollary 2.4. *If all relations in $\mathcal{R}$ are symmetric and transitive, then the system of substantial $\mathcal{R}$–neighborhoods forms a base of a topology in A.*

Definition 2.9. The type $T(A)$ of a first-order structure $\mathbf{A} = (A, r, \mathcal{R})$ is the set $\{(R, \alpha(R)); R \in \mathcal{R}\}$ of pairs $(R, \alpha(R))$; where $\alpha(R))$ is the arity of the relation $\mathbf{R}$ with the name R.

f

We assume that two first-order structures $\mathbf{A} = (A, r, \mathcal{R})$ and $\mathbf{B} = (B, p, \mathcal{P})$ have the same type if there is a one-to-one mapping $f : T(A) \to T(B)$ such that if $f(R, \alpha(R)) = (P, \alpha(P)), then \alpha(R) = \alpha(P)$.

For instance, all binary relations have the same type.

A structural machine M works with structures of a given type and has three components:

- The control device C_M regulates the state of the machine M
- The processor P_M performs transformation of the processed structures and its actions (operations) depend on the state of the machine M and the state of the processed structures
- The functional space $S\,p_M$ consists of three components:
  - The input space In_M, which contains the input structure.
  - The output space Out_M, which contains the output structure.
  - The processing space PS_M, in which the input structure(s) is transformed into the output structure(s).

We assume that all structures — the input structure, the output structure and the processed structures — have the same type.

Computation of a structural machine M determines the trajectory of computation, which is a tree in general case and a sequence in the deterministic case and a single processor unit.

There are two forms functional spaces $S\,p_M$ and $US\,p_M$:

- $S\,p_M$ is the set of all structures that can be processed by the structural machine M and is called a categorical functional space
- $US\,p_M$ is a structure for which all structures that can be processed by the structural machine M are substructures and is called a universal functional space

There are three basic types of control devices:

- A central control device controls all processors of the structural machine
- A cluster control device controls a cluster of processors in the structural machine
- An individual control device controls a single processor in the structural machine

There are two basic types of processors:

- A localized processor is a single abstract device
- A distributed processor, which is also called a total processor, consists of a system of unit processors or processor units

In turn, there are three basic types of distributed processors:

- A homogeneous distributed processor consists of a system of identical unit processors, i.e., all these unit processors are copies of one processor
- An almost homogeneous distributed processor consists of a system in which almost all unit processors are identical
- A heterogeneous distributed processor consists of a system of different unit processors

As a result, we have three structural types of processors.

There are also three basic classes of distributed processors:

- In a synchronous distributed processor, all unit processors perform each operation at the same time
- In an almost synchronous distributed processor, almost all unit processors perform each operation at the same time
- In an asynchronous distributed processor, unit processors function independently

A cellular automaton is a synchronous distributed processor, while a Petri net is an asynchronous distributed processor.

It is natural to suppose that each unit processor performs only local operations. In a general case, each unit processor moves from one structure element to another, performing operations in their neighborhoods (Fig. 2). This makes it possible to consider a localized processor as a special type of a distributed processor with one unit processor.

A standard example of a localized processor is the head of a Turing machine with one head or a finite automaton. One head of TM correspond to one growth zone of the slime mould.

A standard example of a homogeneous distributed processor is the system of all heads of a Turing machine with several heads. It is also possible to perceive a Physarum machine as multiprocessor structural machine.

Examples of heterogeneous distributed processors are processing devices in evolutionary automata such as evolutionary finite automata, evolutionary Turing machines or evolutionary inductive Turing machines (Burgin & Eberbach, 2009*b*).

Note that not all heterogeneous distributed processors are the same and to discern their properties it is possible to use measures of homogeneity constructed in (Burgin & Bratalskii, 1986).

There are three types of localized processors:

- A processor localized to one structure element (e.g., a node)
- A processor localized to an R–neighborhood of one structure element (e.g., a node) where $\mathbf{R}$ is a relation from $\mathcal{R}$
- A processor localized to an $\mathcal{R}$–neighborhood of one structure element (e.g., a node)

In what follows, localization of a processor is formalized by the concept of the processor topos.

There are three sorts of distributed processors:

- A constant distributed processor has a fixed number of localized unit processors.
- A variable distributed processor can change the number of localized unit processors.
- A growing distributed processor can increase the number of localized unit processors.

Growing distributed processors are special kinds of variable distributed processors.

There are three types of variable (growing) distributed processors:

- In a bounded variable (growing) distributed processor, the quantity of localized unit processors is always between two numbers, e.g., between 1 and 10, (is bounded by some number).
- An unbounded variable distributed processor can use any finite number of localized unit processors in its functioning.
- An infinite distributed processor can use any (even infinite) number of localized unit processors.

Cellular automata give examples of structural machines with infinite distributed processors. One-dimensional cellular automata work with such structures as words. Two-dimensional cellular automata work with such structures as two-dimensional arrays.

Now let us consider characteristics of unit processors in structural machines. Each unit processor p of a structural machine M has its topos, observation zone and operation zone.

Definition 2.10. The topos T_p of a unit processor p is the part of the structure occupied by this processor. When we take into account time of processing, the topos T_p of the processor p is denoted by $T_p(t)$.

This definition allows defining topoi for total processors of structural machines.

Definition 2.11. The topos T_A of a total (distributed) processor A is the union of the topoi of all its unit processors. When we take into account time of processing, the topos T_A of the processor A is denoted by $T_A(t)$.

Note that in the general case, the topos of a processor can be infinite although the constructive conditions on algorithms prohibit infinite topoi. Cellular automata give an example of a processor with an infinite topos.

Axiom T. Topoi of different unit processors do not intersect.

Localization of unit processors implies restrictions on their topoi. Namely, the topos of a unit processor localized to one structure element consists of this structure element, the topos of a unit processor localized to an R–neighborhood of one structure element is a part of this neighborhood, and the topos of a unit processor localized to an $\mathcal{R}$–neighborhood of one structure element is a part of that neighborhood.

Definition 2.12. The observation zone Ob_p of a unit processor p is the part of the structure $S\,p_M$ observed by this processor from its topos. When we take into account time of processing, the observation zone Ob_p of the processor p is denoted by observation zone $Ob_p(t)$.

As in the case of topoi, we define observation zones for total processors.

Definition 2.13. The observation zone Ob_A of a total (distributed) processor A is the union of the observation zones of all its unit processors. When we take into account time of processing, the observation zone Ob_A of the processor A is denoted by observation zone $Ob_A(t)$.

It is natural to suppose that observation zones of processors impact their functioning.

Axiom Z. Operations performed by unit processors depend only on their observation zone.

Definition 2.14. The operation zone Op_p of a unit processor p is the part of the structure $S\,p_M$ that can be changed by this processor from its topos. When we take into account time of processing, the operation zone Op_p of the processor p is denoted by observation zone $Op_p(t)$.

For instance, the head of a Turing machine with one linear tape is unit processor. The topos of the head is one cell in the tape. Its observation zone is the same cell and the symbol written in it. Its operation zone is the symbol written in the cell occupied by the head.

Usually, these parts of the functional space PS_M satisfy the following conditions:

$$T_p \subseteq Op_p \subseteq Ob_p$$

and

$$T_p(t) \subseteq Op_p(t) \subseteq Ob_p(t)$$

Informally, it means that the topos of a processor is inside its operation zone, while it is possible to perform operations only inside the observation zone.

These conditions are true, for example, for Turing machines, but they are not satisfied for pushdown automata (Hopcroft *et al.*, 2001).

Often we have $Op_p = Ob_p$ and T_p consists of a single node (element from A).

As in the case of observation zones, we define operation zones for total processors.

Definition 2.15. The operation zone Op_A of a total (distributed) processor A is the union of the operation zones of all its unit processors.. When we take into account time of processing, the operation zone Op_A of the processor A is denoted by operation zone $Op_A(t)$.

Functioning of processors in structural machines includes not only structure transformations but also transitions from one topos to another.

Definition 2.16. The transition zone Tr_p of a unit processor p consists of all topoi where p can move in one step from its present topos.

For instance, the transition zone Tr_h of the head h of a Turing machine with one linear tape consists of three adjacent cells with h is situated in the middle cell.

In some cases, it is useful to assume that the transition zone of a unit processor is included in its observation zone.

Definition 2.17. A processor unit p is called: ¢ topologically uniform if all its topoi are isomorphic ¢ operationally uniform if all its operation zones are isomorphic ¢ transitionally uniform if all its transition zones are isomorphic ¢ observationally uniform if all its observation zones are isomorphic ¢ topologically standardized if all its topoi have the same type, e.g., are R–neighborhoods ¢ operationally standardized if all its operation zones have the same type, e.g., are R–neighborhoods ¢ transitionally standardized if all its transition zones have the same type, e.g., are R–neighborhoods ¢ observationally standardized if all its observation zones have the same type, e.g., are R–neighborhoods

For instance, processor unit p is topologically uniform if all its topoi consist of a single node.

Lemma 2.7. *Any topologically (operationally, transitionally or observationally) uniform processor unit p is topologically (correspondingly, operationally, transitionally or observationally) standardized.*

Usually, processors of abstract and physical automata (machines) are topologically operationally, transitionally and observationally uniform. At the same time, processors in chemical and biological automata (machines) can be non-uniform. An example of a non-uniform processor

is a processor that can read from or write to up to five cells in the memory. Thus, when this processor writes to one cell, its topos consists of one cell, while when this processor writes to three cell, its topos consists of these three cells.

Topoi, observation zones and operation zones of unit processors allow us to define topoi, observation zones and operation zones of distributed processors.

There are different types of processor units. A processor unit can be:

- Controlled (by the central control device of the structural machine).
- Autonomous, when it has its own control device.
- Cooperative, when it has its own control device but the functioning of this processor unit depends on the states both of its own control device and of the central control device of the structural machine.

For instance, in a many-head Turing machine T, all heads are controlled processor units. The control device of T controls them. At the same time, all finite automata in a cellular automaton are autonomous processor units.

We remind that a finite state machine also called a finite state automaton is an abstract system that can be in a finite number of different finite states and functioning of which is described as changes of its states.

Proposition 2.6. *A structural machine M is a finite state machine if and only if:*

- *Its structural space $S\,p_M$ is finite, i.e., in the case of universal structural space, it is a finite structure, or in the case of categorical structural space, it consists of a finite number of finite structures.*
- *The number of unit processors is finite and each of them can be in a finite number of different finite states.*

For instance, a finite automaton is a finite state machine, while a Turing machine is not a finite state machine.

Definition 2.18. A temporally finite state machine is an abstract system that can be in a finite number of different finite states at any moment of time and functioning of which is described as changes of its states.

Proposition 2.7. *A structural machine M is a temporally finite state machine if and only if:*

- *At any moment of time, its structural space $S\,p_M$ is finite, i.e., in the case of universal structural space, it is a finite structure, or in the case of categorical structural space, it consists of a finite number of finite structures.*
- *At any moment of time, the number of unit processors is finite and each of them can be in a finite number of different finite states.*

- *Any operation of each unit processor involves only a finite number of structure elements and relations*

For instance, a Turing machine is a temporally finite state machine, while finite dimensional and general machines of (Blum *et al.*, 1989) are not temporally finite state machines.

Definition 2.19. An operation of a processor is local or more exactly, unilocal if it is performed with one structural element (e.g., node) and some (all) of its relations (a pointed operation), e.g., deleting a structural element (e.g., a node) and all its binary connections (links or edges), adding a link to a structural element or changing a label of a structural element.

For instance, the head h of a Turing machine performs only local operations, while the head of a pushdown automaton can perform nonlocal operations. Processors of automata that perform operations of unrestricted formal grammars are mostly nonlocal.

Definition 2.20. An operation of a processor P is $\mathcal{R}$–local if it is performed with elements (e.g., nodes) from the $\mathcal{R}$–neighborhood of a definite element (e.g., node) and with some (all) of their relations (a singularly local operation).

An operation of a processor P is topologically $\mathcal{R}$–local if it is performed with elements (e.g., nodes) from the $\mathcal{R}$–neighborhood of a topos of P (e.g., node) and with some (all) of their relations (a topologically local operation).

Proposition 2.8. *If (1) a structural machine M operates with structures in which $\mathcal{R}$ contains only one binary relation, (2) a topos of a topologically uniform processor P of M is one structural element, and (3) an operation O of P is topologically local, then O is singularly local.*

Definition 2.21. a) An operation of a processor is $\mathcal{R}$–local or totally local if it is performed with elements (e.g., nodes) from the $\mathcal{R}$–neighborhood of a definite element (e.g., node) and with some (all) of their relations. b) An operation of a processor P is topologically $\mathcal{R}$–local if it is performed with elements (e.g., nodes) from the $\mathcal{R}$–neighborhood of a topos of P (e.g., node) and with some (all) of their relations (a wholly local operation).

Proposition 2.9. *If a topos of a topologically uniform processor P is one structural element and an operation O of P is wholly local, then O is totally local.*

Definitions imply the following result.

Proposition 2.10. *If R belongs to $\mathcal{R}$, then any R–local operation is $\mathcal{R}$–local.*

Let us consider operations performed by processors of structural machines.
The first group of operations consists of the transition operations:

1. Moving the processor from one topos, e.g., a structure element, to another topos. This operation is local when both elements belong to some relation from $\mathcal{R}$.
2. Changing the operation zone of the processor
3. Changing the observation zone of the processor

The second group of operations consists of the substantial transforming operations:

1. Adding a structure element, e.g., a node.
2. Deleting (removing) a structure element, e.g., a node, from a neighborhood of the element where the processor is situated and all relations that include this element.
3. Deleting (removing) a link from a relation R that connects some structure elements with the element where the processor is situated.
4. Adding a link to a relation $\mathbf{R}$ that connects some structure elements with the element where the processor is situated.
5. Deleting (removing) a relation $\mathbf{R}$ from $\mathcal{R}$.
6. Adding a new relation to R.

The third group of operations consists of the symbolic transforming operations:

1. Renaming a node
2. Naming a node
3. Denaming a node, i.e., deleting the name of a node
4. Renaming a link
5. Naming a link
6. Denaming a link, i.e., deleting the name of a link

Example 2.2. Operation of deleting the element f from the first order structure $\mathbf{A} = (A, r, \mathcal{R})$ in Fig. 2 where (A) shows the structure before operation and (B) shows the structure after operation. Note that such operations often change relations in the processed structure. Besides, the processor (processor unit) moves from the place (position) f to the place (position) g (see Fig. 2). This operation is performed according to the instruction $(q, f, f) \to (q, g, \sim f)$, in which q is the state of the processor (processor unit), f is the place (position) of the processor (processor unit) before the operation, g is the place (position) of the processor (processor unit) after the operation and $\sim f$ means elimination of f.

Example 2.3. Operation of adding the relation P for elements g and f in the first order structure $\mathbf{A} = (A, r, \mathcal{R})$ where (A) shows the structure before operation and (B) shows the structure after operation (see Fig. 3). Besides, the processor (processor unit) moves from the place (position) g to the place (position) e. This operation is performed according to the instruction $(p, g, f) \to (p, e, P(g, f))$, in which p is the state of the processor (processor unit), g is the place (position) of the processor (processor unit) before the operation, e is the place (position) of the processor (processor unit) after the operation, f is an observed element and means addition of the pair (g, f) to the relation P.

Structural machines can simulate Turing machines, Kolmogorov algorithms (machines), storage modification machines and cellular automata. We prove some of these results in the next section.

Structural machines also can simulate processes generated by logical calculi (Schumann & Adamatzky, 2011), λ-calculus and formal grammars being able to perform operations used in

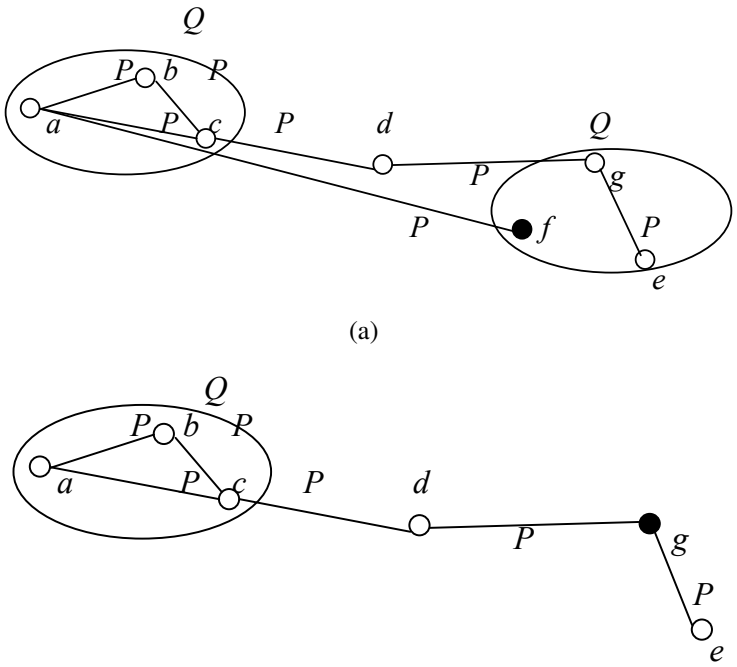

Figure 2. The graphical representation of an operation on first order structures.

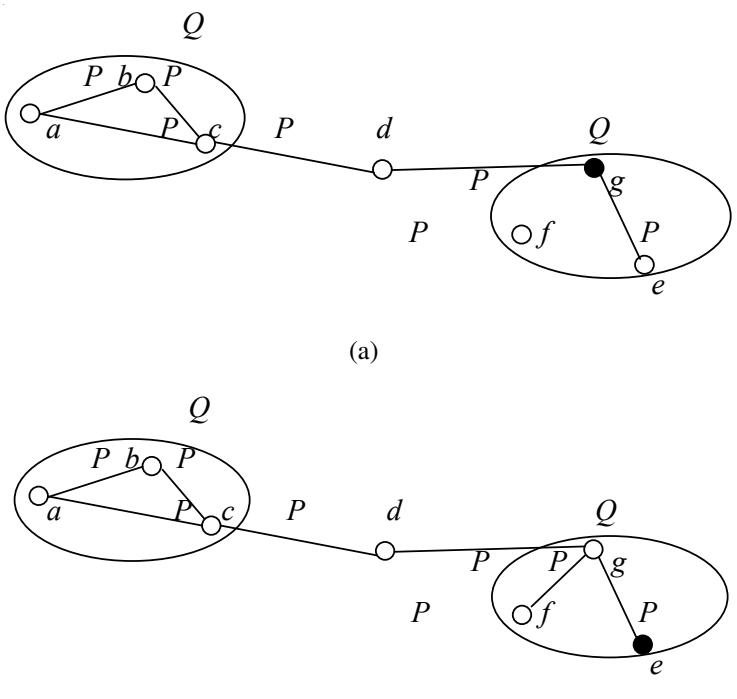

Figure 3. The graphical representation of an operation on first order structures.

various databases. Structural machines can also compute partial recursive functions and limit partial recursive functions.

It is possible to build structural machines that can work not only with discrete but also with continuous data because structures can be continuous and there are no restrictions on relations in processed structures. This possibility turns artificial neural networks, Shannons differential analyzer (Shannon, 1941), a finite dimensional and general machines of (Blum *et al.*, 1989) and Type 2 Turing machines (Weihrauch, 2000) into special cases of structural machines.

This shows that it is practical to discern discrete structural machines, which work with discrete structures, have discrete systems of states and operations, and continuous structural machines. In continuous structural machines one two or all three of the following components can be continuous, i.e., continuous processed structures, continuous system of states and/or continuous operations.

Being able to construct various forms of structural machines and their flexibility show that it is natural to use structural machines for a theoretical study of natural computations performed by biological, chemical and physical systems. For instance, in Section 4, we show how to use structural machines as abstract automata for simulation of such biological automata as Physarum machines (Adamatzky, 2007) based on slime mold computations.

3. Structural machines, Turing machines and inductive Turing machines

In this section, we study relations between structural machines and other popular computational models.

Theorem 3.1. *A structural machine with a centralized processor can simulate any Turing machine with the linear time complexity.*

It is sufficient to simulate a Turing machine with one linear tape and one head (cf., for example, (Burgin, 2005; Hopcroft *et al.*, 2001)).

When we want to model a Turing machine by a structural machine, it is easy to build a linear structure of elements with sequential elements connected by a binary relation and this structure will represent the linear tape of the Turing machine. However, the question is how the structural machine will discern left from right simulating the moves of the Turing machine. Here we give three solutions to this problem.

Let us consider a Turing machine $T = (A, Q, q_0, F, R)$. This formula shows that the Turing machine T is determined by the alphabet $\Sigma = \{a_1, a_2, \ldots, a_m\}$, the set of states $Q = \{q_0, q_1, q_2, \ldots, q_n\}$, the set of the final states $F = \{p_1, p_2, \ldots, p_u\} \subseteq Q$, the start state q_0 and the system of rules R, each of which has the form

$$q_h a_i \to a_j q_k$$

$$q_h \Lambda \to a_j q_k$$

$$q_h a_i \to \Lambda q_k$$

$$q_h a_i \to q_k L$$

$$q_h a_i \to q_k c R$$

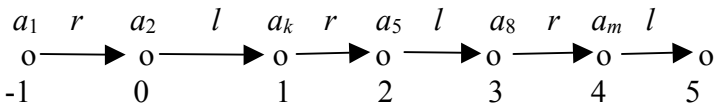

Figure 4. The graphical representation of processed first-order structures.

where $a_i, a_j \in A$, $q_h, q_k \in Q$, R means that the head of T moves to the right, L means that the head of T moves to the left and Λ is the empty symbol, which denotes empty cell of the memory.

Here we give a linear solution to the problem of simulation, i.e., we use only one-dimensional first-order structures in the simulation.

Let us describe the structural machine $M_T = (C, P, \mathcal{R})$ with the control device C, processor P and processing space $\mathcal{R}$, which coincides with the input space and output space, such that M_T simulates the Turing machine T.

In it, C is a finite automaton with the states $Q^o = \{q_0^{odd}, q_1^{odd}, q_2^{odd}, \ldots, q_n^{odd}, q_0^{even}, q_1^{even}, q_2^{even}, \ldots, q_n^{even}\}$, the final states $F^o = \{p_1^{odd}, p_2^{odd}, \ldots, p_u^{odd}, p_1^{even}, p_2^{even}, \ldots, p_u^{even}\} \subseteq Q^o$, and the start state q_0^{odd}.

The processor P is similar to the head of a Turing machine but it observes not only the content of a cell but also its connections, and P can change not only the content of the observed cell but also its connections.

$\mathcal{R} = \{R, L, \Sigma\}$ where: R is a binary relation elements of which are denoted (named) by r; L is a binary relation elements of which are denoted (named) by l; Σ contains unary relations a where $a \in \Sigma$.

Relations r and l are also called connections or links. The unary relations a from Σ are used for naming the structure elements by symbols from Σ.

The machine M_T processes first-order structures that have the form $A = (N, z, \mathcal{R}_a)$ where $N = \{-h, -h+1, \ldots, -1, 0, 1, 2, 3, \ldots, k; -h < k+1\}$, the processing space $\mathcal{R}_a$ contains two binary relations $R_a \subseteq \mathbf{R}$ and $L_a \subseteq L$ in which links r connect elements $2t$ and $2t+1$ for all integer numbers t with $-h \leq 2t \leq k--1$, while links l connect elements $2t--1$ and $2t$ for all integer numbers t with $-h+1 \leq 2t \leq k$; and m unary relations $a_1, a_2, \ldots, a_m$ and z is the assignment of relations from $\mathcal{R}_a$ (names from the alphabet Σ) to elements from N. Note that the number $-h$ can be positive, e.g., when $h = -3$.

An arbitrary input structure has the form $A_0 = (N_0 = \{1, 2, 3, \ldots, m\}, z_0, \mathcal{R}_0)$ where $\mathcal{R}_0 =$ contains two binary relations $\mathbf{R}_0 \subseteq R$ and $L_0 \subseteq L$ in which links r connect elements $2t$ and $2t+1$ for all t with $1 \leq 2t \leq x--1$, while links l connect elements $2t--1$ and $2t$ for all t with $1 \leq 2t \leq x$; and m unary relations $a_1, a_2, \ldots, a_m$ and z_0 is the assignment of relations from $\mathcal{R}_0$ (names from the alphabet Σ) to the elements from N_0 , of the relation r to the pairs of elements $2t$ and $2t+1$ for all t with $1 \leq 2t \leq x--1$ and of the relation l to the pairs of elements $2t--1$ and $2t$ for all t with $1 \leq 2t \leq x$.

Expressions in the rules for the machine M_T have the following meanings:
$a_i(r, l)$ in the left part of a rule means that the processor observes three relations a_i, r and l at the named element where the processor is situated.
$a_i(l)$ in the left part of a rule means that the processor observes only two relations a_i and l at the named element where the processor is situated.

$a_i(r)$ in the left part of a rule means that the processor observes only two relations a_i and r at the named element where the processor is situated.
$o(r, l)$ in the left part of a rule means that the processor observes two relations (connections) r and l at the element where the processor is situated.
$o(l)$ in the left part of a rule means that the processor observes only one relation (connection) l at the element where the processor is situated.
$o(r)$ in the left part of a rule means that the processor observes only one relation (connection) r at the element where the processor is situated.

The symbol o means a structure element to which no unary relation a_i is assigned, that is, an empty or not named structure element.

The symbol a_i in a rule means a structure element to which the unary relation (name) a_i is assigned.

We build rules of the in the machine M_T following way:
Renaming rules
Two rules $q_h^{odd}a_i(r, l) \to a_j q_k^{odd}$ and $q_h^{even}a_i(r, l) \to a_j q_k^{even}$ are assigned to the rule $q_h a_i \to a_j q_k$
Two rules $q_h^{odd}o(r, l) \to a_j q_k^{odd}$ and $q_h^{even}o(r, l) \to a_j q_k^{even}$ are assigned to the rule $q_h\Lambda \to a_j q_k$
Two rules $q_h^{odd}a_j(r, l) \to oq_k^{odd}$ and $q_h^{even}a_j(r, l) \to oq_k^{even}$ are assigned to the rule $q_h a_j \to \Lambda q_k$

Transition rules
Two rules $q_h^{odd}a_i(r, l) \to q_k^{even}r$ and $q_h^{even}ai(r, l) \to q_k^{odd}l$ are assigned to the rule $q_h a_i \to q_k L$
Two rules $q_h^{odd}a_i(r, l) \to q_k^{even}l$ and $q_h^{even}a_i(r, l) \to qkoddr$ are assigned to the rule $q_h a_i \to q_k R$
Two rules $q_h^{odd}(r, l) \to q_k^{even}r$ and $q_h^{even}o(r, l) \to a_j q_k oddl$ are assigned to the rule $q_h\Lambda \to q_k L$
Two rules $q_h^{odd}(r, l) \to q_k^{even}l$ and $q_h^{even}o(r, l) \to qkoddr$ are assigned to the rule $q_h\Lambda \to q_k R$
Construction rules
$q_h^{odd}aj(l) \to q_h^{odd}a_j(l)[o]$ [creation of a new structural element near the named (full) end-element where the processor is situated],
$q_h^{odd}aj(r) \to q_h^{odd}a_j(r)[o]$ [creation of a new structural element near the named (full) end-element where the processor is situated],
$q_h^{even}aj(l) \to q_h^{even}[o]$ [creation of a new structural element near the named (full) end-element where the processor is situated],
$q_h^{even}aj(r) \to q_h^{even}[o]$ [creation of a new structural element near the named (full) end-element where the processor is situated],
$q_h^{odd}aj(l) \to q_h^{odd}a_j(r, l)$ [connecting a new structural element to the nearby (full) named end-element where the processor is situated],
$q_h^{odd}aj(r) \to q_h^{odd}a_j(r, l)$ [connecting a new structural element to the nearby (full) named end-element where the processor is situated],
$q_h^{even}aj(l) \to q_h^{even}a_j(r, l)$ [connecting a new structural element to the nearby (full) named end-element where the processor is situated],
$q_h^{even}aj(l) \to q_h^{even}a_j(r, l)$ [connecting a new structural element to the nearby (full) named end-element where the processor is situated],
$q_h^{odd}o(l) \to q_h^{odd}[o]$ [creation of a new structural element near the empty (not named) end-element where the processor is situated],

$q_h^{odd}o(r) \to q_h^{odd}[o]$ [creation of a new structural element near the empty (not named) end-element where the processor is situated],
$q_h^{even}o(l) \to q_h^{even}[o]$ [creation of a new structural element near the empty (not named) end-element where the processor is situated],
$q_h^{even}o(r) \to q_h^{even}[o]$ [creation of a new structural element near the empty (not named) end-element where the processor is situated],
$q_h^{odd}o(l) \to q_h^{odd}o(r,l)$ [connecting a new structural element to the nearby empty (not named) end-element where the processor is situated],
$q_h^{even}o(r) \to q_h^{even}o(r,l)$ [connecting a new structural element to the nearby empty (not named) end-element where the processor is situated],
$q_h^{even}o(l) \to q_h^{even}o(r,l)$ [connecting a new structural element to the nearby empty (not named) end-element where the processor is situated],
$q_h^{even}o(l) \to q_h^{even}o(r,l)$ [connecting a new structural element to the nearby empty (not named) end-element where the processor is situated].

Construction operations add new structural elements and lacking relations, allowing processor to perform prescribed movements in all cases.

The structural machine M_T works following these rules. When it comes either to a state q_h^{even} or q_h^{odd} with q_h from F, then the machine M_T stops and the created structure with filled structure elements is the result of computation of the structural machine M_T.

When the processor of the structural machine M_T is at an odd structure element, then the state of the machine is some q_h^{odd}, and when the processor is at an even structure element, then the state of the machine is some q_h^{even}. When the head of the Turing machine T comes to the end of the tape, the processor of the structural machine M_T creates an empty structure element and the adequate connection to this element. The goal is to allow the processor of M_T to go (by this connection) to the empty structure element, simulating in such a way the move of the head of T to the empty cell.

As a result of these operations, the structural machine M_T simulates the Turing machine T, while the number of performed operations is $O(n)$ when T makes n operations.

Theorem is proved.

Being able to simulate Turing machines, structural machines can be more efficient than Turing machines. To show this, we take some alphabet Σ and consider the following algorithmic problem.

The Word Symmetry Problem. Given an arbitrary word w in the alphabet Σ, find if $w = uu^\star$ for some word u where $u^\star$ is the inverse of u.

A word $w = uu^\star$ is called a palindrome because written backwards it coincides with itself.

Theorem 3.2. *A structural machine with a centralized processor can solve the Word Symmetry Problem with the linear time complexity, i.e., with time $T(n) = O(n)$.*

Let us describe the structural machine $M = (C, P, \mathcal{R})$ with the control device C, processor P and processing space $\mathcal{R}$, which coincides with the input space and output space, such that M solves the Word Symmetry Problem with the linear time complexity.

In it, the control device C is a finite automaton with the states $Q = \{q_0, (q_0, a), (q_1, a), (q_2, a), q_f; a \in \Sigma\}$, the final states $F = \{q_f\} \subseteq Q$, and the start state q_0 .

The processor P is similar to the head of a Turing machine but it observes not only the content of a cell but also its connections, and P can change not only the content of the observed cell but

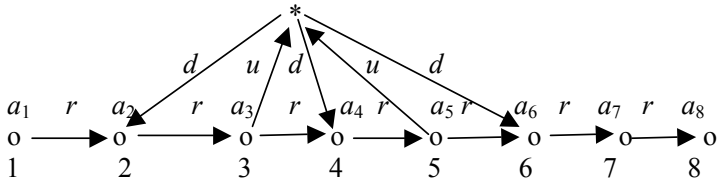

Figure 5. The graphical representation of processed first order structures.

also its connections.

$\mathcal{R} = \{R, L, U, D, £\}$ where:
R is a binary relation elements of which are denoted (named) by r;
U is a binary relation elements of which are denoted (named) by u;
DB is a binary relation elements of which are denoted (named) by db;
DE is a binary relation elements of which are denoted (named) by de;
L is a binary relation elements of which are denoted (named) by l;
Σ contains unary relations a where $a \in \Sigma$.
Relations r, u, d and l are also called connections or links. The unary relations a from Σ are used for naming the structure elements by symbols from Σ.

The machine M processes first-order structures that have the form $A = (N \cup \{\star\}; z, \mathcal{R}a)$ where $N = \{1, 2, 3, \ldots, k\}$, $\mathcal{R}a = \{$ our binary relations $D_a \subseteq D$, $U_a \subseteq U$, $R_a \subseteq R$ and $L_a \subseteq L$ in which links r and l can connect elements t and $t + 1$ for all $t = 1, 2, 3, \ldots, k - 1$, while links u and d can connect elements t with the element $\star$; and m unary relations $a_1, a_2, \ldots, a_m \in \Sigma \}$ and z is the assignment of relations from $\mathcal{R}a$ (names from the alphabet Σ) to elements from N.

An arbitrary input structure has the form $A_0 = (N_0 = \{1, 2, 3, \ldots, h\}, z_0, \mathbf{R}_0)$ where $R_0 = \{$ a binary relation $\mathbf{R}_0 \subseteq \mathbf{R}$ and m unary relations $a_1, a_2, \ldots, a_m \in \Sigma\}$ and links r from R_0 connect elements t and $t + 1$ for all $t = 1, 2, 3, \ldots, h - 1$.

The processor P of the machine M performs the following operation:
mv$[c(a, b)]$ denotes the transition of P from its topos (the cell where it is situated) with the name a to the cell with the name b by the link with the name c, which connects these two cells.
ch$[q \to p]$ denotes changing the state of M from q to the name p.
rn$[c(a, b) \to k(a, b)]$ denotes renaming of the link with the name c by giving it the new name k.
rn$[a \to b]$ denotes renaming of the cell where P is situated by changing its name a to the name b.
rn$[c(a, e); e \to b]$ denotes renaming of the cell that has the name e and is connected by a link with the name c to the cell where P is situated by changing its name e to the name b.
bd$[\star]$ denotes building a new cell.
nm$[\star \to b]$ denotes naming a new cell.
bd$[c(a, \star)]$ denotes building a link with the name c from the cell where P is situated with the name a to a new cell.
bd$[c(a, b)]$ denotes building a link with the name c from the cell where P is situated with the name a to a cell with the name b.

Note that all these operations are local and $\mathcal{R}$–local. There are five types of neighborhoods

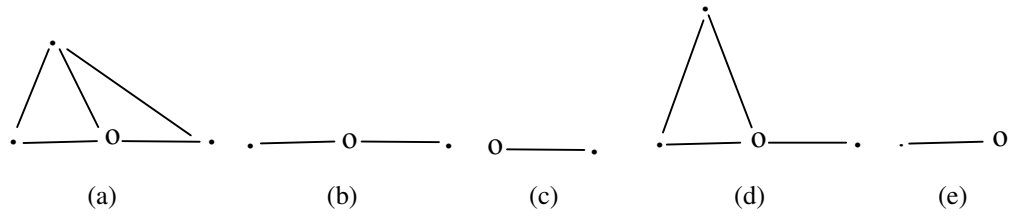

Figure 6. Types of neighborhoods of elements in processed first-order structures.

(Fig. 6) and all operations are performed only with parts (structure elements and links) of these neighborhoods.

Now we describe the rules of the machine M. In the rules, the left side describes the observation/control zone, while the right side shows the performed operation. Symbols a_j denote arbitrary elements from the alphabet Σ. We build the rules of the machine M in following way:
$[a_1; q_0] \to \mathbf{rn}[a \to N]; \mathbf{ch}[q_0 \to q_f]$
$[a_1; r(a_1, a_2); q_0] \to \mathbf{ch}[q_0 \to (q_0, a_1)]; mathbf fmv[r(a_1, a_2)]$
$[a_2; r(a_1, a_2); (q_0, a_1)] \to \mathbf{rn}[a_2 \to N]; \mathbf{ch}[(q_0, a_1) \to q_f]$ for all $a_2 \neq a_1$ $[a_1; r(a_1, a_2); (q_0, a_1)] \to \mathbf{rn}[a_2 \to T]; \mathbf{ch}[(q_0, a_1) \to q_f]$
$[a_2; r(a_1, a_2), r(a_2, a_3); (q_0, a_1)] \to \mathbf{bd}[\star]; \mathbf{nm}[\star \to \alpha]; \mathbf{bd}[db(a_2, \alpha)]; \mathbf{ch}[(q_0, a_1) \to (q_1, a_1)]; \mathbf{rn}[r(a_1, a_2) \to l(a_1, a_2)]; \mathbf{mv}[r(a_2, a_3)]$
$[a_3; r(a_2, a_3), r(a_3, a_4), db(a_2, \alpha); (q_1, a_1)] \to \mathbf{bd}[u(a_3, \alpha)]; \mathbf{mv}[r(a_3, a_4)]; \mathbf{ch}[(q_1, a_1) \to (q_0, a_1)]$
$[a_4; r(a_3, a_4), r(a_4, a_5), u(a_3, \alpha); (q_0, a_1)] \to \mathbf{bd}[u(a_4, \alpha)]; \mathbf{mv}[r(a_4, a_5)]; \mathbf{ch}[(q_0, a_1) \to (q_1, a_1)]$
$[a_k; r(a_{k-1}, a_k), u(a_{k-1}, \alpha); (q_0, a_1)] \to \mathbf{rn}[a_k \to N]; \mathbf{ch}[(q_0, a_1) \to q_f]$ for all $a_k \neq a_1$
$[a_k; r(a_{k-1}, a_k), u(a_{k-1}, \alpha); (q_1, a_1)] \to \mathbf{rn}[a_k \to N]; \mathbf{ch}[(q_1, a_1) \to q_f]$
$[a_k = a_1; r(a_{k-1}, a_k), u(a_{k-1}, \alpha); (q_0, a_1)] \to \mathbf{ch}[(q_0, a_1) \to q_1]; \mathbf{mv}[r(a_{k-1}, a_k)]$ (the case $a_k = a_1$)
$[a_{k-1}; r(a_{k-1}, a_k), u(a_{k-1}, \alpha); (q_0, a_1)] \to \mathbf{ch}[(q_0, a_1) \to (q_2, a_{k-1})]; \mathbf{rn}[r(a_{k-1}, a_k) \to l(a_{k-1}, a_k)]; \mathbf{rn}[r(a_{k-1}, a_{k-2}) \to l(a_{k-1}, a_{k-2})]; \mathbf{rn}[u(a_{k-2}, \alpha) \to de(a_{k-2}, \alpha)]; \mathbf{mv}[u(a_{k-1}, \alpha)]$
$[\alpha; de(a_{k-2}, \alpha), db(a_2, \alpha); (q_2, a_{k-1})] \to \mathbf{mv}[db(a_2, \alpha)]$
$[a_2; r(a_2, a_3), db(a_2, \alpha), l(a_1, a_2); (q_2, a_{k-1})] \to \mathbf{rn}[a_k \to N]; \mathbf{ch}[q_0 \to q_f]$ for all $a_{k-1} \neq a_2$
$[a_2; r(a_2, a_3), db(a_2, \alpha), l(a_1, a_2); (q_2, a_2)] \to \mathbf{rn}[db(a_2, \alpha) \to u(a_2, \alpha)]; \mathbf{mv}[r(a_2, a_3)]$ (the case $a_{k-1} = a_2$)
$[a_3; r(a_2, a_3), r(a_3, a_4), u(a_3, \alpha); (q_2, a_2)] \to \mathbf{ch}[(q_2, a_2) \to (q_2, a_3)]; \mathbf{rn}[r(a_2, a_3) \to l(a_2, a_3)]; \mathbf{rn}[r(a_3, a_4) \to l(a_3, a_4)]; \mathbf{rn}[u(a_4, \alpha) \to db(a_4, \alpha)]; \mathbf{mv}[u(a_3, \alpha)]$
$[\alpha; de(a_{k-2}, \alpha), db(a_4, \alpha); (q_2, a_3)] \to \mathbf{mv}[de(a_{k-2}, \alpha)]$ $[a_{k-2}; r(a_{k-2}, a_{k-3}), de(a_{k-2}, \alpha), l(a_{k-1}, a_{k-2}); (q_2, a_3)] \to \mathbf{rn}[a_{k-2} \to N]; \mathbf{ch}[(q_3, a_3) \to q_f]$ for all $a_{k-2} \neq a_3$
$[a_{k-2} = a_3; r(a_{k-2}, a_3), de(a_2, \alpha), l(a_{k-1}, a_{k-2}); (q_2, a_3)] \to \mathbf{rn}[de(a_2, \alpha) \to u(a_2, \alpha)]; \mathbf{mv}[r(a_{k-2}, a_{k-3})]$ (the case $a_{k-2} = a_3$)
$[a_{k-2}; r(a_{k-2}, a_{k-3}), de(a_{k-2}, \alpha), l(a_{k-1}, a_{k-2}); (q_2, a_3)] \to \mathbf{rn}[a_{k-2} \to T]; \mathbf{ch}[(q_3, a_3) \to q_f]$

The result of computations is defined in the following way:

- When the machine M comes to the final state and the name of the processor topos is N, then the result is negative, i.e., the input word is not symmetric.

- When the machine M comes to the final state and the name of the processor topos is T, then the result is positive, i.e., the input word is symmetric.

Using these rules the machine M checks if the input word is symmetric or not. In the process of computation, the processor P comes to each structure element (cell) not more than two times and in each cell, the processor P performs not more than five operations. Assuming, as it is done in the theory of algorithms and computation, that each operation takes one unit of time, we see that the machine M can solve the Word Symmetry Problem with the time complexity $T(n) = 10n$. Theorem is proved. At the same time, by Barzdins theorem, any deterministic Turing machine can solve the Word Symmetry Problem only with time complexity $O(n^2)$ (cf. (Trahtenbrot, 1974)). It means that computational complexity of structural machines is essentially less than computational complexity of Turing machines for some problems.

Another problem with time complexity $O(n^2)$ for Turing machines and with time complexity $O(n)$ for structural machines is inversion of a given word.

Theorem 3.3. *A structural machine with a centralized processor can simulate any simple inductive Turing machine.*

Proof. It is demonstrated how structural machine with a centralized processor can simulate any Turing machine with one tape. In the theory of algorithms and computation, it is proved that a Turing machine with one tape can simulate a Turing machine with any number of tapes. At the same time, a simple inductive Turing machine has exactly the same structure and operations (instructions) as a Turing machine with three tapes. Consequently, working in the inductive mode, a structural machine with a centralized processor can simulate any simple inductive Turing machine. □

It is also possible to simulate membrane computations (Păun & Rozenberg, 2002) with structural machines, while some classes of structural machines work as neural Turing machines (Graves *et al.*, 2014). However, these results are presented in another work of the authors.

4. Modeling slime mold computations and Physarum machines by structural machines

Physarum polycephalum belongs to the species of order *Physarales*, subclass *Myxogastromycetidae*, class *Myxomycetes*, division *Myxostelida*. It is commonly known as a true, acellular or multiheaded slime mould, see introduction in (Stephenson *et al.*, 1994). Plasmodium is a 'vegetative' phase, a single cell with a myriad of diploid nuclei. The plasmodium is visible to the naked eye. The plasmodium looks like an amorphous yellowish mass with networks of protoplasmic tubes.

The plasmodium behaves and moves as a giant amoeba forming a network of biochemical oscillators (Matsumoto *et al.*, 1986; Nakagaki *et al.*, 2000). The plasmodium's behavior is determined by external stimuli and excitation waves travelling and interacting inside the plasmodium. The plasmodium can be considered as a reaction-diffusion (Adamatzky, 2007) encapsulated in an elastic growing membrane.

When plasmodium is placed on an appropriate substrate, the plasmodium propagates, searches for sources of nutrients and follows gradients of chemo-attractants, humidity and illumination

Figure 7. Physarum forms a network of protoplasmic tubes on virtually any surface.

Figure 8. A spanning tree approximated by live Physarum. Planar data points are represented by oat flakes (dark blobs), edges of the tree by protoplasmic tubes; few duplicated tubes are considered as a single edge.

forming veins of protoplasm, or protoplasmic tubes (Fig. 7). The veins can branch, and eventually the plasmodium spans the sources of nutrients with a dynamic proximity graph, resembling, but not perfectly matching graphs from the family of k-skeletons (Kirkpatrick *et al.*, 1985).

Due to its unique features and relative ease of experimentation with, the plasmodium has become a test biological substrate for implementation of various computational tasks. The problems solved by the plasmodium include maze-solving, spanning trees and proximity graphs (Fig. 8,

calculation of efficient networks, construction of logical gates, sub-division of spatial configurations of data points, and robot control (see overview in (Adamatzky, 2016)). A computation in the plasmodium is implemented by interacting biochemical and excitation waves, redistribution of electrical charges on plasmodiums membrane and spatiotemporal dynamics of mechanical waves. Plasmodium of P. polycephalum performs complex computation by three general mechanisms: morphological adaptation of its body plan and transport network, wave propagation of information through its protoplasmic transport network, and competition and entrainment of oscillations in partial bodies relatively small fragments of plasmodium connected via protoplasmic tubes. All three mechanisms are closely associated with one another (for example, morphological adaptation is dependent on local oscillatory activity and protoplasmic flux). In (Adamatzky, 2007), it is demonstrated how to simulate Kolmogorov algorithms (Kolmogorov, 1953; Kolmogorov & Uspensky, 1958) with living slime mould in experimental laboratory conditions.

To model a Physarum machine by a structural machine, we have to interpret components of a slime mould as components of a structural machine and behaviour of the slime mould as computations of the structural machine.

A Physarum machine PM is realized by a many-headed slime mould, which is a single cell with a myriad of diploid nuclei. It is possible to treat this cell as a primitive object SM with a set of inner states. Examples of such states are “to be alive or “not to be alive. In a context of physical measurements it would be more correct to use.

In this context, we represent the object SM by the control device C_M of the structural machine M, which models the Physarum machine. The control device C_M can be assigned to be an active growing zone.

A many-headed slime mould has several active growth zones exploring concurrently the physical space around the slime mould (Fig. 9). Thus, it is natural to treat a Physarum machine as a structural machine with a distributed processor P and to interpret each active growth zone as the operation zone of a unit processor p.

As it was already demonstrated, a first-order structure $\mathbf{A} = (A, r, \mathcal{R})$, in which the set $\mathcal{R}$ consists of a single binary relation R naturally represents the structure of a living slime mould established by blobs of slime mould and active zones where structural elements (e.g., nodes) from the set A represent blobs of slime mould and active zones, while elements from the relation R (e.g., edges) represent connecting tubes A Physarum machine has two types of nodes: stationary nodes presented by sources of nutrient (oat flakes), and dynamic nodes, which are sites where two or more protoplasmic tubes originate (Adamatzky, 2007).

However, a slime mould often has a more sophisticated structure. Despite being a single cell, the slime mould can colonize substantial areas, up to hundreds of centimeters. The network of blobs, active zones and protoplasmic tubes is not uniform but forms clusters. These clusters are also connected by thick protoplasmic tubes, which represent the incidence relation that connects groups of elements from A. Therefore, we use second-order structures to model a slime mould with clusters. Thus, taking a second-order structure $\mathbf{A} = (A, r, \mathcal{R})$, in which the set $\mathcal{R}$ consists of a binary relation R, a system of binary relations $C_1, C_2, C_3, \ldots, C_n$, and a binary relation Q, we represent the structure of a living slime with clusters in the following way:

- elements from the set A represent blobs of slime mould and active zones,

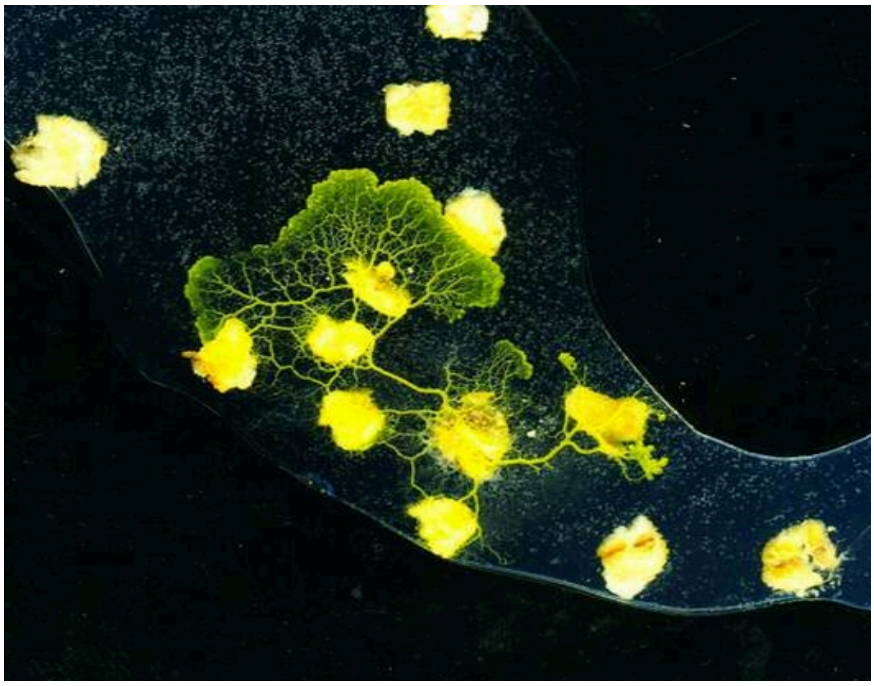

Figure 9. Physarum propagates in several directions simultaneously attracted by sources of nutrients.

- elements from the relation R represent tubes connecting blobs of the slime mould and active zones,
- each relation C_i represents one cluster of the slime mould, namely, if the cluster with the number i consists of blobs and active zones $a_1, a_2, a_3, \ldots, a_m$, then $C_i = (a_1, a_2, a_3, \ldots, a_m) \subseteq A^m$
- elements from the relation Q represent tubes connecting clusters.

This allows us to consider the sensorial space of the slime mould as the input space In_M of the machine M because the slime mould sees the world as a configuration of gradient fields.

The output space Out, which contains the output structure. The output space is the morphology of the slime mould, i.e., the configuration of growth zones, blobs occupying nutrients, and network protoplasmic tubes connecting them, is moulded by the output space Out_M of the machine M.

In a similar way, the cyto-skeletal network inside the slime mould body forms the processing space of the Physarum machine and is naturally modeled by the processing space PS_M of the structural machine M.

In slime mould, oscillatory patterns control the behaviors of the cell. In structural machines, oscillatory patterns are represented by the names of the nodes (structural elements) and links between these elements.

5. Conclusion

We determined and studied structural machines demonstrating that they can simulate Turing machines (Theorem 1) and inductive Turing machines (Theorem 3). We also proved that structural machines are more efficient than Turing machines (Theorem 2). In addition, we explained how structural machines describe and model behavior biological computers such as Physarum machine, which is based on slime mould *P. polycephalum*. This shows the big computational potential of slime mould as a biological computer. Further work can go in two directions: implementation of practical algorithms on structural machines and development of structural machines models for ultra-cellular computing based on cytoskeleton. The development of practical algorithms is necessary to allow the structural machines to 'enter the real world' and not just remain one of the many formal accomplishments of theoretical computer science. Physarum machines can solve dozens of problems from computational geometry, graph optimization and control. They also can be used as organic electronic elements (Erokhin *et al.*, 2012; Gale *et al.*, 2015; Adamatzky, 2014; Whiting *et al.*, 2015). The structural machine might form a platform for developing Physarum programming languages, compilers and interface between human operators and the slime mould (Schumann & Adamatzky, 2011; Schumann & Pancerz, 2014, 2015). The development of structural machine models of ultra-cellular computing is necessary because the behavior of the slime mould, as of most other cells, is governed by actin and tubuline networks inside the cells. Here we mention actin because it is a dominating cytoskeleton protein in P. polycephalum. Actin is a filament-forming protein forming a communication and information processing cytoskeletal network of eukaryotic cells. Actin filaments play a key role in developing synaptic structure, memory and learning of animals and humans. This is why it is important to develop abstractions of the information processing on the actin filaments. While designing experimental laboratory prototypes of computing devices from living slime mould *P. polycephalum* (Adamatzky, 2016), we found that actin networks might play a key role in distributed sensing, decentralized information processing and parallel decision making in a living cell (Adamatzky & Mayne, 2015; Mayne *et al.*, 2015). The actin-automata exhibit a wide a range of mobile and stationary patterns, which were later used to design computational models of quantum (Siccardi & Adamatzky, 2016) and Boolean (Siccardi *et al.*, 2016) gates implementable on actin fibre, as well as realization of universal computation with cyclic tag systems (Martínez *et al.*, 2017). The previously proposed model of an actin filament in a form of a finite-state machine, or automaton network, (Adamatzky & Mayne, 2015) constitutes a very special case of studied in this paper structural machines, which provide much more powerful tools for exploration of possibilities of biologically based computation. Detailed formalization of the information processing capabilities of the actin networks, including their polymerization and growths, and interaction with other intra-cellular proteins would immensely advance nano-computing and theoretical computer science making an imperative impact on development of future and emergent computing architectures. Structural machines provide means for simulation of membrane computations (Păun, 2000; Păun & Rozenberg, 2002), while some classes of structural machines work as neural networks or neural Turing machines (Graves *et al.*, 2014). Exposition of these results is given in another work of the authors. It is necessary to remark that due to their flexibility, definite classes of structural machines allow much better modeling of quantum computations than conventional models. Now there are various theoretical models of quantum computation: quantum

Turing machines (Deutsch, 1985) and quantum circuits (Feynman, 1986) correspond to discrete computing, while modular functors describe topological quantum computation (Freedman *et al.*, 2003), to mention but a few. It is interesting that while some theoretical models are recursive algorithms, which are not more powerful than Turing machines (Deutsch, 1985), other theoretical models are more powerful than Turing machines (Kieu, 2003). Structural machines provide a theoretical framework for unification of different models of quantum computation. This opportunity brings us to the open problem of modeling of quantum computation by structural machines.

References

Adamatzky, Andrew (2001). *Computing in nonlinear media and automata collectives*. CRC Press.

Adamatzky, Andrew (2007). Physarum machines: encapsulating reaction–diffusion to compute spanning tree. *Naturwissenschaften* **94**(12), 975.

Adamatzky, Andrew (2014). Slime mould electronic oscillators. *Microelectronic Engineering* **124**, 58–65.

Adamatzky, Andrew (2016). *Advances in Unconventional Computing. Volume 1: Theory*. Vol. 22. Springer.

Adamatzky, Andrew and Richard Mayne (2015). Actin automata: Phenomenology and localizations. *International Journal of Bifurcation and Chaos* **25**(02), 1550030.

Adamatzky, Andrew, Benjamin De Lacy Costello and Tetsuya Asai (2005). *Reaction-diffusion computers*. Elsevier.

Blum, Lenore, Mike Shub, Steve Smale et al. (1989). On a theory of computation and complexity over the real numbers: *np*-completeness, recursive functions and universal machines. *Bulletin (New Series) of the American Mathematical Society* **21**(1), 1–46.

Burgin, Mark (2003). From neural networks to grid automata.. *Modelling and Simulation*.

Burgin, Mark (2005). Measuring power of algorithms, programs, and automata. *Artificial Intelligence and Computer Science* pp. 1–61.

Burgin, Mark (2012). *Structural Reality*. Nova Science Publishers.

Burgin, Mark and Eugene A Bratalskii (1986). The principle of asymptotic uniformity in complex system modeling. *Operation Research and Automated Control Systems* pp. 115–122.

Burgin, Mark and Eugene Eberbach (2009*a*). On foundations of evolutionary computation: An evolutionary automata approach. *Handbook of Research on Artificial Immune Systems and Natural Computing: Applying Complex Adaptive Technologies* pp. 342–360.

Burgin, Mark and Eugene Eberbach (2009*b*). Universality for Turing machines, inductive Turing machines and evolutionary algorithms. *Fundamenta Informaticae* **91**(1), 53–77.

Calude, C and M Dinneen (2015). Unconventional computation and natural computation. In: *14th International Conference, UCNC*. Springer.

Chomsky, Noam (1956). Three models for the description of language. *IRE Transactions on information theory* **2**(3), 113–124.

Cooper, S Barry (2013). What makes a computation unconventional?. In: *Computing Nature*. pp. 255–269. Springer.

Deutsch, David (1985). Quantum theory, the Church-Turing principle and the universal quantum computer. In: *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*. Vol. 400. The Royal Society. pp. 97–117.

Elgot, Calvin C and Abraham Robinson (1964). Random-access stored-program machines, an approach to programming languages. *Journal of the ACM* **11**(4), 365–399.

Erokhin, Victor, Gerard David Howard and Andrew Adamatzky (2012). Organic memristor devices for logic elements with memory. *International Journal of Bifurcation and Chaos* **22**(11), 1250283.

Feynman, Richard P (1986). Quantum mechanical computers. *Foundations of physics* **16**(6), 507–531.

Freedman, Michael, Alexei Kitaev, Michael Larsen and Zhenghan Wang (2003). Topological quantum computation. *Bulletin of the American Mathematical Society* **40**(1), 31–38.

Gale, Ella, Andrew Adamatzky and Ben de Lacy Costello (2015). Slime mould memristors. *BioNanoScience* **5**(1), 1–8.

Graves, Alex, Greg Wayne and Ivo Danihelka (2014). Neural turing machines. *arXiv preprint arXiv:1410.5401*.

Hopcroft, John E, Rajeev Motwani and Jeffrey D Ullman (2001). Introduction to automata theory, languages, and computation. *ACM SIGACT News* **32**(1), 60–65.

Kendon, Viv, Angelika Sebald and Susan Stepney (2015). Heterotic computing: past, present and future. *Phil. Trans. R. Soc. A* **373**(2046), 20140225.

Kieu, Tien D (2003). Computing the non-computable. *Contemporary Physics* **44**(1), 51–71.

Kirkpatrick, DG, JD Radke and G Toussaint (1985). Computational geometry.

Kolmogorov, Andrei N (1953). On the concept of algorithm. *Uspekhi Mat. Nauk* **8**(4), 175–176.

Kolmogorov, Andreı N and Vladimir A Uspensky (1958). K opredeleniu algoritma. *Uspekhi Matematicheskikh Nauk [Russian Mathematical Surveys]* **13**(4), 3–28.

Kung, H and C Leiserson (1978). Science, CMUDoC: Systolic arrays for (VLSI). CMUCS.

Martínez, Genaro J, Andrew Adamatzky and Harold V McIntosh (2017). A computation in a cellular automaton collider rule 110. In: *Advances in Unconventional Computing*. pp. 391–428. Springer.

Matsumoto, Kenji, Tetsuo Ueda and Yonosuke Kobatake (1986). Propagation of phase wave in relation to tactic responses by the plasmodium of physarum polycephalum. *Journal of Theoretical Biology* **122**(3), 339–345.

Mayne, Richard, Andrew Adamatzky and Jeff Jones (2015). On the role of the plasmodial cytoskeleton in facilitating intelligent behavior in slime mold Physarum polycephalum. *Communicative & integrative biology* **8**(4), e1059007.

Nakagaki, Toshiyuki, Hiroyasu Yamada and Ágota Tóth (2000). Intelligence: Maze-solving by an amoeboid organism. *Nature* **407**(6803), 470.

Păun, Gheorghe (2000). Computing with membranes. *Journal of Computer and System Sciences* **61**(1), 108–143.

Păun, Gheorghe and Grzegorz Rozenberg (2002). A guide to membrane computing. *Theoretical Computer Science* **287**(1), 73–100.

Petri, Carl A (1962). Fundamentals of a theory of asynchronous information flow.

Schumann, Andrew and Andy Adamatzky (2011). Physarum spatial logic. *New Mathematics and Natural Computation* **7**(03), 483–498.

Schumann, Andrew and Krzysztof Pancerz (2014). Towards an object-oriented programming language for Physarum polycephalum computing: A Petri net model approach. *Fundamenta Informaticae* **133**(2-3), 271–285.

Schumann, Andrew and Krzysztof Pancerz (2015). Physarumsoft-a software tool for programming Physarum machines and simulating Physarum games. In: *Computer Science and Information Systems (FedCSIS), 2015 Federated Conference on*. IEEE. pp. 607–614.

Shannon, Claude E (1941). Mathematical theory of the differential analyzer. *Studies in Applied Mathematics* **20**(1-4), 337–354.

Shepherdson, John C and Howard E Sturgis (1963). Computability of recursive functions. *Journal of the ACM (JACM)* **10**(2), 217–255.

Siccardi, Stefano and Andrew Adamatzky (2016). Quantum actin automata and three-valued logics. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* **6**(1), 53–61.

Siccardi, Stefano, Jack A Tuszynski and Andrew Adamatzky (2016). Boolean gates on actin filaments. *Physics Letters A* **380**(1), 88–97.

Stephenson, Steven L, Henry Stempen and Ian Hall (1994). *Myxomycetes: a handbook of slime molds*. Timber Press Portland, Oregon.

Stepney, Susan, Samuel L Braunstein, John A Clark, Andy Tyrrell, Andrew Adamatzky, Robert E Smith, Tom Addis,

Colin Johnson, Jonathan Timmis, Peter Welch et al. (2005). Journeys in non-classical computation i: A grand challenge for computing research. *International Journal of Parallel, Emergent and Distributed Systems* **20**(1), 5–19.

Trahtenbrot, BA (1974). Algorithms and computing automata. moscow, sovetskoye radio.

Von Neumann, John, Arthur W Burks et al. (1966). Theory of self-reproducing automata. *IEEE Transactions on Neural Networks* **5**(1), 3–14.

Weihrauch, Klaus (2000). Introduction to computable analysis. texts in theoretical computer science.

Whiting, James GH, Ben PJ de Lacy Costello and Andrew Adamatzky (2015). Transfer function of protoplasmic tubes of Physarum polycephalum. *Biosystems* **128**, 48–51.