Structural machines and slime mould computation

Mark Burgin[a] and Andy Adamatzky[b]

[a]Department of Mathematics, UCLA, Los Angeles, CA, USA; [b]Unconventional Computing Centre and Department of Computer Science, UWE, Bristol, UK

ABSTRACT

A Physarum machine is a programmable amorphous biological computer experimentally implemented in the vegetative state of true slime mould *Physarum polycephalum*. It comprises an amorphous yellowish mass with networks of protoplasmic veins, programmed by spatial configurations of attracting and repelling gradients. The goal of this paper to advance formalism of Physarum machines providing theoretical tools for exploration of possibilities of these machines and extension of their applications. To achieve this goal, we introduce structural machines and study their properties.

ARTICLE HISTORY

Received 17 January 2016

Accepted 29 January 2017

KEYWORDS

Structural machine; slime mould; Physarum machine; unconventional computing; bio-inspired computing; computing automaton; computational efficiency

1. Introduction

Research in unconventional, or nature-inspired, computing aims to uncover novel principles of efficient information processing and computation in physical, chemical and biological systems, to develop novel non-standard algorithms and computing architectures, and also to implement conventional algorithms in non-silicon, or wet, substrates. Despite the profound potential offered by unconventional computing, only a handful of experimental prototypes of chemical and biological computation are reported so far: gas-discharge analogue path finders (Reyes 2002); maze-solving micro-fluidic circuits (Fuesterman et al. 2003); geometrically constrained universal chemical computers (Motoike and Yoshikawa 2003; Gorecki, Gorecka, and Igarashi 2009); specialized and universal chemical reaction–diffusion processors (Adamatzky, Costello, and Asai 2005); universal extended analogue computers (Mills 2008); maze-solving chemo-tactic droplets (Lagzi et al. 2010); enzyme-based logical circuits (Privman et al. 2009; Katz and Privman 2010); spatially extended crystallization computers for optimization and computational geometry (Adamatzky, Costello, and Asai 2005); molecular logical gates and circuits (Stojanovic, Mitchell, and Stefanovic 2002).

A weak representation of laboratory experiments in the field of unconventional computers could be explained by technical difficulties, costs of prototyping of novel computing substrates and also psychological barriers. At the same time, physicists found that it is possible to substitute physical experiments by theoretical modelling and computer simulation. Existence of adequate and efficient models of physical systems and processes is the necessary condition for the relevant theoretical modelling and computer simulation. That is why, following the example of physicists, here we construct an efficient theoretical model for biological and chemical computers – structural machines, which encompass contemporary models of computations providing, at the same time, powerful means for exploration of biological and chemical computations by theoretical tools and computer simulation.

The paper is organized as follows. In the second section, we describe Physarum machines. In the third section, we describe structural machines and study their properties. In the fourth section, we show how structural machines model Physarum machines. Discussions of the obtained results and future directions for research are in the fifth section.

2. Slime mould and Physarum 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 multi-headed slime mould. 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 (Figure 1).

The plasmodium behaves and moves as a giant amoeba. It feeds on bacteria, spores and other microbial creatures and micro-particles (Stephenson and Stempen 2000). When foraging for its food the plasmodium propagates towards sources of food particles, surrounds them, secretes enzymes and digests the food. Typically, the plasmodium forms a network of protoplasmic tubes connecting the masses of protoplasm at the food sources, which has been shown to be efficient in terms of network length and resilience (Nakagaki et al. 2004). When several sources of nutrients are scattered in the plasmodium's range, the plasmodium forms a network of protoplasmic tubes connecting the masses of protoplasm at the food sources (Figure 1). The plasmodium is a network of biochemical oscillators (Matsumoto, Ueda, and Kobatake 1988; Nakagaki et al. 1999). Waves of excitation or contraction originate from several sources, e.g. induced by external stimuli and perturbations. The waves travel along the plasmodium and interact one with another in collisions. The oscillatory cytoplasm of the plasmodium is a spatially extended non-linear excitable medium. Growing

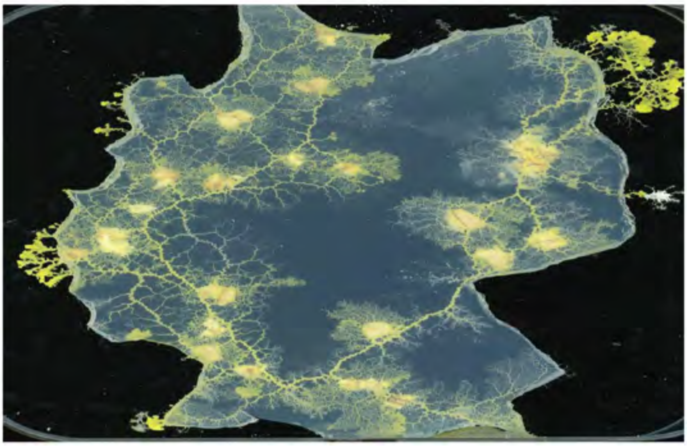

Figure 1. Slime mould Physarum polycephalum on an agar plate.
Notes: Oat flakes are seen as solid masses. They are spanned by protoplasmic networks.

and feeding plasmodium exhibits characteristic rhythmic contractions with articulated sources. The contraction waves are associated with waves of electrical potential change. The plasmodium's behaviour is determined by external stimuli and excitation waves travelling and interacting inside the plasmodium (Nakagaki et al. 1999). The plasmodium can be considered as a reaction–diffusion (Adamatzky 2007a) or an excitable (Achenbach and Weisenseel 1981) medium encapsulated in an elastic growing membrane.

The large size of the plasmodium allows the single cell to be highly amorphous. The plasmodium shows synchronous oscillation of cytoplasm throughout its cell body, and oscillatory patterns control the behaviours of the cell. All the parts of the cell behave cooperatively in exploring the space, searching for nutrients and optimizing the network of streaming protoplasm. 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. When sources of nutrients are located and engulfed, the plasmodium forms veins of protoplasm or protoplasmic tubes. 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 and Radke 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, calculation of efficient networks, construction of logical gates, sub-division of spatial configurations of data points and robot control (see e.g. Nakagaki, Yamada, and Tóth 2000; Tsuda, Aono, and Gunji 2004; Tero, Kobayashi, and Nakagaki 2007; Tsuda, Zauner, and Gunji 2007; Adamatzky 2010). A computation in the plasmodium is implemented by interacting biochemical and excitation waves, redistribution of electrical charges on plasmodium's 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 2007b), it is demonstrated how to simulate Kolmogorov algorithms with living slime mould in experimental laboratory conditions. A *Kolmogorov algorithm* (Kolmogorov 1953; Kolmogorov and Uspensky 1958) is an abstract machine defined on a dynamically changing graph-based structure, which is called a *Kolmogorov complex*. A Kolmogorov algorithm determines a computational process on a Kolmogorov complex – a finite undirected connected graph with distinctly labelled nodes. The nodes are labelled in such a manner that any two closest neighbours of any node have different labels. The graph is an analogue of a storage structure. A computational process propagates on the graph activating nodes, as well as removing and adding edges. There is only one active node at any step of the development, i.e. the neighbourhood of any active node is fixed for any particular algorithm.

A program for a Kolmogorov algorithm specifies how to replace the neighbourhood of an active node with a new neighbourhood, depending on the labels of edges connected to the active node and the labels of the nodes in proximity to the active node (Blass and Gurevich 2003). In the present paper, we advance theoretical constructs based on blending advances in the slime mould computing (Adamatzky 2010, 2016), conceptual approaches to computing on graphs (Kolmogorov 1953; Kolmogorov and Uspensky 1958) and Burgin structural machines (Burgin 2012).

3. Structural machines

Structural machines of the first-order work with first-order structures.

Definition 1. A *first-order structure* is a triad of the form

$$A = (A, r, \boldsymbol{R})$$

Here,

- A is the set of elements of the structure $\boldsymbol{A}$ called *structure elements.*
- $\boldsymbol{R}$ is the set of relations in the structure $\boldsymbol{A}$
- r is the incidence relation that connects groups of elements from A with relations from $\boldsymbol{R}$
- Namely, if $\boldsymbol{R}$ is an n-ary relation from $\boldsymbol{R}$ and $a_1, a_2, a_3, \ldots, a_n$ are elements from A, then

$$r\left(R; a_1, a_2, a_3, \ldots, a_n\right) \text{ means } \left(a_1, a_2, a_3, \ldots, a_n\right) \ni \boldsymbol{R}$$

Here, we denote by $\boldsymbol{R}$ the relation with the name R.

It is important to discern 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 $\boldsymbol{A}$ on a set A has a binary relation $\boldsymbol{R}$ with the name R and in the process of computation, the machine connects two elements from A by a link assigning it to the relation $\boldsymbol{R}$. As a result, $\boldsymbol{R}$ becomes larger but preserves the same name R.

Graphs, directed graphs, labelled graphs, words, texts, tapes of Turing machines and Kolmogorov complexes are particular cases of structures that have only unary and binary relations. Note that labels (types) are unary relations.

When $\boldsymbol{R}$ consists of one binary and several unary relations, then the first-order structure is a labelled (named) graph. When $\boldsymbol{R}$ contains only binary and unary relations, then the first-order structure is a labelled (named) multigraph.

Example 1. Below is the graphical representation of a first-order structure $\boldsymbol{A} = (A, r, \boldsymbol{R})$, where $A = \{a, b, c, d, e, f, g\}$ and $\boldsymbol{R}$ consists of one binary relation P and one ternary relation Q:

It is possible to interpret elements of the relation Q in Figure 2 as two clusters of nutrients, colonized by a single slime mould (Figure 3). 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.

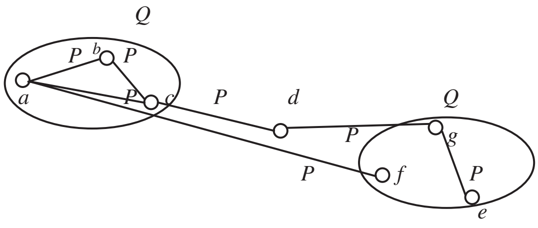

Figure 2. The graphical representation of a first-order structure.

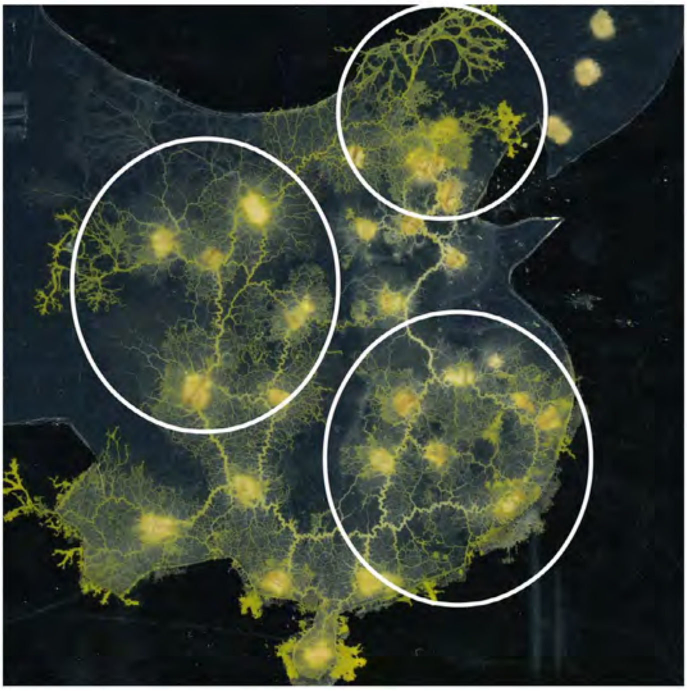

Figure 3. Tightly connected sub-components of the Physarum graph.

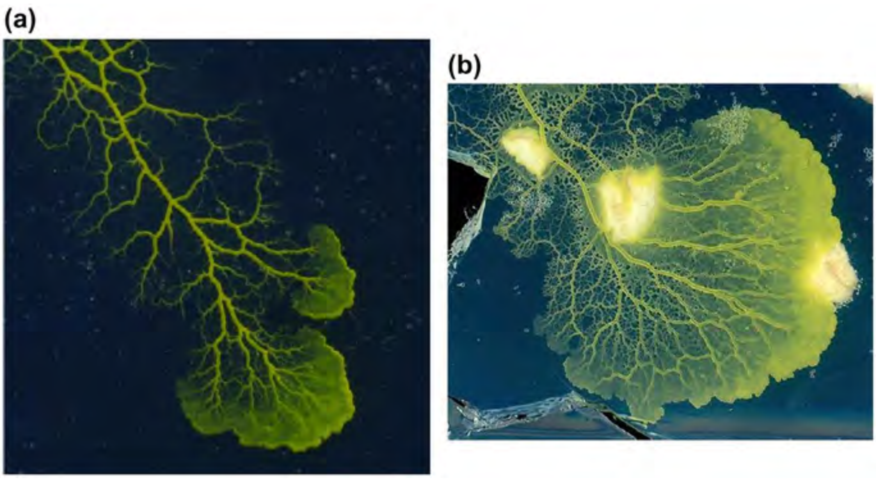

Figure 4. Active growing zone (a) and branching of two active zones (b).

When the set $\boldsymbol{R}$ consists of a single binary relation R, it is possible to represent the structure $\boldsymbol{A} = (A, r, \boldsymbol{R})$ in a living slime mould in the following way. The structure elements from the set A are represented by active growing zones or blobs of slime mould occupying source of nutrients (Figure 4): the zone, or lamellipodium, consists of an acting network which polymerization is triggered by signals from receptors distributed in the membrane. The receptors themselves respond to environmental stimuli, mainly chemo-attractants and chemo-repellents. The blobs of slime mould and active zones are connected by protoplasmic tubes. The tubes represent elements from the binary relation R in A.

On the other hand, first-order structure $\boldsymbol{A} = (A, r, \boldsymbol{R})$, in which the set $\boldsymbol{R}$ 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 (Figure 5).

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 centimetres. The network of blobs, active zones and protoplasmic tubes is not uniform but forms clusters (Figure 3). 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. (Burgin 2012). A *second-order structure* is a triad of the form

$$A = (A, r, \boldsymbol{R})$$

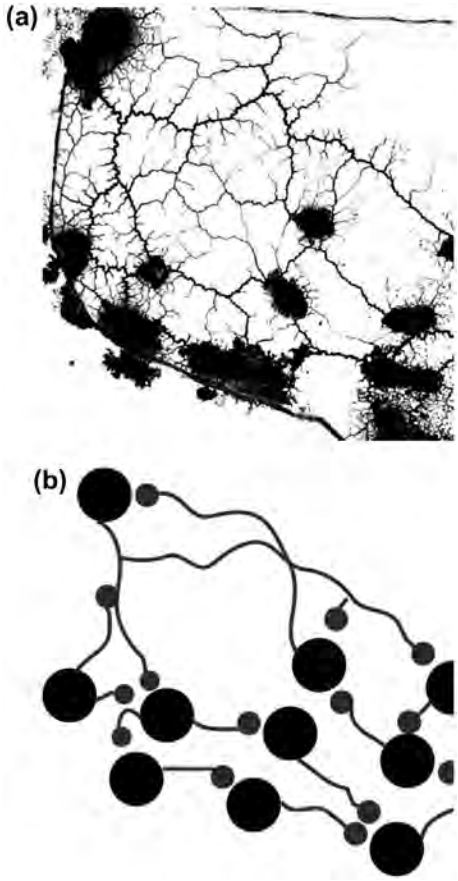

Figure 5. Scheme of structural representation of Physarum machines: (a) a snapshot of a Physarum protoplasmic network; (b) a network abstraction. Blobs of slime mould, which are occupying sources of nutrients, are solid black shapes in (a) and black discs in (b). Protoplasmic tubes are clearly visible in (a) and they shown as arcs (b). Grey discs are here to indicate influences between blobs, which is determined by direction of cytoplasm shuffling in the Physarum cell (Adamatzky and Schubert 2014).

Here,

- A is the set of elements of the structure $\boldsymbol{A}$ called *structure elements***.**
- $\boldsymbol{R}$ is the set of relations in the structure $\boldsymbol{A}$
- r is the incidence relation that connects groups of elements from A with relations from $\boldsymbol{R}$
- $\boldsymbol{R} = \boldsymbol{R}_1 \cup \boldsymbol{R}_2$
- $\boldsymbol{R}_1$ in the set A
- $\boldsymbol{R}_2$ is the set of relations in the set $\boldsymbol{R}_1$, i.e. elements from $\boldsymbol{R}_2$ are relations between relations from $\boldsymbol{R}_1$

Second-order structures are processed by structural machines of the second order.

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

Definition 3. If $\boldsymbol{R}$ is an n–ary relation from $\boldsymbol{R}$, then:

(α) the *substantial R–neighbourhood* of a structure element a in a structure $\boldsymbol{A} = (A, r, \boldsymbol{R})$ is a set of the form

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

(β) the *link R–neighbourhood* of a structure element a in a structure $\boldsymbol{A} = (A, r, \boldsymbol{R})$ is a set of the form

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

(χ) the *full R–neighbourhood* of a structure element a in a structure $\boldsymbol{A} = (A, r, \boldsymbol{R})$ is the set

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

Informally, the substantial R–neighbourhood of a structure element a consists of all structure elements connected to a by the relation $\boldsymbol{R}$.

The link *R–neighbourhood* of a structure element a consists of all elements from the relation $\boldsymbol{R}$ that contain a.

The full *R–neighbourhood* of a structure element a is the union of the substantial *R–neighbourhood* d and link *R–neighbourhood* d of a.

Examples

(1) Taking the structure presented in Figure 2, we see that $\{a, b, c, f\}$ is the substantial P–neighbourhood of the structure element a determined by the relation P from the structure in Figure 2.

(2) Taking the structure presented in Figure 2, we see that $\{a, b, c\}$ is the substantial Q–neighbourhood of the structure element a determined by the relation Q from the structure in Figure 2.

(3) Taking the structure presented in Figure 2, we see that $\{a, f\}$ is the substantial P-neighbourhood of the structure element f determined by the relation P from the structure in Figure 2.

(4) Taking the structure presented in Figure 2, we see that $\{f, g, e\}$ is the substantial Q–neighbourhood of the structure element f determined by the relation Q from the structure in Figure 2.

(5) Taking the structure presented in Figure 2, we see that $\{(a, b), (a, c), (a, f)\}$ is the link P–neighbourhood of the structure element a determined by the relation Q from the structure in Figure 2.

(6) Taking the structure presented in Figure 2, we see that $\{(a, b, c)\}$ is the link Q-neighbourhood of the structure element a determined by the relation Q from the structure in Figure 2.

Definition 3 implies the following result.

Lemma 1. The substantial R–neighbourhood of a structure element a in a structure $\boldsymbol{A} = (A, r, \boldsymbol{R})$ is uniquely defined.

Relations between relations from $\boldsymbol{R}$ imply relations between neighbourhoods.

Proposition 1. *If $R, Q \in \boldsymbol{R}$ and $R \subseteq Q$, then $O_{RS}a \subseteq O_{QS}a$, $O_{RL}a \subseteq O_{QL}a$ and $O_{RF}a \subseteq O_{QF}a$ for any structure element a.*

Neighbourhoods of elements allow us to build neighbourhoods of sets of elements.

Definition 4. If $\boldsymbol{R}$ is an n–ary relation from $\boldsymbol{R}$ and $Z \subseteq A$, then:

(α) the *substantial R–neighbourhood* of the set Z in a structure $\boldsymbol{A} = (A, r, \boldsymbol{R})$ is the set

$$O_{RS}X = \bigcup_{a \in X} O_{RS}a$$

(β) the *link R–neighbourhood* of a structure element a in a structure $\boldsymbol{A} = (A, r, \boldsymbol{R})$ is a set of the form

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

(χ) the *full R–neighbourhood* of a structure element a in a structure $\boldsymbol{A} = (A, r, \boldsymbol{R})$ is the set

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

Lemma 1 implies the following result.

Lemma 2. The substantial (link or full) R–neighbourhood of any set Z of structure elements is uniquely defined.

Proposition 1 implies the following result.

Proposition 2. *If $R, Q \in \boldsymbol{R}$ and $R \subseteq Q$, then $O_{RS}Z \subseteq O_{QS}Z$, $O_{RL}Z \subseteq O_{QL}Z$ and $O_{RF}Z \subseteq O_{QF}Z$ for any set Z of structure elements.*

Substantial neighbourhood 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 (Čech 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 (Čech 1966):

(1) $cl(\emptyset) = \emptyset$
(2) $Z \subseteq cl(Z)$ for any $Z \subseteq X$
(3) $cl(Z \cup Y) \subseteq cl(Z) \cup cl(Y)$ for any $Z, Y \subseteq X$
(4) $Y \subseteq Z$ implies $cl(Y) \subseteq cl(Z)$ for any $Z, Y \subseteq X$

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

Proposition 3. Substantial R–neighbourhoods 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-neighbourhood $O_{RS}X$ 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,

$$\begin{aligned} cl(Z \cup Y) &= O_{RS}(Z \cup Y) \\ &= \bigcup_{a \in Z \cup Y} O_{RS}a = \left(\bigcup_{a \in Z} O_{RS}a\right) \cup \left(\bigcup_{a \in Y} O_{RS}a\right) \\ &= O_{RS}Z \cup O_{RS}Y = cl(Z) \cup cl(Y) \end{aligned}$$

This gives us Axiom 3.

Axiom 4 is implied by Proposition 2.

Proposition 3 is proved.

Definition 5. (a) the *substantial* $\boldsymbol{R}$–neighbourhood of the set Z in a structure $\boldsymbol{A} = (A, r, \boldsymbol{R})$ is the set

$$O_{\boldsymbol{RS}}a = \bigcup_{R} \in \boldsymbol{R} O_{RS}a$$

(b) the *link* $\boldsymbol{R}$*–neighbourhood* of a structure element a in a structure $\boldsymbol{A} = (A, r, \boldsymbol{R})$ is a set of the form

$$O_{\boldsymbol{RL}}a = \bigcup_{R \in \boldsymbol{R}} O_{RL}a$$

(c) the *full* $\boldsymbol{R}$*–neighbourhood* of a structure element a in a structure $\boldsymbol{A} = (A, r, \boldsymbol{R})$ is the set

$$O_{\boldsymbol{RF}}a = \bigcup_{R \in \boldsymbol{R}} O_{RF}a$$

Definition 4 implies the following result.

Lemma 3. The substantial (link or full) $\boldsymbol{R}$–neighbourhood of a structure element a in a structure $\boldsymbol{A} = (A, r, \boldsymbol{R})$ is uniquely defined.

Proposition 3 and Definition 4 imply the following result.

Corollary 1. Substantial $\boldsymbol{R}$–neighbourhoods define a pretopology in the set A.

Definition 6. If $\boldsymbol{R}$ is an n-ary relation from $\boldsymbol{R}$, then the substantial R–neighbourhood of a structure element a in a structure $\boldsymbol{A} = (A, r, \boldsymbol{R})$ 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, …, n, we have

$$(a_1, a_2, a_3, \ldots, a_n) \in \boldsymbol{R} \text{ if and only if} \left(a_{i_1}, a_{i_2}, a_{i_3}, \ldots, a_{in}\right) \in \boldsymbol{R}$$

Lemma 4. If the substantial R–neighbourhood $O_{RS}a$ of a structure element a is symmetric, then a structure element b belongs to the R–neighbourhood $O_{RS}a$ if and only if a belongs to the substantial R–neighbourhood $O_{RS}b$ of b.

Definition 7. If $\boldsymbol{R}$ is an n-ary relation from $\mathbf{R}$, then the R–neighbourhood of a structure element a in a structure $\boldsymbol{A} = (A, r, \mathbf{R})$ 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 $(a_1, a_2, a_3, \ldots, a_n) \in \boldsymbol{R}$ and $(d_1, \ldots, a_i, \ldots, d_n) \in \boldsymbol{R}$, then there are elements $b_1, b_2, b_3, \ldots, b_n$ from A such that $(b_1, \ldots, a_i, \ldots, d_j, \ldots, b_n) \in \boldsymbol{R}$

Proposition 4. If a relation $\boldsymbol{R} \in \mathbf{R}$ is symmetric and transitive, then substantial R–neighbourhoods of two structure elements either coincide or do not intersect.

Corollary 2. If all relations in $\mathbf{R}$ are symmetric and transitive, then substantial $\mathbf{R}$–neighbourhoods of two structure elements either coincide or do not intersect.

Corollary 3. If $\mathbf{R}$ consists of one symmetric binary relation R, i.e. $\boldsymbol{A}$ is a graph and R is also transitive, then any substantial $\mathbf{R}$–neighbourhood (R–neighbourhood) is a complete graph

Proposition 5. If a relation $\boldsymbol{R} \in \mathbf{R}$ is symmetric and transitive, then the system of substantial R–neighbourhoods forms a base of a topology in A.

Corollary 4. If all relations in $\mathbf{R}$ are symmetric and transitive, then the system of substantial $\mathbf{R}$–neighbourhoods forms a base of a topology in A.

Definition 8. The *type* $\mathrm{T}(\boldsymbol{A})$ of a first-order structure $\boldsymbol{A} = (A, r, \mathbf{R})$ is the set $\{(R, \alpha(R)); \boldsymbol{R} \in \mathbf{R}\}$ of pairs $(R, \alpha(R))$ where $\alpha(R)$ is the arity of the relation $\boldsymbol{R}$ with the name R.

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

There are two forms functional spaces Sp_M :

- Sp_M is the set of all structures that can be processed by the structural machine M and is called a *categorical functional space*
- Sp_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 two basic types of processors:

- A *localized processor* is a single abstract device
- A *distributed processor* consists of a system of *unit processors* or *processor units*

In turn, there are two 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
- A *heterogeneous distributed processor* consists of a system of different unit processors

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

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 neighbourhoods (Figure 6). This makes it possible to consider a localized processor as a special type of a distributed processor with one unit processor.

An 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.

An example of a homogeneous distributed processor is the system of all heads of a Turing machine with several heads. It is possible to perceive a Physarum machine as multiprocessor structural machine because typically there are several active growth zones exploring concurrently the physical space around the slime mould.

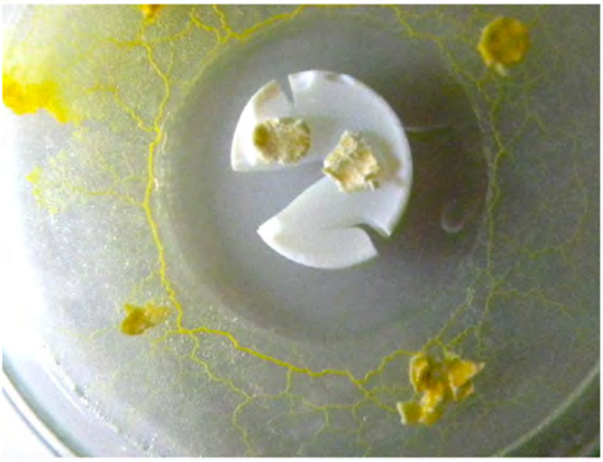

Figure 6. A ring like architecture of Physarum machine.
Notes: Large yellow blobs of the slime mould colonize irregularly shaped oat flakes. The Physarum blobs are elementary processors. Each elementary processor establishes the local communication interface with other elementary processors via protoplasmic tubes.

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 and Eberbach 2009).

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 and Bratalskii 1986).

There are three types of localized processors:

- Localized to one structure element (e.g. node)
- Localized to an R–neighbourhood of one structure element (e.g. node) where $\boldsymbol{R}$ is a relation from $\boldsymbol{R}$
- Localized to an $\boldsymbol{R}$–neighbourhood of one structure element (e.g. 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 9. The *topos* T_p of the 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 $T_p(t)$.

It is natural to assume the following condition.

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–neighbourhood of one structure element is a part of this neighbourhood and the topos of a unit processor localized to an $\boldsymbol{R}$–neighbourhood of one structure element is a part of that neighbourhood.

Definition 10. The *observation zone* Ob_p of the processor p is the part of the structure Sp_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 $Ob_p(t)$.

Axiom Z. It is assumed that operations performed by processors depend only on their observation zone.

Definition 11. The *operation zone* Op_p of the processor p is the part of the structure Sp_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, Motwani, and Ullman 2001).

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

Definition 12. The *transition zone* Tr_p of the 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 13. 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-neighbourhoods
- *operationally standardized* if all its operation zones have the same type, e.g. are R-neighbourhoods
- *transitionally standardized* if all its transition zones have the same type, e.g. are R-neighbourhoods
- *observationally standardized* if all its observation zones have the same type, e.g. are R-neighbourhoods

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

Lemma 5. 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 multi-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 6. A structural machine M is a finite state machine if and only if:

- Its structural space Sp_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 14. 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 7. A structural machine M is a temporally finite state machine if and only if:

- At any moment of time, its structural space Sp_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, Shub, and Smale (1989) are not temporally finite state machines.

Definition 15. 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 (Hopcroft, Motwani, and Ullman 2001). Processors of automata that perform operations of unrestricted formal grammars are mostly nonlocal (Hopcroft, Motwani, and Ullman 2001).

Definition 16. (a) An operation of a processor P is R*–local* if it is performed with elements (e.g. nodes) from the R–neighbourhood of a definite element (e.g. node) and with some (all) of their relations (a *singularly local operation*).

(b) An operation of a processor P is topologically R*–local* if it is performed with elements (e.g. nodes) from the R–neighbourhood of a topos of P (e.g. node) and with some (all) of their relations (a *singularly local operation*).

Lemma 6. If $\boldsymbol{R}$ contains only one binary relation, a topos of a topologically uniform processor P is one structural element and an operation O of P is totally local, then O is local.

Definition 17. (a) An operation of a processor is $\boldsymbol{R}$*–local* or *totally local* if it is performed with elements (e.g. nodes) from the $\boldsymbol{R}$–neighbourhood of a definite element (e.g. node) and with some (all) of their relations.

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

Lemma 7. If a topos of a topologically uniform processor P is one structural element and an operation O of P is local, then O is totally local.

Definitions imply the following result.

Proposition 8. If R belongs to $\boldsymbol{R}$, then any R–local operation is $\boldsymbol{R}$–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 $\boldsymbol{R}$.
(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 neighbourhood 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 ***R*** that connects some structure elements with the element where the processor is situated.
(5) *Deleting* (*removing*) a relation ***R*** from ***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. Operation of deleting the element *f* from first-order structure $\boldsymbol{A} = (A, r, R)$ where (A) shows the structure before operation and (B) shows the structure after operation. Besides, the processor (processor unit) moves from the place (position) *f* to the place (position) *g* (see Figure 7). 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 3. Operation of adding the relation *P* for elements *g* and *f* in the first-order structure $\boldsymbol{A} = (A, r, R)$ where (A) shows the structure before operation and (B) shows the structure after operation (see Figure 8). 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*.

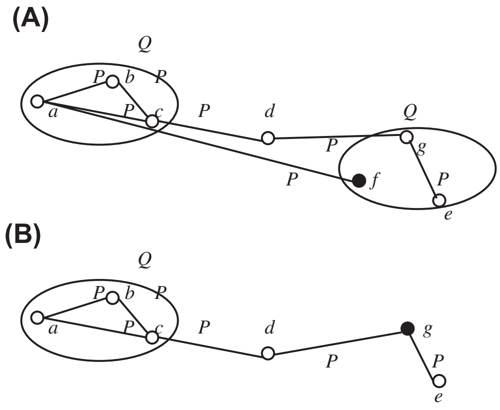

Figure 7. Deleting element. The graphical representation of an operation on first-order structures.

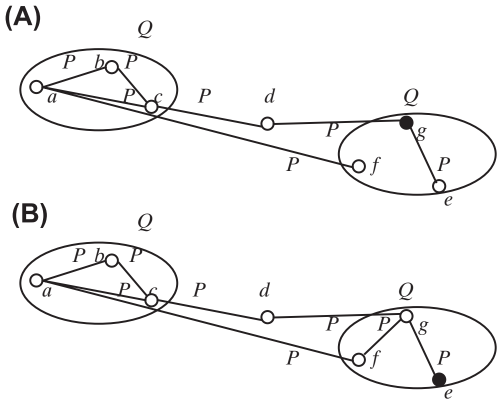

Figure 8. Adding element. The graphical representation of an operation on first-order structures.

Structural machines can simulate Turing machines, Kolmogorov algorithms (machines), storage modification machines and cellular automata (cf. Section 5).

Structural machines also can simulate processes generated by logical calculi, λ-calculus and formal grammars being able to perform operations used in various databases.

Structural machines can compute partial recursive functions and limit partial recursive functions.

Note that there are 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. As a result, artificial neural networks, finite dimensional and general machines of Blum, Shub, and Smale (1989) are particular cases of structural machines.

Thus, we can 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.

Thus, it is natural to use structural machines for a theoretical study of natural computations performed by biological, chemical and physical systems. Here, we use structural machines as abstract automata modelling functioning of such biological automata as Physarum machines based on slime mould computations.

4. Structural machine as a model of the slime mould computations

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 multi-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 (Figure 4(a)).

A multi-headed slime mould has several active growth zones exploring concurrently the physical space around the slime mould, e.g. two active growth zones are shown in Figure 4(b). Thus, it is natural to treat a Physarum machine as a structural machine with a distributed processor P (see Figures 5 and 6) 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 $\boldsymbol{A} = (A, r, \boldsymbol{R})$, in which the set $\boldsymbol{R}$ 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 2007b).

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 neighbourhoods. The network of blobs, active zones and protoplasmic tubes is not uniform but forms clusters (Figure 3). These clusters are also connected by thick protoplasmic tubes, which represent the incidence relation that connects groups of elements from A (Figure 4). Therefore, we use second-order structures to model a slime mould with clusters. Thus, taking a second-order structure $\boldsymbol{A} = (A, r, \boldsymbol{R})$, in which the set $\boldsymbol{R}$ 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,
- 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 modelled by the processing space PS_M of the structural machine M (Figure 9).

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

In its interpretation as a Physarum machine, Plasmodium of *P. Polycephalum* performs complex computation by three general mechanisms: (1) morphological adaptation of its body plan and transport network; (2) wave propagation of information through its protoplasmic transport network; and (3) competition and entrainment of oscillations in partial

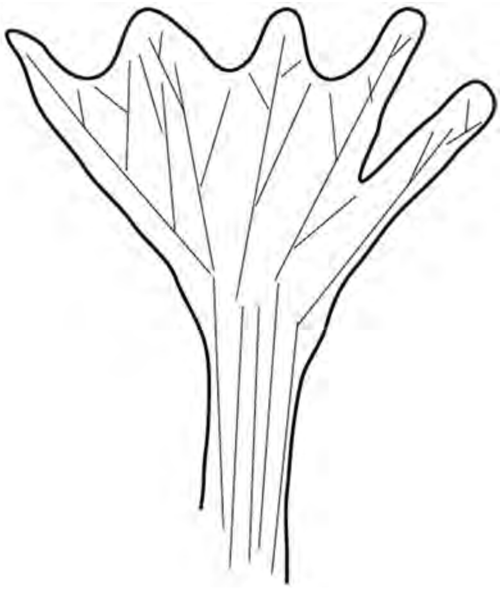

Figure 9. Fine structure of an elementary processing unit, the active growth zone or lamellipodium. The zone grows up. Cytoskeletal structure is shown as a tree-like network of actin filaments. The polymerization of the filaments is responsible for the directional growth of the zone.

bodies – relatively small fragments of plasmodium connected via protoplasmic tubes (Adamatzky 2010).

In the context of structural machines, morphological adaptation of the slime mould body plan and transport network is performed by transformation of, e.g. adding new, nodes and links in the structural space *Sp*.

A structural machine performs wave propagation of information in the slime mould protoplasmic transport network by renaming of nodes and links in the structural space *Sp*.

A structural machine simulates competition and entrainment of oscillations in partial bodies by transformation of nodes and links in the structural space *Sp*.

The basic instruction of Physarum machines are: ADD NODE, REMOVE NODE, ADD EDGE and REMOVE EDGE (Adamatzky 2007b). All these operations are modelled by substantial transforming operations of structural machines (see Section 3).

An important issue of any automaton/computer is how this machine gives the result. Different schemas of the computational result give different modes of computation by the same computing device (Burgin 2015).

To model Physarum machines, we utilize the *indication mode* of the structural machine functioning, which is based on an indication function.

Let us consider a class of structures **STR** that contains from all structures from the functional space Sp_M of a structural machine *M*.

Definition 18. A partial mapping f_{ind} : **STR** → {0, 1} is called an *indication function*.

There are different types of indication functions:

(1) An *observational indication function* is defined by an observer, who (which) determines its values.

(2) A *fitness indication function* is defined by a fitness function, which is often used in optimization processes and evolutionary computations (Burgin and Eberbach 2009).
(3) A *listings indication function* is defined by a list of possible (admissible) results.
(4) An *analytic indication function* is defined by a formula.
(5) An *algorithmic indication function* is defined by an algorithm.

Indication functions are used to select results of structural machine computations in general and Physarum machine computations, in particular.

Definition 19. A structural machine M functions in the *indication mode* with respect to an indication function f_{ind} if the result of computation is determined by applying the indication function f_{ind} to the trajectory of computation. Namely, when Q belongs to the trajectory of computation and $f_{ind}(Q) = 1$, then the structure Q is the result of computation.

For instance, the formula

$$\exists\, m\, \forall n > m\, (Q_{n,} Q_m \in \text{ Out} \Rightarrow Q_n = Q_m)$$

defines the inductive mode of computation as a particular case of the indication mode of computation with an analytic indication function.

In the Physarum machine, an observational indication function is used for selecting the result. Namely, outputs of Physarum machines are recorded optically (Adamatzky 2007b).

This shows that the Physarum machine can work in the recursive mode but the natural functioning of this biological computer is the super-recursive mode when slime mould is functioning in the continuous fashion and its outputs are recorded from time to time by observation (Burgin 2005).

5. Discussions

We formalized behaviour, and computing potential, of slime mould *P. polycephalum* as abstract structural machines demonstrating their potential. 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 (Adamatzky 2015, 2016). The structural machine might form a platform for developing Physarum programming languages, compilers and interface between human operators and the slime mould (Schumann et al. 2014; Siccardi and Adamatzky 2015; Pancerz and Schumann 2016).

The development of structural machine models of ultra-cellular computing is necessary because the behaviour 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 2015, 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 et al. 2014; Adamatzky and Mayne 2015; Mayne, Adamatzky, and Jones 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 and Adamatzky 2015) and Boolean (Siccardi, Tuszynski, and Adamatzky 2016) gates implementable on actin fibre, as well as realization of universal computation with cyclic tag systems (Martinez, Adamatzky, and Mclntosh 2015). The previously proposed model of an actin filament in a form of a finite-state machine, or automaton network, (Adamatzky and 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.

Disclosure statement

No potential conflict of interest was reported by the authors.

Notes on contributors

Mark Burgin received his MA and PhD in Mathematics from Moscow State University and doctor of Science in Logic and Philosophy from the National Academy of Sciences of Ukraine. He was a professor at the Institute of Education, Kiev; at International Solomon University, Kiev; at Kiev State University, Ukraine; and the director of the Assessment Laboratory in the Research Center of Science at the National Academy of Sciences of Ukraine. Currently, he is working at UCLA, USA. Burgin is a member of New York Academy of Sciences and an honorary professor of the Aerospace Academy of Ukraine. Burgin is a member of the Science Advisory Committee at Science of Information Institute, Washington. He was the editor-in-chief of the international journals *Integration* and *Information*, and is an associate editor of the *International Journal on Computers and their Applications* and *International Journal of Swarm Intelligence & Evolutionary Computation* and a member of the Editorial Board of the *Journal of Mathematical and Computational Science* and *TrippleC*. Burgin was a member of organizing and programme committees of more than 80 conferences. He also organized and directed several ongoing research seminars in mathematics and computer science, such as *Theoretical Computer Science* (UCLA) and *Foundations of Mathematics and Information Sciences* (National Academy of Sciences of Ukraine). Burgin is doing research, has publications and taught courses in various areas of mathematics, computer science, information sciences, system theory, artificial intelligence, software engineering, logic, psychology, education, social sciences and methodology of science. He originated such theories as the mathematical theory of technology, system theory of time, general information theory, theory of named sets, hyperprobability theory and neoclassical analysis (in mathematics) and made essential contributions to such fields as foundations of mathematics, theory of algorithms, theory of knowledge, theory of intellectual activity and complexity studies. His practical experience includes design of operating systems for supercomputers, CAD systems for electrical engineering and problem-oriented languages for such systems, databases for biological information and general expert systems, as well as mathematical modelling of databases and expert systems. Burgin has authorized and co-authorized more than 500 papers and 21 books, including "*Theory of Knowledge*" (2016), "*Structural Reality*" (2012), "*Hypernumbers and Extrafuctions*" (2012), "*Theory of Named Sets*" (2011), "*Theory of Information*" (2010), "*Measuring Power of Algorithms, Computer Programs, and Information Automata*" (2010), "*Neoclassical Analysis: Calculus Closer to the Real World*" (2008), "*Super-recursive Algorithms*" (2005), "*On the Nature and Essence of Mathematics*" (1998), "*Intellectual Components of Creativity*" (1998), "*Fundamental Structures of Knowledge and Information*" (1997), "*Introduction to the Modern Exact Methodology of Science*" (1994), *The Structure-Nominative Analysis of Theoretical Knowledge* (1992), and "*The World of Theories and Power of Mind* (1992).

Andy Adamatzky is a professor in the Department of Computer Science and the director of the Unconventional Computing Centre, University of the West of England, Bristol, UK. He does research in reaction–diffusion computing, cellular automata, physarum computing, massive parallel computation, applied mathematics, collective intelligence and robotics, bionics, computational psychology, non-linear science, novel hardware and future and emergent computation.

References

Achenbach, F., and M. H. Weisenseel. 1981. "Ionic Currents Traverse the Slime Mould Physarum." *Cell Biology International Reports* 5 (4): 375–379.

Adamatzky, A. 2007a. "Encapsulating Reaction-diffusion Computers." In *Machines, Computations, and Universality*, LNCS v. 4664, 1–11. New York: Springer-Verlag.

Adamatzky, A. 2007b. "Physarum Machine: Implementation of a Kolmogorov–Uspensky Machine on a Biological Substrate." *Parallel Processing Letters* 17 (4): 455–467.

Adamatzky, A. 2010. *Physarum Machines: Computers from Slime Mould*. London: World Scientific.

Adamatzky, A. 2015. "Thirty-nine Things to Do with Live Slime Mould." ArXiv:1512.08230 [Cs.ET].

Adamatzky, A. 2016. *Advances in Physarum machines. Sensing and computing with slime mould*. London: Springer.

Adamatzky, A., and R. Mayne. 2015. "Actin Automata: Phenomenology and Localizations." *International Journal of Bifurcation and Chaos* 25 (02): 1550030.

Adamatzky, A., and T. Schubert. 2014. "Slime Mold Microfluidic Logical Gates." *Materials Today* 17 (2): 86–91.

Adamatzky, A., B. D. Costello, and T. Asai. 2005. *Reaction-diffusion Computers*. New York: Elsevier.

Adamatzky, A., R. Armstrong, B. De Lacy Costello, Y. Deng, J. Jones, R. Mayne, T. Schubert, G. Ch Sirakoulis, and X. Zhang. 2014. "Slime Mould Analogue Models of Space Exploration and Planet Colonisation." *Journal of the British Interplanetary Society* 67: 290–304.

Blass, A., and Y. Gurevich. 2003. "Algorithms: A Quest for Absolute Definitions." *Bulletin of the EATCS*. 81: 195–225.

Blum, L., M. Shub, and S. Smale. 1989. "On a Theory of Computation and Complexity over the Real Numbers: NP- Completeness, Recursive Functions and Universal Machines." *Bulletin of the American Mathematical Society* 21 (1): 1–47.

Burgin, M. 2005. *Super-recursive Algorithms*. New York, NY: Springer.

Burgin, M. 2012. *Structural Reality*. New York: Nova Science.

Burgin, M. 2015. "Super-recursive Algorithms and Modes of Computation." *Proceedings of the 2015 European Conference on Software Architecture Workshops*, 10:1–10:5. Dubrovnik/Cavtat, Croatia.

Burgin, M., and E. A. Bratalskii. 1986 "The Principle of Asymptotic Uniformity in Complex System Modelling", In *Operation Research and Automated Control Systems*, 115–122. Kiev: Institute of Cybernetics. [in Russian].

Burgin, M., and E. Eberbach. 2009. "On Foundations of Evolutionary Computation: An Evolutionary Automata Approach." In *Handbook of Research on Artificial Immune Systems and Natural Computing*, edited by Hongwei Mo, 342–360. Hershey, PA: IGI Global.

Čech, E. 1966. *Topological Spaces*. London: John Wiley.

Fuerstman, M. J., P. Deschatelets, R. Kane, A. Schwartz, P. J. Kenis, J. M. Deutch, and G. M. Whitesides. 2003. "Solving Mazes Using Microfluidic Networks." *Langmuir* 19: 4714–4722.

Gorecki, J., J. N. Gorecka, and Y. Igarashi. 2009. "Information Processing with Structured Excitable Medium." *Natural Computing* 8 (3): 473–492.

Hopcroft, J. E., R. Motwani, and J. D. Ullman. 2001. *Introduction to Automata Theory, Languages, and Computation*. New York: Addison Wesley.

Katz, E., and V. Privman. 2010. “Enzyme-based Logic Systems for Information Processing.” *Chemical Society Reviews* 39 (5): 1835–1857.

Kirkpatrick, D. G., and J. D. Radke. 1985. *Computational Geometry*. North-Holland: Elsevier.

Kolmogorov, A. N. 1953. “On the Concept of Algorithm.” *Uspekhi Matematicheskikh Nauk* 8 (4): 175–176.

Kolmogorov, A. N., and V. A. Uspensky. 1958/1963. “On the Definition of an Algorithm.” *Uspekhi Matematicheskikh Nauk* 13 (Russian). English Translation in: AMS Translations 2 (21): 217–245.

Lagzi, I., S. Soh, P. J. Wesson, K. P. Browne, and B. A. Grzybowski. 2010. “Maze Solving by Chemotactic Droplets.” *Journal of the American Chemical Society.* 132 (4): 1198–1199.

Martinez, G. J., A. Adamatzky, and H. V. Mclntosh. 2015. “Computing with Virtual Cellular Automata Collider.” *Science and Information Conference (SAI)*, 62–68. London: IEEE.

Matsumoto, K., T. Ueda, and Y. Kobatake. 1988. “Reversal of Thermotaxis with Oscillatory Stimulation in the Plasmodium of Physarum Polycephalum.” *Journal of Theoretical Biology* 131 (2): 175–182.

Mayne, R., A. Adamatzky, and J. Jones. 2015. “On the Role of the Plasmodial Cytoskeleton in Facilitating Intelligent Behaviour in Slime Mould Physarum Polycephalum.” ArXiv Preprint ArXiv:1503.03012.

Mills, J. W. 2008. “The Nature of the Extended Analog Computer.” *Physica D: Nonlinear Phenomena* 237 (9): 1235–1256.

Motoike, I. N., and K. Yoshikawa. 2003. “Information Operations with Multiple Pulses on an Excitable Field.” *Chaos, Solitons & Fractals* 17 (2–3): 455–461.

Nakagaki, T., H. Yamada, and Á. Tóth. 2000. “Intelligence: Maze-Solving by an Amoeboid Organism.” *Nature* 407 (6803): 470–470.

Nakagaki, Toshiyuki, H. Yamada, and T. Ueda. 1999. “Modulation of Cellular Rhythm and Photoavoidance by Oscillatory Irradiation in the Physarum Plasmodium.” *Biophysical chemistry* 82 (1): 23–28.

Nakagaki, Toshiyuki, R. Kobayashi, Y. Nishiura, and T. Ueda. 2004. “Obtaining Multiple Separate Food Sources: Behavioural Intelligence in the Physarum Plasmodium.” *Proceedings of the Royal Society of London B: Biological Sciences* 271 (1554): 2305–2310.

Pancerz, K., and A. Schumann. 2016. “Some Issues on an Object-oriented Programming Language for Physarum Machines.” In edited by Radim Bris, Jaroslav Majernik, Krzysztof Pancerz, and Elena Zaitseva, *Applications of Computational Intelligence in Biomedical Technology*, 185–199. New York: Springer.

Privman, V., V. Pedrosa, D. Melnikov, M. Pita, A. Simonian, and E. Katz. 2009. “Enzymatic AND-Gate Based on Electrode-immobilized Glucose-6-Phosphate Dehydrogenase: Towards Digital Biosensors and Biochemical Logic Systems with Low Noise.” *Biosensors and Bioelectronics* 25 (4): 695–701.

Reyes, D. R., M. M. Ghanem, G. M. Whitesides, and A. Manz. 2002. “Glow Discharge in Microfluidic Chips for Visible Analog Computing.” *Lab on a Chip* 2: 113–116.

Schumann, A., and A. Adamatzky. 2015. “Physarum Polycephalum Diagrams for Syllogistic Systems.” *Journal of Logics* 2 (1): 35.

Schumann, A., K. Pancerz, A. Adamatzky, and M. Grube. 2014. “Bio-inspired Game Theory: The Case of Physarum Polycephalum.” *Proceedings of the 8th International Conference on Bioinspired Information and Communications Technologies*, 9–16. Lodz: ICST (Institute for Computer Sciences, Social-informatics and Telecommunications Engineering), December 1.

Siccardi, S., and A. Adamatzky. 2015. “Actin Quantum Automata: Communication and Computation in Molecular Networks.” *Nano Communication Networks* 6 (1): 15–27.

Siccardi, S., J. A. Tuszynski, and A. Adamatzky. 2016. “Boolean Gates on Actin Filaments.” *Physics Letters A* 380 (1–2): 88–97.

Stephenson, S., and H. Stempen. 2000. *Myxomycetes: A Handbook of Slime Moulds*. Portland: Timber Press.

Stojanovic, M. N., T. E. Mitchell, and D. Stefanovic. 2002. “Deoxyribozyme-based Logic Gates.” *Journal of the American Chemical Society.* 124 (14): 3555–3561.

Tero, Atsushi, R. Kobayashi, and T. Nakagaki. 2007. “A Mathematical Model for Adaptive Transport Network in Path Finding by True Slime Mold.” *Journal of theoretical biology* 244 (4): 553–564.

Tsuda, S., M. Aono, and Y. P. Gunji. 2004. “Robust and Emergent Physarum Logical-computing.” *Biosystems.* 73 (1): 45–55.

Tsuda, S., K. P. Zauner, and Y. P. Gunji. 2007. “Robot Control with Biological Cells.” *Biosystems.* 87 (2–3): 215–223.