Available online at www.sciencedirect.com

SCIENCE DIRECT®

Theoretical Computer Science 330 (2005) 101–116

Theoretical Computer Science

www.elsevier.com/locate/tcs

Tissue P systems with channel states

Rudolf Freund[a], Gheorghe Păun[b, c, *], Mario J. Pérez-Jiménez[c]

[a]*Faculty of Informatics, Vienna University of Technology, Favoritenstr. 9-11, A-1040 Vienna, Austria*
[b]*Institute of Mathematics of the Romanian Academy, P.O. Box 1-764, 014700 Bucureşti, Romania*
[c]*Research Group on Natural Computing, Department of Computer Science and Artificial Intelligence, University of Sevilla, Avda. Reina Mercedes s/n, 41012 Sevilla, Spain*

Abstract

We consider tissue-like P systems with states associated with the links (we call them *synapses*) between cells, controlling the passage of objects across the links. We investigate the computing power of such devices for the case of using—in a sequential manner—antiport rules of small weights. Systems with two cells are proved to be universal when having arbitrarily many states and minimal antiport rules, or one state and antiport rules of weight two. Also the systems with arbitrarily many cells, three states, and minimal antiport rules are universal. In contrast, the systems with one cell and any number of states and rules of any weight only compute Parikh sets of matrix languages (generated by matrix grammars without appearance checking); characterizations of Parikh images of matrix languages are obtained for such one-cell systems with antiport rules of a reduced weight.

Keywords: Matrix grammars; Membrane computing; P systems; States; Turing computability

1. Introduction

In the area of membrane computing area there are two main classes of systems: cell-like and tissue-like P systems. The former type is inspired from cell organization (and has membranes hierarchically arranged, hence, corresponding to a tree), the latter one mimics the “collaboration” of cells from tissues of various kinds (hence, corresponds to membranes placed in the nodes of an arbitrary graph). Actually, there are two sub-classes of tissue-like P systems, one using symport/antiport rules for communication between cells, and the other

---

[*] Corresponding author.

E-mail addresses: rudi@emcc.at (R. Freund), gpaun@us.es (G. Păun), marper@us.es (M.J. Pérez-Jiménez).

0304-3975/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2004.09.013

one, closer to neural net organization, having states associated with the cells, for controlling multiset rewriting rules which make evolve the multisets of objects in the cells.

In the present paper, we take a different perspective, somewhat mixing the two sub-cases of tissue-like systems: we associate states to the links between cells, and use these states in order to control the communication between cells; in its turn, the communication is done by means of symport/antiport rules. Between two cells at most one link is established (also called *synapse*). Because the states can be changed by using rules, a conflict can appear when two rules used on the same link ask for changing the state to two different new states. That is why we use the rules in a sequential manner: on each possible channel between two cells we use only one rule. At the level of the whole net of cells, the evolution is parallel (synchronous): we have to use a rule on each synapse where a rule can be used.

Considering a sequential use of rules on each link between cells is also challenging from a mathematical point of view; the maximal parallelism, usual in membrane computing, combined with the definition of successful computations as the halting ones, is a powerful tool in "programming" the work of P systems of various types (in particular, it provides a way to implement "appearance checking", as in regulated context-free grammars). In our framework, the expected loss in power induced by the sequential use of rules is compensated by the use of states.

The issue of considering states associated with the communication channels between membranes is part of a more general research topic, that of considering tissue-like P systems with a dynamic structure (dynamically changing membranes and/or links between them). Our approach can be considered as a partial answer to this general problem, as the states control the passage of objects across the links, selectively permitting the objects to pass, possibly completely inhibiting certain channels.

The power of systems as suggested above, with antiport rules of small weights used sequentially are shown to be Turing complete in the case of two cells (even with minimal antiport rules, if "enough" states are used) and to characterize the Parikh images of languages generated by matrix grammars without appearance checking in the case of one cell (no matter how many states and no matter how general the rules are that are used).

The case of the parallel use of rules (in a step we can use simultaneously all rules which pass from a given state to a unique next state)—as well as other related problems—remain to be investigated.

2. Tissue-like P systems with channel states

The reader is supposed to be familiar with basic elements of membrane computing, e.g., from [12]; rather useful is the comprehensive information that can be found in the web page `http://psystems.disco.unimib.it`. For the basic elements of formal language theory needed in the following, we refer to any monograph in this area, in particular, to [14] (we just mention that V^* is the free monoid generated by the alphabet V under the operation of concatenation and the empty string, denoted by λ, as identity; by *RE* we denote the family of recursively enumerable languages, and by *CF* the family of context-free languages; by $\Psi_T(L)$ we denote the Parikh image of the language $L \subseteq T^*$, and by *PsFL* we denote the set of Parikh images of languages from a given family *FL*).

Tissue-like P systems were introduced in [10]. Here we deal with the following type of systems:

A tissue-like P system (of degree $m \geqslant 1$) *with channel states* is a construct

$$\Pi = (O, T, K, w_1, \dots, w_m, E, syn, (s_{(i,j)})_{(i,j)\in syn}, (R_{(i,j)})_{(i,j)\in syn}, i_o),$$

where O is the alphabet of *objects*, $T \subseteq O$ is the alphabet of *terminal* objects, K is the alphabet of *states* (not necessarily disjoint of O), $w_1, \dots, w_m$ are strings over O representing the initial multiset of objects present in the cells of the system (it is assumed that we have m cells, labelled with $1, 2, \dots, m$), $E \subseteq O$ is the set of objects present in arbitrarily many copies in the environment, $syn \subseteq \{(i, j) \mid i, j \in \{0, 1, 2, \dots, m\}, i \neq j\}$ is the set of links between cells (we call them *synapses*; 0 indicates the environment) such that for $i, j \in \{0, 1, \dots, m\}$ at most one of $(i, j), (j, i)$ is present in *syn*, $s_{(i,j)}$ is the *initial state* of the synapse $(i, j) \in syn$, $R_{(i,j)}$ is a finite set of rules of the form $(s, x/y, s')$, for some $s, s' \in K$ and $x, y \in O^*$, associated with the synapse $(i, j) \in syn$, and, finally, $i_o \in \{1, 2, \dots, m\}$ is the *output* cell.

We note the important restriction that there is at most one synapse between two given cells, and the synapse is given as an ordered pair (i, j), with which a state from K is associated. This does not restrict the communication between the two cells (or between a cell and the environment), because we here work with antiport rules, specifying simultaneous movements of objects in the two directions of a synapse.

A rule of the form $(s, x/y, s') \in R_{(i,j)}$ is interpreted as an antiport rule for the ordered pair (i, j) of cells, acting only if the synapse (i, j) has the state s; the application of the rule means moving the objects specified by x from cell i (from the environment, if $i = 0$) to cell j, at the same time with the move of the objects specified by y in the opposite direction, as well as the change of the state of the synapse from s to s'. (The rules with one of x, y being empty are, in fact, symport rules, but we do not explicitly consider this distinction here, as it is not relevant for what follows.) The objects from E are never exhausted, irrespective how many copies of each of them are brought into the system, arbitrarily many copies remain available in the environment.

The computation starts with the multisets specified by $w_1, \dots, w_m$ in the m cells; in each time unit, a rule is used on each synapse for which a rule can be used (if no rule is applicable for a synapse, then no object passes over it and its state remains unchanged). Therefore, the use of rules is sequential at the level of each synapse, but it is parallel at the level of the system: all synapses which can use a rule must do it (the system is synchronously evolving). The computation is successful if and only if it halts and the result of a halting computation is the vector which describes the multiplicity of objects from T present in cell i_o in the halting configuration (the objects from $O - T$ are ignored when considering the result). The set of all vectors computed in this way by the system Π is denoted by $Ps(\Pi)$. The family of sets $Ps(\Pi)$ of vectors computed as above by systems with at most m cells, using at most k states, and rules $(s, x/y, s')$ with $|x| \leqslant i$, $|y| \leqslant i$ is denoted by $PsOtp_m(states_k, anti_i)$. When one of the parameters m, k, i is not bounded, it is replaced by $*$.

Before investigating the computing power of the devices introduced above, let us illustrate their work by an example:

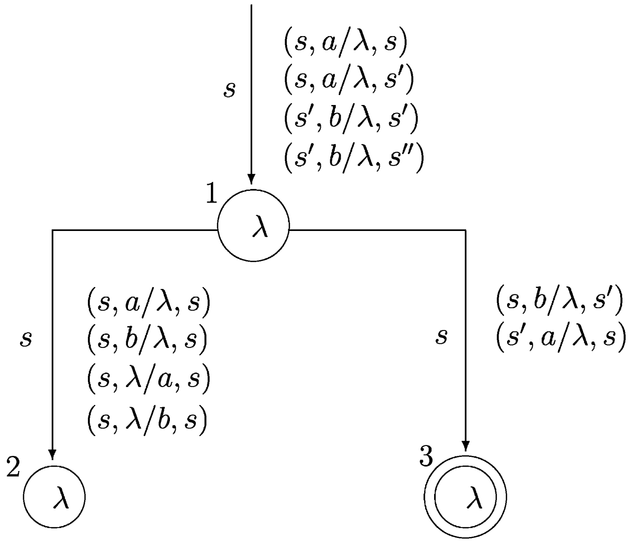

Fig. 1. The system Π_1 (rules and initial configuration).

Example 1. Formally, we consider the following tissue P system with channel states of degree 3:

$$\begin{aligned}
\Pi_1 &= (O, T, K, w_1, w_2, w_3, E, syn, (s_{(i,j)})_{(i,j)\in syn}, (R_{(i,j)})_{(i,j)\in syn}, i_o),\\
O &= \{a, b\},\\
T &= \{a, b\},\\
K &= \{s, s', s''\},\\
w_i &= \lambda, \quad \text{for all} \ \ i \in \{1, 2, 3\},\\
E &= O,\\
syn &= \{(0, 1), (1, 2), (1, 3)\},\\
R_{(0,1)} &= \{(s, a/\lambda, s),\ (s, a/\lambda, s'),\ (s', b/\lambda, s'),\ (s', b/\lambda, s'')\},\\
R_{(1,2)} &= \{(s, a/\lambda, s),\ (s, b/\lambda, s),\ (s, \lambda/a, s),\ (s, \lambda/b, s)\},\\
R_{(1,3)} &= \{(s, b/\lambda, s'),\ (s', a/\lambda, s)\},\\
i_o &= 3.
\end{aligned}$$

The system is pictorially given in Fig. 1, with the synapses represented by arrows, having associated the initial states and the rules from the respective sets (the directionality of the arrows thus specifies the way the rules are applied); each cell has the initial multiset of objects inside and the label outside; the output cell, that one with label 3, is indicated by having it doubly encircled.

The functioning of the system Π_1 is rather clear: in state s, cell 1 brings inside $n \geqslant 0$ copies of object a, then the synapse (0, 1) changes the state to s' when one further a is brought in; in state s' we bring a number $m \geqslant 0$ of copies of object b into cell 1; the process is finished only by passing to state s'', hence, at least one copy of b is introduced. Any copy of a and b can oscillate forever between cells 1 and 2, hence, the computation can stop only if all objects are moved to cell 3, the output one. The channel from cells 1 to 3 can be "opened" only by a copy of b, which changes the state of this synapse to s'; in the presence of s', a copy of a is moved from cells 1 to 3 and the state returns to s. Consequently, we can stop if and only if either the numbers of a and b introduced in cell 1 are equal, or the number of copies of b is larger by 1 than the number of copies of a. That is, $Ps(\Pi_1) = \{(n,n) \mid n \geqslant 1\} \cup \{(n, n+1) \mid n \geqslant 1\}$.

It is worth noting that the system uses only rules where one object passes through a synapse, in either direction.

3. Technical prerequisites

In the proofs of the next section we will use register machines and matrix grammars (without appearance checking), that is why we introduce these computing devices here.

In what concerns register machines, we refer to [11] for original definitions, and to [5,6] for definitions like that we use in this paper.

A (*non-deterministic*) *register machine* is a construct $M = (n, R, l_0, l_h)$, where n is the number of registers, R is a finite set of instructions injectively labelled with elements from a given set $lab(M)$, l_0 is the initial/start label, and l_h is the final label.

The instructions are of the following forms:

- $l_1 : (add(r), l_2, l_3)$,

  Add 1 to the contents of register r and proceed to one of the instructions (labelled with) l_2 and l_3. (We say that we have an ADD instruction.)
- $l_1 : (sub(r), l_2, l_3)$,

  If register r is not empty, then subtract 1 from its contents and go to instruction l_2, otherwise proceed to instruction l_3. (We say that we have a SUB instruction.)
- $l_h : halt$,

  Stop the machine. The final label l_h is only assigned to this instruction.

A register machine M is said to generate a vector $(s_1, \ldots, s_k)$ of natural numbers if, starting with the instruction with label l_0 and all registers containing the number 0, the machine stops (it reaches the instruction $l_h : halt$) with the first k registers containing the numbers $s_1, \ldots, s_k$.

The register machines are known to be computationally universal, equal in power to (non-deterministic) Turing machines: they generate exactly the sets of vectors of natural numbers which can be generated by Turing machines, that is, the family *PsRE*.

Without loss of generality, in the proofs of the following section we will assume that in each ADD instruction $l_1 : (add(r), l_2, l_3)$ and in each SUB instruction $l_1 : (sub(r), l_2, l_3)$ the labels l_1, l_2, l_3 are mutually distinct: For instance, to achieve this goal, we replace each Add instruction $l_1 : (add(r), l_2, l_3)$ by the instruction $l_1 : (add(r), l_2', l_3'')$ and each SUB instruction $l_1 : (sub(r), l_2, l_3)$ by the instruction $l_1 : (sub(r), l_2', l_3'')$, respectively, and in both cases we add the instructions $l_2' : (add(n+1), l_2''', l_2^{iv})$, $l_2''' : (sub(n+1), l_2, l_2')$, $l_2^{iv} : (sub(n+1), l_2, l_2')$, $l_3'' : (add(n+1), l_3^{v}, l_3^{vi})$, $l_3^{v} : (sub(n+1), l_3, l_3'')$, $l_3^{vi} : (sub(n+1), l_3, l_3'')$, where $n+1$ is a new register (this can be the same for all ADD and all SUB instructions we start from), and all primed labels are distinct and different from the initial labels.

In the following, we also use *matrix grammars*. For details, we refer to [3] and to the chapter of [14] devoted to regulated rewriting; here we only introduce the particular case we need below.

A *matrix grammar* (*without appearance checking*) is a construct $G = (N, T, S, M)$, where N, T are disjoint alphabets, $S \in N$, and M is a finite set of ordered sequences of the form $(A_1 \rightarrow x_1, \ldots, A_n \rightarrow x_n)$, $n \geqslant 1$, of context-free rules over $N \cup T$ (with $A_i \in N, x_i \in (N \cup T)^*$, in all cases); N is the non-terminal alphabet, T is the terminal alphabet, S is the axiom, while the elements of M are called matrices.

For $w, z \in (N \cup T)^*$ we write $w \Longrightarrow z$ if there are a matrix $(A_1 \rightarrow x_1, \ldots, A_n \rightarrow x_n)$ in M and strings $w_i \in (N \cup T)^*$, $1 \leqslant i \leqslant n+1$, such that $w = w_1$, $z = w_{n+1}$, and, for all $1 \leqslant i \leqslant n$, $w_i = w_i' A_i w_i''$, $w_{i+1} = w_i' x_i w_i''$, for some $w_i', w_i'' \in (N \cup T)^*$. The language generated by G is defined by $L(G) = \{w \in T^* \mid S \Longrightarrow^* w\}$.

By *MAT* we denote the family of languages generated by matrix grammars. It is known that *PsCF* $\subset$ *PsMAT* $\subset$ *PsRE* (for instance, *PsMAT* contains non-semilinear sets of vectors, which is not the case with *PsCF*; on the other hand, the one-dimensional vectors from *PsMAT* are semilinear, while *PsRE* contains non-semilinear sets of numbers).

The power of matrix grammars is not decreased if we only work with matrix grammars in the *binary normal form* (see [3]). A matrix grammar $G = (N, T, S, M)$ is in the binary normal form if it has $N = N_1 \cup N_2 \cup \{S\}$, where these three sets are mutually disjoint, and each matrix in M is of one of the following forms:
(1) $(S \rightarrow XA)$, with $X \in N_1$, $A \in N_2$,
(2) $(X \rightarrow Y, A \rightarrow x)$, with $X, Y \in N_1$, $A \in N_2$, $x \in (N_2 \cup T)^*$, $|x| \leqslant 2$,
(3) $(X \rightarrow \lambda, A \rightarrow x)$, with $X \in N_1$, $A \in N_2$, and $x \in T^*$, $|x| \leqslant 2$.
Moreover, there is only one matrix of type 1 and a matrix of type 3 is used only once, in the last step of a derivation.

In the following we shall use a slightly different variant of this binary normal form by adding one new non-terminal *f* indicating its unique final "state", i.e., from a matrix grammar $G = (N, T, S, M)$ in the binary normal form as above we construct the matrix grammar $G_f = (N \cup \{f\}, T, S, M_f)$ in *f-binary normal form* with

$$\begin{aligned}
M_f = (M - \{&(X \rightarrow \lambda, A \rightarrow x) \mid (X \rightarrow \lambda, A \rightarrow x) \in M, \\
&X \in N_1, A \in N_2, x \in T^*\}) \\
&\cup \{(X \rightarrow f, A \rightarrow x) \mid (X \rightarrow \lambda, A \rightarrow x) \in M, \\
&X \in N_1, A \in N_2, x \in T^*\}) \\
&\cup \{(f \rightarrow \lambda)\}.
\end{aligned}$$

Hence, M_f contains rules of the following forms:
(1) $(S \rightarrow XA)$, with $X \in N_1$, $A \in N_2$,
(2) $(X \rightarrow Y, A \rightarrow x)$, with $X, Y \in N_1$, $A \in N_2$, $x \in (N_2 \cup T)^*$, $|x| \leqslant 2$,
(3) $(X \rightarrow f, A \rightarrow x)$, with $X \in N_1$, $A \in N_2$, and $x \in T^*$, $|x| \leqslant 2$,
(4) $(f \rightarrow \lambda)$.

Moreover, there is only one matrix of type 1 and only one matrix of type 4, which is only used in the last step of a derivation yielding a terminal result.

It is obvious that a usual tissue-like P system (without states) can be considered as having the same state associated with all synapses, never changing. Because P systems with one membrane and using antiport rules of weight at least two are universal in the case of maximally parallel use of rules (see, e.g. [5,7,8]), it is expected that a similar result holds true also in our case. However, this does not happen: if we have only one cell, irrespective how many states and how complex rules we use, we get at most the Parikh images of matrix languages (without appearance checking). The explanation of this important difference between our results and those from [5,7,8] lies in the difference between the way the two types of systems work: sequentially here, in a maximally parallel manner in the cited papers (as we have mentioned in the Introduction, the maximal parallelism together with the halting condition for defining the successful computations provides the necessary tools for simulating the appearance checking, which is not the case for the sequential use of rules; moreover, the appearance checking is exactly the difference between *MAT* and universality—matrix grammars with appearance checking are equivalent to Turing machines). However, universality can be obtained also in our case as soon as we use at least two cells.

We start with the characterization of the Parikh images of matrix languages.

Lemma 2. $PsMAT \subseteq PsOtp_1(state_*, anti_1)$.

Proof. Let us consider a matrix grammar $G = (N_1 \cup N_2 \cup \{S, f\}, T, S, M)$ in the f-binary normal form where $(S \to X_0 A_0)$ is the initial matrix of *M*. Then we construct the tissue-like P system with channel states

$$\begin{aligned}
\Pi &= (O, T, K, A_0 Z, O, \{(0,1)\}, X_0, R_{(0,1)}, 1),\\
O &= N_2 \cup T \cup \{Z\},\\
K &= N_1 \cup \{f\} \cup \{\langle X, \alpha\rangle \mid X \in N_1 \cup \{f\}, \alpha \in N_2 \cup T\},\\
R_{(0,1)} &= \{(X, \alpha/A, Y) \mid (X \to Y, A \to \alpha) \in M,\\
&\quad X \in N_1, Y \in N_1 \cup \{f\}, A \in N_2, \alpha \in N_2 \cup T \cup \{\lambda\}\}\\
&\quad \cup \{(X, \alpha_1/A, \langle Y, \alpha_2\rangle), (\langle Y, \alpha_2\rangle, \alpha_2/\lambda, Y) \mid (X \to Y, A \to \alpha_1\alpha_2) \in M,\\
&\quad X \in N_1,\ Y \in N_1 \cup \{f\}, A \in N_2, \alpha_1, \alpha_2 \in N_2 \cup T\}\\
&\quad \cup \{(f, A/A, f) \mid A \in N_2\} \cup \{(f, \lambda/Z, f)\}\\
&\quad \cup \{(X, Z/Z, X) \mid X \in N_1\}.
\end{aligned}$$

The matrices $(X \to Y, A \to x)$ of *M* are simulated by simultaneously changing the state of the unique synapse and exchanging an internal object *A* for the multiset *x*. If *x* consists of at most one symbol, then the simulation is done in only one step. If $x = \alpha_1\alpha_2$, then the objects α_1, α_2 are brought into the system in two consecutive steps. When the state *f* is introduced, we check whether the derivation in *G* is terminal and only in the affirmative case we halt. As long as the state of the synapse (0, 1) is not *f*, the computation continues, at least by a rule of the form $(X, Z/Z, X)$ for some $X \in N_1$. The auxiliary object *Z* is sent out by means of the rule $(f, \lambda/Z, f)$ and then the computation stops. Consequently, $\Psi_T(L(G)) = Ps(\Pi)$. □

The number of states can be decreased to one if we use more powerful rules.

Lemma 3. *PsMAT* $\subseteq$ *PsOtp*$_1(state_1, anti_2)$.

Proof. Consider a matrix grammar $G = (N_1 \cup N_2 \cup \{S, f\}, T, S, M)$ in the f-binary normal form where $(S \to X_0 A_0)$ is the initial matrix of *M*; now we construct the tissue-like P system with channel states

$$\begin{aligned}
\Pi &= (O, T, \{s\}, X_0 A_0, O, \{(0,1)\}, s, R_{(0,1)}, 1),\\
O &= N_1 \cup \{f\} \cup N_2 \cup T \cup \{\langle X, \alpha\beta\rangle \mid X \in N_1 \cup \{f\}, \alpha, \beta \in N_2 \cup T\},\\
R_{(0,1)} &= \{(s, Yx/XA, s) \mid (X \to Y, A \to x) \in M\\
&\quad X \in N_1, Y \in N_1 \cup \{f\}, A \in N_2, x \in N_2 \cup T \cup \{\lambda\}\}\\
&\quad \cup \{(s, Y\langle Y, \alpha_1\alpha_2\rangle/XA, s), (s, \alpha_1\alpha_2/\langle Y, \alpha_1\alpha_2\rangle, s) \mid\\
&\quad (X \to Y, A \to \alpha_1\alpha_2) \in M,\\
&\quad X \in N_1,\ Y \in N_1 \cup \{f\}, A \in N_2, \alpha_1, \alpha_2 \in N_2 \cup T\}\\
&\quad \cup \{(s, \alpha/\alpha, s) \mid \alpha \in N_1 \cup N_2\} \cup \{(s, \lambda/f, s)\}.
\end{aligned}$$

The state plays no rôle, the matrices of *M* are simulated by the antiport rules. As long as at least one non-terminal from $N_1 \cup N_2$ is present, the computation must continue. Hence, the equality $\Psi_T(L(G)) = Ps(\Pi)$ is obvious. □

We now pass to considering the opposite inclusions, proving that one-cell systems cannot exceed the power of matrix grammars, irrespective how many states we use and how complex the rules are that we use.

Lemma 4. *PsOtp*$_1(state_*, anti_*) \subseteq$ *PsMAT*.

Proof. Let $\Pi = (O, T', K, w_1, E, \{(0,1)\}, s_0, R_{(0,1)}, 1)$ be a tissue-like P system with channel states.

Then we first construct the matrix grammar $G = (N, T, S, M)$ with

$$\begin{aligned}
N &= K \cup \{s' \mid s \in K\} \cup \{a' \mid a \in O\} \cup \{S\},\\
T &= \{s'' \mid s \in K\} \cup O
\end{aligned}$$

and the following matrices:

(1) $(S \to s_0 h(w_1))$,

(2) $(s_1 \to s_2 h(x))$, for $(s_1, x/\lambda, s_2) \in R_{(0,1)}$,

(3) $(s_1 \to s_2, y_1' \to \lambda, \ldots, y_k' \to \lambda)$, for $(s_1, \lambda/y, s_2) \in R_{(0,1)}$ and
$y = y_1 y_2 \ldots y_k, k \geqslant 1$, with $y_i \in O, 1 \leqslant i \leqslant k$,

(4) $(s_1 \to s_2, y_1' \to h(x), y_2' \to \lambda, \ldots, y_k' \to \lambda)$, for $(s_1, x/y, s_2) \in R_{(0,1)}$ and
$y = y_1 y_2 \ldots y_k, k \geqslant 1$, with $y_i \in O, 1 \leqslant i \leqslant k$,

(5) $(s \to s')$, for $s \in K$,
$(s' \to s', a' \to a)$, for $s \in K, a \in O$,
$(s' \to s'')$, for $s \in K$,

where *h* is the morphism which replaces each $a \in O$ by a'.

In the presence of non-terminals from *K*, we simulate the rules from $R_{(0,1)}$; at any moment we can introduce a primed state, in the presence of which we transform each a' for $a \in O$ into the terminal *a*; we end the derivation by replacing the primed state by a double primed version of it, which is a terminal symbol for *G*.

Now, consider the regular language

$$L = \{s_1{}'' z_1 y z_2 \mid (s_1, x/y, s_2) \in R_{(0,1)}, s_1, s_2 \in K, x, y, z_1, z_2 \in O^*, y \neq \lambda\}$$
$$\cup \{s_1{}'' z \mid (s_1, x/\lambda, s_2) \in R_{(0,1)}, s_1, s_2 \in K, x, z \in O^*\}.$$

This language contains all strings that describe configurations for which the computation in Π is not halting. Thus, the language $L' = \{s'' \mid s \in K\}O^* - L$ contains all strings which describe halting configurations. Therefore, $L(G) \cap L'$ identifies all halting configurations that were encoded in the strings of $L(G)$. Now consider the morphism g which erases the symbols s'', $s \in K$, as well as all symbols from $O - T'$. The equality $Ps(\Pi) = \Psi_{T'}(g(L(G) \cap L'))$ holds. As the family of matrix languages is closed under intersection with regular languages and morphisms (clearly, L and L' are regular), we obtain $Ps(\Pi) \in$ *PsMAT*, and this completes the proof. □

By combining the previous three lemmas, we get the following characterizations of *PsMAT*:

Theorem 5. *PsMAT* $= PsOtp_1(state_k, anti_i) = PsOtp_1(state_*, anti_j)$ *for all* $k \geqslant 1$ *and* $i \geqslant 2$ *as well as for all* $j \geqslant 1$ (*each of* k, i, j *can also be equal to* $*$).

Obviously, one-cell systems with one state and antiport rules of weight 1 can only generate finite languages. However, if at least two cells are used, then even with antiport rules of minimal weight we again get computational universality. The result is relevant both in comparison with the previous theorem (thus specifying a sharp borderline between universality and non-universality), and if we compare it with the main result of [1], where the universality (of cell-like P systems with a maximal use of symport/antiport rules of minimal weight) is obtained when using five membranes. In our case, two cells suffice, a fact which proves the power of using states.

Theorem 6. *PsRE* $= PsOtp_m(state_*, anti_i)$ *for all* $m \geqslant 2$ *and* $i \geqslant 1$.

Proof. We only prove the inclusion *PsRE* $\subseteq PsOtp_2(state_*, anti_1)$. To this aim, let us consider a register machine $M = (n, R, l_0, l_h)$ (with $lab(M) = \{g_1, \ldots, g_t\}$) generating the set of vectors $N(M) \subseteq \mathbf{N}^k$, for some $k \geqslant 1$, and construct the tissue-like P system (of degree 2)

$$\Pi = (O, T, K, \lambda, w_2, E, \{(0,1), (1,2), (0,2)\}, l_0, s, s, R_{(0,1)}, R_{(1,2)}, R_{(0,2)}, 1),$$
$$O = \{a_i \mid 1 \leqslant i \leqslant n\} \cup \{l, l', l''', l^v \mid l \in lab(M)\},$$
$$T = \{a_i \mid 1 \leqslant i \leqslant k\},$$
$$K = \{s, s'\} \cup \{l, l'', l^{iv} \mid l \in lab(M)\},$$
$$w_2 = g'_1 g'_2 \ldots g'_t,$$
$$E = O,$$

with the following sets of rules:

(1) For each ADD instruction $l_1 : (add(r), l_2, l_3)$ of M, we introduce the rules $(l_1, a_r/\lambda, l_2)$ and $(l_1, a_r/\lambda, l_3)$ in $R_{(0,1)}$.

Clearly, the instruction of the register machine is correctly simulated by Π (the current label of the synapse $(0, 1)$ is always related to the label of the current instruction from the computation of M).

(2) For each SUB instruction $l_1 : (sub(r), l_2, l_3)$ from R we introduce the rules indicated in the table below in the sets of rules of Π. The rules are given as used in the five steps necessary in Π to simulate this instruction.

Step	$R_{(0,1)}$	$R_{(1,2)}$	$R_{(0,2)}$
1	$(l_1, l_1/\lambda, l_1'')$	Nothing	Nothing
2	$(l_1'', l_1'''/\lambda, l_1^{iv})$	$(s, l_1/\lambda, l_1)$	Nothing
3	$(l_1^{iv}, l_1^{v}/l_1''', l_1^{iv})$	$(l_1, a_r/l_2', s')$ or nothing	$(s, l_2'/l_1, s)$
4	$(l_1^{iv}, \lambda/l_2', l_2)$ or nothing	$(s', l_1^{v}/\lambda, s)$ or $(l_1, l_1^{v}/l_3', s)$	Nothing
5	New instruction or $(l_1^{iv}, \lambda/l_3', l_3)$	Nothing	$(s, l_3'/l_1^{v}, s)$

Under the control of the label l_1, we bring the object l_1 into the first cell (and the state of the synapse $(0, 1)$ is changed to l_1''). In the second step, object l_1 is sent to the second cell, thus changing the label of the synapse $(1, 2)$ to l_1. Simultaneously, l_1''' is brought into the first cell (under the control of the label l_1'' of the synapse $(0, 1)$, which is changed to l_1^{iv}). Now, we can start checking whether there is any a_r in cell 1. If this is the case, then the rule $(l_1, a_r/l_2', s')$ must be used, and it sends a copy of a_r to cell 2; if no copy of a_r is present, then no rule is applied on the synapse $(1, 2)$. Simultaneously, l_1 leaves cell 2 and in exchange l_2' is brought (back) from the environment, while on the synapse $(0, 1)$ we use the rule $(l_1^{iv}, l_1^{v}/l_1''', l_1^{iv})$; its rôle is to bring the "checker" l_1^{v} into the system, leaving to cell 1 the time to send a copy of a_r to cell 2, provided that such a copy exists.

In the next step, l_1^{v} is sent to cell 2, nothing is used on the synapse $(0, 2)$, while on the synapse $(0, 1)$ we have two possibilities. If a_r was available, hence, l_2' was brought into cell 1, then this objects is sent to the environment and the label of the synapse $(0, 1)$ becomes l_2. In this way, we have completed the simulation of the SUB instruction for the case when the subtraction was possible. If no a_r was available, then we do not communicate between cell 1 and the environment.

However, the way l_1^{v} passes from cell 1 to cell 2 depends on the label of the synapse $(1, 2)$, which, in turn, depends on the fact whether or not a_r existed. If a_r was present, then the label is s', and l_1^{v} just returns the label to s, making possible a new simulation; otherwise, the label is l_1, hence, l_1^{v} is exchanged with l_3' and the label is returned to s, too.

In either case, in the next step no rule can be used on the synapse $(1, 2)$, while l_1^{v} is sent from cell 2 to the environment, in exchange with l_3'; in this way, also l_3' is available again for a possible use in a subsequent step. If a_r was not present, then in step 5 we send l_3' from cell 1 to the environment, and the label of the synapse $(0, 1)$ becomes l_3. This correctly completes the simulation of the SUB instruction for the case that the subtraction was not possible.

We should like to emphasize the important details that in cell 1 we only have copies of the objects a_j for those j representing non-zero registers in M, and that the contents of cell 2 is restored, with objects l' present for all $l \in lab(M)$ – with one further copy of one of the objects l_2', l_3' (during the simulation, we bring both of them from the environment into cell 2, although only one of them then is sent to cell 1 in order to change the label of the synapse $(0, 1)$).

(3) No rule is introduced for label l_h of synapse $(0, 1)$, hence, the work of Π will stop exactly when the work of M stops.

From the explanations given above we conclude that $N(M) = Ps(\Pi)$. □

The previous proof uses a number of states which depend on the number of labels used by the register machine simulated by our system. The number of states can even be reduced to 1 at the expense of increasing the weight of rules by one.

Theorem 7. *$PsRE = PsOtp_m(state_k, anti_i)$ for all $m \geqslant 2$, $k \geqslant 1$, and $i \geqslant 2$.*

Proof. We again consider a register machine $M = (n, R, l_0, l_h)$ (with $lab(M) = \{g_1, \ldots, g_t\}$) generating the set of vectors $N(M) \subseteq \mathbf{N}^k$, for some $k \geqslant 1$, and construct the tissue-like P system (of degree 2)

$$\begin{aligned}
\Pi &= (O, T, K, l_0, w_2, E, \{(0,1), (1,2), (0,2)\}, s, s, s, R_{(0,1)}, R_{(1,2)}, R_{(0,2)}, 1),\\
O &= \{a_i \mid 1 \leqslant i \leqslant n\} \cup \{l, l', l'', l''' \mid l \in lab(M)\} \cup \{e\},\\
T &= \{a_i \mid 1 \leqslant i \leqslant k\},\\
K &= \{s\},\\
w_2 &= e g_1 g_2 \ldots g_t,\\
E &= O,
\end{aligned}$$

with the following sets of rules:

(1) For each ADD instruction $l_1 : (add(r), l_2, l_3)$ from R, we introduce the rules
$(s, l_2 a_r / l_1, s)$ and $(s, l_3 a_r / l_1, s)$ in $R_{(0,1)}$.

(2) For each SUB instruction $l_1 : (sub(r), l_2, l_3)$ from R we introduce the rules indicated in the table below in the sets of rules of Π. The rules are given as used in the five steps necessary in Π to simulate this instruction. The states play no rôle in the computation, the SUB instructions of M are simulated by the antiport rules in a way rather similar to that from [7], but using the rules in a sequential manner and making use of having two cells (and the environment) for controlling the computation.

The label l_1 is replaced by l_1', l_1'' in the first cell. In the second step, if a copy of a_r is present, then the object l_1' is sent to the second cell together with a copy of a_r and the auxiliary object e is brought into cell 1; if no copy of a_r exists, then l_1' waits in cell 1. Simultaneously, l_1''' is brought into the first cell in exchange of l_1''. In the third step, l_1''' checks what happened in cell 1 in the previous step: if we here have e (i.e., a_r was present), then the objects l_1''', e bring the label l_2 from cell 2, thus completing the simulation of the SUB instruction for the case when the subtraction was possible. If we still have l_1' in cell 1, then the objects l_1''', l_1' bring l_3 from cell 2, thus completing the simulation of the instruction for the case when the subtraction was not possible.

In cell 2, we exchange l_1' with l_2 (which is brought in from the environment), either in step 3 (in the case when a_r was present), or in one of steps 4 and 5; in the latter case, the rule $(s, l_2/l_1', s)$ is used in alternate steps with the rule $(s, l_3/l_1''', s)$, which brings the label l_3 into the system. In this way, the contents of cell 2 is restored, hence, we can continue simulating the instructions of M.

Step	$R_{(0,1)}$	$R_{(1,2)}$	$R_{(0,2)}$
1	$(s, l_1' l_1''/l_1, s)$	Nothing	Nothing
2	$(s, l_1'''/l_1'', s)$	$(s, l_1' a_r/e, s)$	Nothing
3	Nothing	$(s, l_1''' e/l_2, s)$ or $(s, l_1''' l_1'/l_3, s)$	$(s, l_2/l_1', s)$
4	New instruction	Nothing	$(s, l_3/l_1''', s)$ or $(s, l_2/l_1', s)$
5	New instruction	New instruction	$(s, l_2/l_1', s)$ or $(s, l_3/l_1''', s)$

(3) We also introduce the rule

$(s, \lambda/l_h, s)$ in $R_{(0,1)}$,

hence, the work of Π will stop exactly when the work of M stops (and with the copies of the objects a_i, $1 \leqslant i \leqslant k$, in cell 1 representing the result of the computation).

From the explanations given above we infer that $N(M) = Ps(\Pi)$. □

The previous result shows that when rules of weight at least two are available, the hierarchies on the number of cells and states simultaneously collapse at level two and level one, respectively.

For antiport rules of minimal weight such a strong result is not known, although we can again bound the number of states (the hierarchy now collapses at level three), yet only provided that the number of cells can be arbitrary.

Theorem 8. *$PsRE = PsOtp_*(state_k, anti_i)$ for all $k \geqslant 3$ and $i \geqslant 1$.*

Proof. Consider a register machine $M = (n, R, l_0, l_h)$, with u ADD instructions, v SUB instructions, and generating $N(M) \subseteq \mathbf{N}^k$, for some $k \geqslant 1$.

Then we construct the tissue-like P system with channel states

$$\Pi = (O, T, K, w_1, \ldots, E, syn, s, \ldots, s, R_{(0,1)}, \ldots, 1),$$

of degree $1 + u + 2v$, with the cells labelled by 1, $add_1, \ldots, add_u, sub_1, sub_1', \ldots, sub_v, sub_v'$, with the initial state of all synapses being s and the output cell being that one with label 1, as well as

$$O = \{a_i \mid 1 \leqslant i \leqslant n\} \cup lab(M) \cup \{e, \#\},$$
$$T = \{a_i \mid 1 \leqslant i \leqslant k\},$$
$$K = \{s, s', s''\},$$
$$w_1 = l_0,$$
$$w_{add_i} = \#, \text{ for all } 1 \leqslant i \leqslant u,$$

$$\begin{aligned}
w_{sub_i} &= \#, \text{ for all } 1 \leqslant i \leqslant v,\\
w_{sub'_i} &= e, \text{ for all } 1 \leqslant i \leqslant v,\\
E &= O,\\
syn &= \{(0,1)\}\\
&\quad \cup \{(1, add_i),\ (0, add_i) \mid 1 \leqslant i \leqslant u\}\\
&\quad \cup \{(1, sub_i),\ (sub_i, sub'_i),\ (0, sub_i) \mid 1 \leqslant i \leqslant v\}
\end{aligned}$$

and with the sets of rules associated with the synapses as follows:

$$\begin{aligned}
R_{(0,1)} &= \{(s, \#/\#, s),\ (s, \lambda/l_h, s)\},\\
R_{(1,add_i)} &= \{(s, l_1/\lambda, s'),\ (s', \lambda/a_r, s''),\ (s'', \lambda/l_2, s),\ (s'', \lambda/l_3, s),\\
&\quad\ (s', \lambda/\#, s),\ (s'', \lambda/\#, s)\},\\
R_{(0,add_i)} &= \{(s, a_r/\lambda, s),\ (s, l_2/\lambda, s),\ (s, l_3/\lambda, s),\\
&\quad\ (s, e/\lambda, s')\}, \text{ for all } i \in \{1, 2, \ldots, u\},\\
&\quad\ \text{with the } i\text{th ADD rule being } l_1 : (add(r), l_2),\\
R_{(1,sub_i)} &= \{(s, l_1/\lambda, s'),\ (s', a_r/\lambda, s''),\ (s'', \lambda/l_2, s),\\
&\quad\ (s'', \lambda/\#, s),\ (s', \lambda/l_3, s)\},\\
R_{(sub_i,sub'_i)} &= \{(s, l_2/e, s'),\ (s', e/\lambda, s),\ (s, l_3/\lambda, s)\},\\
R_{(0,sub_i)} &= \{(s, l_2/l_1, s'),\ (s', l_3/\lambda, s)\}, \quad \text{for all } i \in \{1, 2, \ldots, v\},\\
&\quad\ \text{with the } i\text{th SUB rule being } l_1 : (sub(r), l_2, l_3).
\end{aligned}$$

The structure of the system Π, in the initial configuration, together with the sets of rules associated with the typical synapses, is pictorially indicated in Fig. 2. With the copies of the objects a_i, $1 \leqslant i \leqslant n$, we simulate the work of the register machine M; at the end of a halting computation, the copies of the objects a_i, $1 \leqslant i \leqslant k$, in cell 1 represent the result of the computation.

The simulation of ADD instructions of M is done with the help of the cells add_i, $1 \leqslant i \leqslant u$. Specifically, for each instruction add_i of the form $l_1 : (add(r), l_2, l_3)$ we proceed as follows. First, l_1 passes to cell add_i and the state of the synapse $(1, add_i)$ is changed to s'. This makes possible the passage of a_r from cell add_i to cell 1; because the state of the synapse becomes s'', in the next step we can also bring l_2 or l_3 into cell 1, returning the state of the synapse to s. The objects a_r, l_2, l_3 must be available in cell add_i at the right moment, because otherwise the trap symbol # is brought from cell add_i to cell 1, and then the computation never stops (the rule $(s, \#/\#, s)$ will be used forever on the synapse $(0, 1)$). The objects a_r, l_2, l_3 are brought to cell add_i from the environment in the presence of state s of synapse $(0, add_i)$; in order to stop bringing objects into cell add_i, we change the state of this synapse from s to s', when bringing inside the auxiliary object e. Therefore, the instruction $l_1 : (add(r), l_2, l_3)$ is correctly simulated (the states of the used synapses have returned to the initial s, hence, we can simulate other instructions).

The SUB instruction sub_i, of the form $l_1 : (sub(r), l_2, l_3)$, is simulated through the interaction of cell 1 with the cells sub_i and sub'_i, in the following way. First, the object l_1 is sent from cell 1 to cell sub_i, and the state of the synapse $(1, sub_i)$ is changed to s'. In the next step, l_1 exits cell sub_i, being exchanged with l_2, and the state of the synapse $(0, sub_i)$

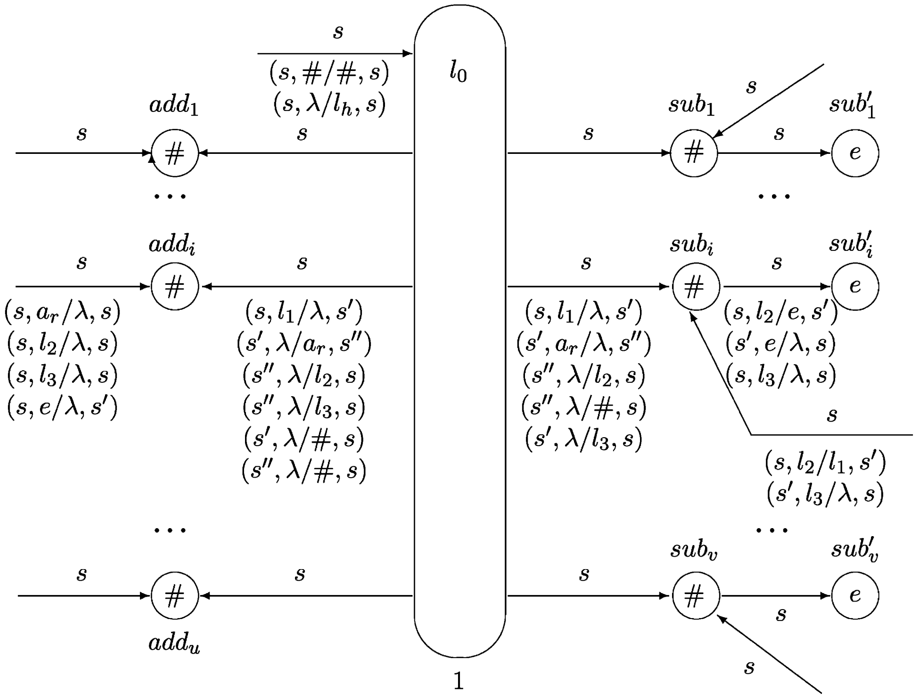

Fig. 2. The structure of the system from the proof of Theorem 8.

becomes s'. Simultaneously, if any copy of a_r is present in cell 1, then the rule $(s', a_r/\lambda, s'')$ is used, hence, one copy of a_r leaves cell 1 and the state of the synapse $(1, sub_i)$ becomes s''. If no copy of a_r exists in cell 1, then the state of the synapse remains s' and no rule is used here. In the third step, if the state of the synapse $(1, sub_i)$ is s'', then l_2 passes from cell sub_i to cell 1, returning the state of this synapse to s (and making possible the simulation of another rule). At the same time, l_3 enters cell sub_i, returning the state of the synapse $(0, sub_i)$ to s. Instead of passing to cell 1, the object l_2 can also pass to cell sub'_i, but in this case the trap symbol will be sent to cell 1, by means of the rule $(s'', \lambda/\#, s)$, and the computation will never stop. If the simulation of the case when a_r exists is correct, i.e., l_2 enters cell 1, then l_3 will pass to cell sub'_i in the next step (as the state of the synapse (sub_i, sub'_i) has remained s, the rule $(s, l_3/\lambda, s) \in R_{(sub_i, sub'_i)}$ can be used). If no copy of a_r is present in cell 1, then, after passing l_1 to cell sub_i and exchanging it with l_2 from the environment, l_2 must pass to cell sub'_i, in exchange with e, replacing state s by s' on the synapse (sub_i, sub'_i). At the same time, l_3 enters cell sub_i. In the next step, l_3 cannot go to cell sub'_i, because of the state s' of the synapse (sub_i, sub'_i), hence, it has to go to cell 1 by means of the rule $(s', \lambda/l_3, s)$ (the state of this synapse has remained s', because no a_r has changed s' into s'' as above). At the same time, the auxiliary object e passes back from cell sub_i to cell sub'_i, returning the state of this synapse to s.

The simulation of the SUB instruction now is complete, the states of the synapses are again s, hence, the simulation of instructions of M can continue.

In the whole simulation process, it is essential that in each ADD instruction $l_1 : (add(r), l_2, l_3)$ and in each SUB instruction $l_1 : (sub(r), l_2, l_3)$ the labels l_1, l_2, l_3 are mutually different.

When the halt label l_h is introduced in cell 1, it exits by means of the rule $(s, \lambda/l_h, s)$ and the computation stops. We conclude that $N(M) = Ps(\Pi)$ and this ends the proof. □

4. Further variants

The previous systems work in the generative mode, using the rules in a sequential manner. Obvious variations are obtained by considering the accepting mode. A possibility is to designate a cell as the input one, and to start the computation by introducing a multiset in that cell; this multiset is accepted if and only if the computation halts.

Another possibility is to consider as accepted the sequence of objects taken from the environment during a halting computation (as in [2,4,13]) and in this way we obtain language recognizing devices. The example from Section 2 works in a way for which this mode to define the recognized language is apparent—the language recognized by Π_1 is non-regular.

Then, of interest is to consider a parallel use of rules. In order to avoid conflicts in changing the labels, in each step, on each synapse, all rules leading from a state s to the same state s' should be considered. More specifically, “tables” of the form $T_{i,j}(s, s') = \{(s, x/y, s') \mid (s, x/y, s') \in R_{(i,j)}\}$ can be defined, for each synapse (i, j) and for each pair (s, s') of states; in each step one table is non-deterministically chosen and then used in a maximally parallel manner.

All these possibilities remain to be investigated. In general, we believe that the tissue-like P systems (with channel states) deserve further research efforts, motivated both by the mathematical problems they raise and also by the interesting connections with inter-cell communication in tissues (an important biological fact, see, e.g., [9]), neuron interaction in the brain, distributed computing (internet included).

References

[1] F. Bernardini, A. Păun, Universality of minimal symport/antiport: five membranes suffice, in: N. Jonoska, Gh. Păun, G. Rozenberg (Eds.), Aspects of Molecular Computing. Essays Dedicated to Tom Head on the Occasion of His 70th Birthday, Lecture Notes in Computer Science, Vol. 2950, Springer, Berlin, 2004, pp. 43–54.

[2] E. Csuhaj-Varjú, G. Vaszil, P automata or purely communicating accepting P systems, in: Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron (Eds.), Membrane Computing, Internat. Workshop WMC 2002, Curtea de Argeş, Romania, Revised Papers, Lecture Notes in Computer Science, Vol. 2597, Springer, Berlin, 2003, pp. 219–233.

[3] J. Dassow, Gh. Păun, Regulated Rewriting in Formal Language Theory, Springer, Berlin, 1989.

[4] R. Freund, M. Oswald, A short note on analysing P systems with antiport rules, Bull. EATCS 78 (2002) 231 –236.

[5] R. Freund, M. Oswald, P Systems with activated/prohibited membrane channels, in: Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron (Eds.), Membrane Computing, Internat. Workshop WMC 2002, Curtea de Argeş, Romania, Revised Papers, Lecture Notes in Computer Science, Vol. 2597, Springer, Berlin, 2003, pp. 261–268.

[6] R. Freund, Gh. Păun, On the number of non-terminal symbols in graph-controlled, programmed and matrix grammars, in: M. Margenstern, Y. Rogozhin (Eds.), Proc. Conf. Universal Machines and Computations, Chişinău, 2001, Lecture Notes in Computer Science, Vol. 2055, Springer, Berlin, 2001, pp. 214–225.

[7] R. Freund, Gh. Păun, On deterministic P systems, 2003, submitted for publication.

[8] P. Frisco, H.J. Hoogeboom, Simulating counter automata by P systems with symport/antiport, in: Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron (Eds.), Membrane Computing, Internat. Workshop WMC 2002, Curtea de Argeş, Romania, Revised Papers, Lecture Notes in Computer Science, Vol. 2597, Springer, Berlin, 2003, pp. 288–301.

[9] W.R. Loewenstein, The Touchstone of Life, Molecular Information, Cell Communication, and the Foundations of Life, Oxford University Press, New York, Oxford, 1999.

[10] C. Martín-Vide, J. Pazos, Gh. Păun, A. Rodriguez-Paton, Tissue P systems, Theoret. Comput. Sci. 296 (2) (2003) 295–326.

[11] M.L. Minsky, Computation: Finite and Infinite Machines, Prentice-Hall, Englewood Cliffs, NJ, USA, 1967.

[12] Gh. Păun, Computing with Membranes: An Introduction, Springer, Berlin, 2002.

[13] Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron (Eds.), Membrane Computing, Internat. Workshop WMC 2002, Curtea de Argeş, Romania, Revised Papers, Lecture Notes in Computer Science, Vol. 2597, Springer, Berlin, 2003.

[14] G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages (3 volumes), Springer, Berlin, 1997.