Tissue-like P Systems with Channel-States

Rudolf FREUND[1], Gheorghe PĂUN[2,3], Mario J. PÉREZ JIMÉNEZ[3]

[1]Faculty of Computer Science
Vienna University of Technology
Favoritenstr. 9–11, A–1040 Vienna, Austria
E-mail: rudi@emcc.at

[2]Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucureşti, Romania

[3]Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mail: {gpaun,marper}@us.es

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 proven to be universal when having arbitrarily many states and minimal antiport rules, or two states, 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. A series of open problems are also formulated.

1 Introduction

In membrane computing area there are two main classes of systems: cell-like and tissue-like P systems. The former type is inspired from the 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 communicating among cells, and the other one, closer to the neural net organization, having states associated with the cells, for controlling multiset rewriting rules which make evolve the multisets of objects from the cells.

In the present paper, we take a different perspective, somewhat mixing the two subcases of tissue-like systems: we associate states to the links between cells, and use these states in order to control the communication among cells; in its turn, the communication is done by means of symport/antiport rules. Among 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 among 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 among 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 rules 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 States

The reader is supposed to be familiar with basic elements of membrane computing, e.g., from [11] (rather useful is the comprehensive information which can be found in the web page `http://psystems.disco.unimib.it`), in particular, with the tissue-like P systems introduced in [9]. Here we deal with the following type of systems (for the very few elements of computability – mainly formal language theory – we refer to any monograph in this area, in particular, to [13]; just for the sake of completeness, we 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).

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

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

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), $w_1, \ldots, w_m$ 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 $1, 2, \ldots, m$), $E \subseteq O$ is the set of objects present in arbitrarily many copies in the environment, $syn \subseteq \{(i,j) \mid i,j \in \{0,1,2,\ldots,m\}, i \neq j\}$ is the set of links among cells (we call them *synapses*; 0 indicates the environment) such that for $i,j \in \{0,1,\ldots,m\}$ 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, $i_o \in \{1,2,\ldots,m\}$ is the *output* cell.

We note the important restriction that there is at most one synapse among two given cells, and the synapse is given as an ordered pair (i,j), with which a state from K is associated. The fact that the pair is ordered does not restrict the communication among the two cells (or between a cell and the environment), because we work here in the general case of 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 empty are, in fact, symport rules, but we do not explicitly consider here this distinction, 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 $w_1, \ldots, w_m$ 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 $|x| \leq i$, $|y| \leq i$ is denoted by $PsOtP_m(states_k, anti_i)$. When one of the parameters m, k, i is not bounded, it is replaced with $*$. We also denote by $PsFL$ the set of Parikh images of languages from a given family FL; by RE we denote the family of recursively enumerable languages, and by CF the family of context-free languages.

3 Two Examples

Before investigating the computing power of the above introduced devices, let us illustrate their work by some examples. The first one (of degree 3) is simpler. Formally, it is given as follows:

$$\begin{aligned} \Pi_1 &= (O, T, K, w_1, w_2, w_3, E, syn, (s_{(i,j)})_{(i,j)\in syn}, (R_{(i,j)})_{(i,j)\in syn}, i_o), \\ O &= \{a, b\}, \\ T &= \{a, b\}, \end{aligned}$$

$$
\begin{aligned}
K &= \{s, s', s''\},\\
w_i &= \lambda, \text{ for all } i = 1, 2, 3,\\
E &= O,\\
syn &= \{(0,1), (1,2), (1,3)\},\\
R_{(0,1)} &= \{(s, a/\lambda, s),\ (s, a/\lambda, s'),\ (s', b/\lambda, s'),\ (s', b/\lambda, s'')\},\\
R_{(1,2)} &= \{(s, a/\lambda, s),\ (s, b/\lambda, s),\ (s, \lambda/a, s),\ (s, \lambda/b, s)\},\\
R_{(1,3)} &= \{(s, b/\lambda, s'),\ (s', a/\lambda, s)\},\\
i_o &= 3.
\end{aligned}
$$

The system is pictorially given in Figure 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 inside the initial multiset of objects and outside the label; the output cell, that with label 3, is indicated by having it doubly encircled.

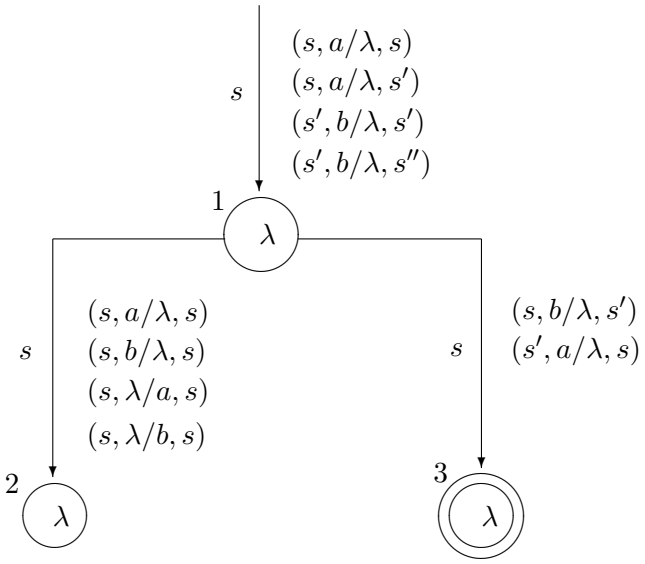

Figure 1. The system Π_1 (rules and initial configuration)

The functioning of the system Π_1 is rather clear: in state s, cell 1 brings inside $n \geq 0$ copies of object a, then the synapse $(0, 1)$ changes the state to s' when one further a in brought in; in state s' we bring in cell 1 a number $m \geq 0$ of copies of object b; 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 among cells 1 and 2, hence the computation can stop only if all objects are moved to cell 3, the output one. The channel from cell 1 to cell 3 can be "open" 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 cell 1 to cell 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 were equal, or the number of copies of b is larger by 1 than the number of copies of a. That is, $Ps(\Pi_1) = \{(n, n) \mid n \geq 1\} \cup \{(n, n+1) \mid n \geq 1\}$.

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

The functioning of the second example we discuss here, Π_2, is much more intricated. Instead of giving this system in a formal manner, we present it pictorially, in Figure 2, following the same conventions as in Figure 1. The output cell is 1 and the only terminal object is a.

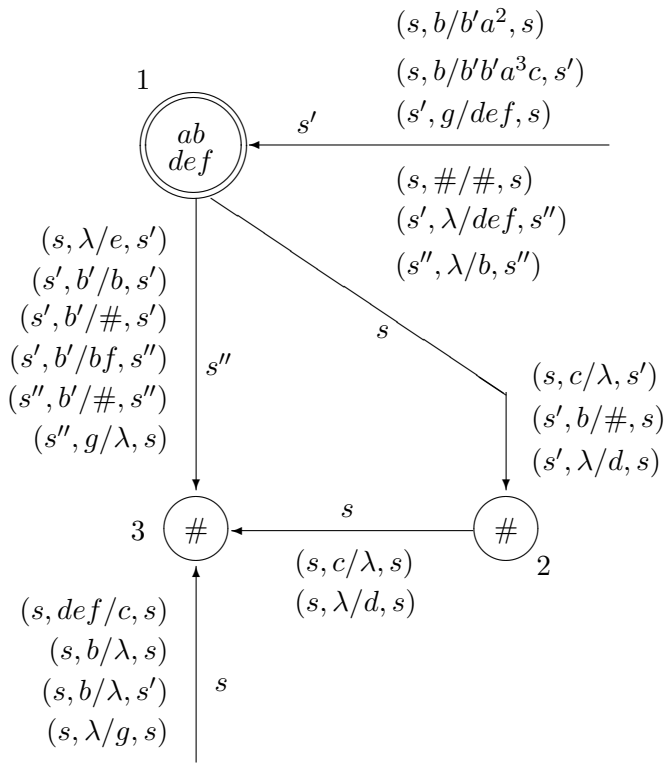

Figure 2. The system Π_2 (rules and initial configuration)

This system computes the squares of natural numbers, in the following way. We start with objects $abdef$ in cell 1. The objects def go along the synapse $(0, 1)$ and change its state to s, bringing g in cell 1; this object passes to cell 3, changing the state of the synapse to s and then exits through the synapse $(0, 3)$.

Assume that we are in a configuration with all synapses in state s, with n^2 copies of object a and n copies of b present in cell 1; initially, after the steps mentioned above, this is the case. Each copy of b is sent to the environment, in exchange of b' and two copies of a; the last copy of b from cell 1 is exchanged for two copies of b' and three copies of a. In this way, the number of copies of a becomes $n^2 + 2n + 1 = (n + 1)^2$. In the last step, also c is brought in cell 1; this object passes to cell 2, "opening" this synapse for object b; if any copy of b is still present in cell 1, then the trap-object $\#$ is brought in cell 1 and the computation never stops.

From cell 2, c passes to cell 3, and from here exits to the environment, bringing in cell 3 the objects def. The object d will go to cell 2 and then to cell 1, restoring the state of the synapse $(1, 2)$ to s, while e goes to cell 1, changing the state of the synapse $(1, 3)$ to s'. This makes possible the exchange of each copy of b' from cell 1 with a copy of b from cell 3 (this last object is continuously brought in cell 3 from the environment – but the process can be finished by passing the synapse $(0, 3)$ to state s'; however, if this happens too early,

then the object # is moved from cell 3 to cell 1 and the computation will last forever). The exchange of b' with b continues until changing the state of the synapse $(1,3)$ to s'', and also moving f from cell 3 to cell 1. This should complete the change of b', otherwise again the trap-object is moved to cell 1. In this moment, all objects def are again in cell 1, as in the initial configuration, hence we can iterate the process (thus passing to the square of the next natural number). If, instead, we send def outside by means of the rule $(s', \lambda/def, s'')$, then the synapse $(0,1)$ passes to state s'', which only allows the exit of all objects b from cell 1. In this way, only copies of object a remain in cell 1. The rule $(s', \lambda/def, s'')$ can be used also in the initial configuration, hence also the square of 1 is obtained.

Consequently, $Ps(\Pi_2) = \{n^2 \mid n \geq 1\}$. Note that in the halting configuration, only copies of the terminal object a are present in the output cell.

As we will see soon, the same set of numbers can be computed by systems with a small number of cells or states, and with simpler rules.

4 Technical Prerequisites

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

In what concerns the register machines, we refer to [10] for original definitions, and to [5], [14] for definitions like that we use in this paper.

An *n-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:

- $l_1 : (add(r), l_2)$,
  Add 1 to the contents of register r and proceed to the instruction (labelled with) l_2. (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 recognize a vector $(s_1, \ldots, s_k)$ of natural numbers if, starting with the instruction with label l_0, with the numbers $s_1, \ldots, s_k$ placed in the first k registers (and the other registers containing the number 0), the machine stops (it reaches the instruction $l_h : halt$) with all registers containing the number 0.

The register machines are know to be computationally universal, equal in power to Turing machines: they recognize exactly the sets of vectors of natural numbers which can be recognized/computed by Turing machines, that is, the family $PsRE$.

Without loss of the generality, in the proofs from the following section we will assume that in each ADD instruction $l_1 : (add(r), l_2)$ and in each SUB instruction $l_1 : (sub(r), l_2, l_3)$ the labels l_1, l_2, l_3 are mutually distinct. This goal can be easily achieved. For instance,

in the case of SUB instructions, we replace each instruction $l_1 : (sub(r), l_2, l_3)$ with the instructions $l_1 : (sub(r), l_2', l_3'')$, $l_2' : (add(n+1), l_2''')$, $l_2''' : (sub(n+1), l_2, l_h)$, $l_3'' : (add(n+1), l_3^{iv})$, $l^{iv} : (sub(n+1), l_3, l_h)$, where $n+1$ is a new register (the same for all starting SUB instructions), and all primed labels are distinct and different from the initial labels.

We also use below the *matrix grammars*. For details, we refer to [3] and to the chapter of [13] devoted to regulated rewriting, and we introduce here only 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 $(A_1 \to x_1, \ldots, A_n \to x_n)$, $n \geq 1$, 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 nonterminal 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 is a matrix $(A_1 \to x_1, \ldots, A_n \to x_n)$ in M and the strings $w_i \in (N \cup T)^*, 1 \leq i \leq n+1$, such that $w = w_1, z = w_{n+1}$, and, for all $1 \leq i \leq n$, $w_i = w_i' A_i w_i'', w_{i+1} = w_i' x_i w_i''$, 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 grammar $G = (N, T, S, M)$ is in this 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 in one of the following forms:

1. $(S \to XA)$, with $X \in N_1, A \in N_2$,
2. $(X \to Y, A \to x)$, with $X, Y \in N_1, A \in N_2, x \in (N_2 \cup T)^*, |x| \leq 2$,
3. $(X \to \lambda, A \to x)$, with $X \in N_1, A \in N_2$, and $x \in T^*, |x| \leq 2$.

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

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

Hence, M_f contains rules of the following forms:

1. $(S \to XA)$, with $X \in N_1, A \in N_2$,
2. $(X \to Y, A \to x)$, with $X, Y \in N_1, A \in N_2, x \in (N_2 \cup T)^*, |x| \leq 2$,
3. $(X \to f, A \to x)$, with $X \in N_1, A \in N_2$, and $x \in T^*, |x| \leq 2$,
4. $(f \to \lambda)$.

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., [7], [6]), 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 [7], [6] lies in the difference between the way the two types of systems work: sequentially here, in a maximally parallel manner in the mentioned 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; then, 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 4.1 $PsMAT \subseteq PsOtP_1(state_*, anti_1)$.

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. We construct the tissue-like P system

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

where $(S \to X_0 A_0)$ is the initial matrix of M.

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, $\Psi_T(L(G)) = Ps(\Pi)$ and the proof is complete. □

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

Lemma 4.2 $PsMAT \subseteq PsOtP_1(state_1, anti_2)$.

Proof. Consider again a matrix grammar $G = (N_1 \cup N_2 \cup \{S, f\}, T, S, M)$ in the f-binary normal form and construct the tissue-like P system

$$\begin{aligned}
\Pi &= (O, T, \{s\}, X_0A_0Z, O, \{(0,1)\}, s, R_{(0,1)}, 1),\\
O &= N_1 \cup \{f\} \cup N_2 \cup T \cup \{\langle X, \alpha\beta\rangle \mid X \in N_1 \cup \{f\}, \alpha, \beta \in N_2 \cup T\},\\
R_{(0,1)} &= \{(s, Yx/XA, s) \mid (X \to Y, A \to x) \in M\\
&\quad X \in N_1, Y \in N_1 \cup \{f\}, A \in N_2, x \in N_2 \cup T \cup \{\lambda\}\}\\
&\cup \{(s, Y\langle Y, \alpha_1\alpha_2\rangle/XA, s), (s, \alpha_1\alpha_2/\langle Y, \alpha_1\alpha_2\rangle, s) \mid (X \to Y, A \to \alpha_1\alpha_2) \in M,\\
&\quad X \in N_1,\ Y \in N_1 \cup \{f\}, A \in N_2, \alpha_1, \alpha_2 \in N_2 \cup T\}\\
&\cup \{(s, \alpha/\alpha, s) \mid \alpha \in N_1 \cup N_2\}\\
&\cup \{(f, \lambda/Z, f)\},
\end{aligned}$$

where $(S \to X_0A_0)$ is the initial matrix of M.

The state plays no rôle, the matrices of M are simulated by the antiport rules. As long as at least a nonterminal from $N_1 \cup N_2$ is present, the computation must continue. The equality $\Psi_T(L(G)) = Ps(\Pi)$ is obvious and this completes the proof. □

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

Lemma 4.3 $PsOtP_1(state_*, anti_*) \subseteq PsMAT$.

Proof. Let $\Pi = (O, T', K, w_1, E, \{(0,1)\}, s_0, R_{(0,1)}, 1)$ be a tissue-like P system. We 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. $(S \to s_0h(w_1))$,
2. $(s_1 \to s_2h(x))$, for $(s_1, x/\lambda, s_2) \in R_{(0,1)}$,
3. $(s_1 \to s_2, x'_1 \to \lambda, \ldots, x'_k \to \lambda)$, for $(s_1, \lambda/x, s_2) \in R_{(0,1)}$,
   for $x = x_1x_2 \ldots x_k$, $k \geq 1$, with $x_i \in O, 1 \leq i \leq k$,
4. $(s_1 \to s_2, y'_1 \to x, y'_2 \to \lambda, \ldots, y'_k \to \lambda)$, for $(s_1, x/y, s_2) \in R_{(0,1)}$,
   for $y = y_1y_2 \ldots y_k$, $k \geq 1$, with $y_i \in O, 1 \leq i \leq k$,
5. $(s \to s', a' \to a)$,
   $(s' \to s', a' \to a)$, for $s \in K, a \in O$,
   $(s' \to s'')$, for $s \in K$,

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

In the presence of nonterminals 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

$$\begin{aligned} L &= \{s_1'' z_1 y z_2 \mid (s_1, x/y, s_2) \in R_{(0,1)}, z_1, z_2 \in O^*\} \\ &\cup \{s_1'' z \mid (s_1, x/\lambda, s_2) \in R_{(0,1)}, z \in T^*\}. \end{aligned}$$

This language contains all strings which 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 which were encoded in the strings of $L(G)$. Consider now the morphism g which erases the symbol s'' as well as all symbols from $O - T'$. The equality $Ps(\Pi) = \Psi_{T'}(g(L(G) \cap L'))$ holds. Because 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 4.1 *$PsMAT = PsOtP_1(state_k, anti_i) = PsOtP_1(state_*, anti_j)$ for all $k \geq 1$ and $i \geq 2$ as well as for all $j \geq 1$ (each of k, i, j can also be equal to $*$).*

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 minimal antiport rules we get again the 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 4.2 *$PsRE = PsOtP_m(state_*, anti_i)$ for all $m \geq 2$ and $i \geq 1$.*

Proof. It is sufficient to 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 $lab(M) = \{g_1, \ldots, g_t\}$, and recognizing the set of vectors $N(M) \subseteq \mathbf{N}^k$, for some $k \geq 1$) and construct the tissue-like P system (of degree 2)

$$\Pi = (O, T, K, \lambda, w_2, E, \{(0,1), (1,2), (0,2)\}, s, s, s, R_{(0,1)}, R_{(1,2)}, R_{(0,2)}, 1),$$

with

$$\begin{aligned} O &= \{a_i \mid 1 \leq i \leq n\} \cup \{b_i \mid 1 \leq i \leq k\} \cup \{l, l', l''', l^v \mid l \in lab(M)\}, \\ T &= \{b_i \mid 1 \leq i \leq k\}, \\ K &= \{s_i \mid 1 \leq i \leq k\} \cup \{s, s'\} \cup \{l, l'', l^{iv} \mid l \in lab(M)\}, \\ w_2 &= g_1' g_2' \ldots g_t', \\ E &= O, \end{aligned}$$

and the following sets of rules.

1. The next rules are introduced in $R_{(0,1)}$, for all $i = 1, 2, \ldots, k$:
   $(s, a_i/\lambda, s_i)$,
   $(s_i, b_i/\lambda, s)$,
   $(s_i, b_i/\lambda, l_0)$.
   By using these rules, at the beginning of a computation we introduce in cell 1 some arbitrary numbers of objects a_i, b_i, for $1 \leq i \leq k$, the same number for a_i and b_i for each i. The copies of a_i will be used for simulating the work of the register machine, the copies of b_i will simply remain in cell 1; if the computation stops, then the result of the computation will be given by the multiplicity of objects b_i. In the last step, the label of the synapse $(0, 1)$ is changed to l_0, the starting label of M.

2. If $N(M)$ also contains the vector $(0, 0, \ldots, 0)$, then we introduce in $R_{(0,1)}$ also the following rules:
   $(s, a_1/\lambda, s')$,
   $(s', \lambda/a_1, l_0)$.
   In this way, we start the computation with cell 1 empty and with the label l_0 marking the synapse $(0, 1)$.

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

4. For each SUB instruction $l_1 : (sub(r), l_2, l_3)$ from R we introduce in the sets of rules of Π the rules indicated in the table below. 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 in the first cell the object l_1 (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 in 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 from the environment (it could be useful when simulating another instruction of M), while on the synapse $(0, 1)$ we use the rule $(l_1^{iv}, l_1^v/l_1''', l_1^{iv})$; its role is to bring the "checker" l_1^v in 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 in 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 among 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 for a possible use at 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 subtraction instruction.

It should be noted the important details that in cell 1 we do not have any object different from a_j, for those j for which we have non-zero registers in M, and b_i, $1 \leq i \leq k$, as in the beginning, 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 above l_2', l_3' (during the simulation, we bring both of them from the environment into cell 2, although only one of them was sent to cell 1 in order to change the label of the synapse $(0,1)$).

Therefore, the simulation of instructions from R can continue.

5. 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 above explanation it is clear that $N(M) = Ps(\Pi)$, and this concludes the proof. □

The previous proof uses a number of states which depends on the number of labels used by the register machine which is simulated by our system. The number of states can be reduced to 2 at the expense of increasing by one the weight of rules.

Theorem 4.3 *$PsRE = PsOtP_m(state_k, anti_i)$ for all $m \geq 2$, $k \geq 2$, and $i \geq 2$.*

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

$$\Pi = (O, T, K, \lambda, w_2, E, \{(0,1),(1,2),(0,2)\}, s, s, s, R_{(0,1)}, R_{(1,2)}, R_{(0,2)}, 1),$$

with

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

and the following sets of rules.

1. The next rules are introduced in $R_{(0,1)}$, for all $i = 1, 2, \ldots, k$:
   $(s, a_i b_i/\lambda, s)$,
   $(s, l_0/\lambda, s')$.
   Like in the previous proof, by these rules we introduce in the first cell the vector to be recognized – the null one included – together with the label l_0, thus starting the simulation of a computation in M.

2. For each ADD instruction $l_1 : (add(r), l_2)$ from R, we introduce in $R_{(0,1)}$ the rule
   $(s', l_2 a_r/l_1, s')$.
   From now on, the states play no role in the computation, the instructions of M are simulated by the antiport rules in a way rather similar to that from [6], but using rules in a sequential manner, and making use of the two cells (and the environment) for controlling the computation.

3. For each SUB instruction $l_1 : (sub(r), l_2, l_3)$ from R we introduce in the sets of rules of Π the rules indicated in the table below. 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	$(s', l_1' l_1''/l_1, s')$	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 $(s, l_3/l_1''', s)$

   The label l_1 is replaced in the first cell by l_1', l_1''. In the second step, if a copy of a_r is present, then object l_1' is sent to the second cell together with a copy of a_r and the auxiliary object e is brought in cell 1; if no copy of a_r exists, then l_1' waits in cell 1. Simultaneously, l_1''' is brought in 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 have here e (hence a_r was present), then $l_1''' e$ bring from cell 2 the label l_2, completing the simulation of the instruction for the case when the substraction was possible. If we still have l_1' in cell 1, then $l_1''' l_1'$ bring l_3 from cell 2, thus completing the simulation of the instruction for the case when the subtraction is 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 in the system the label l_3. In this way, the contents of cell 2 is restored, hence we can continue simulating the instructions of M.

4. We also introduce in $R_{(0,1)}$ the rule
   $(s', \lambda/l_h, s')$,
   hence the work of Π will stop exactly when the work of M stops (and no other object than the initial b_is are present in cell 1).

From the above explanation it is clear that $N(M) = Ps(\Pi)$, and this concludes the proof. □

This result shows that when rules of weight at least two are available, the hierarchies on the number of membranes and states collapse, simultaneously at level two. This is

not known for minimal antiport rules, although we can again bound the number of states (the hierarchy collapses at level three) provided that the number of membranes can be arbitrary.

Theorem 4.4 *$PsRE = PsOtP_*(state_k, anti_i)$ for all $k \geq 3$ and $i \geq 1$.*

Proof. Consider a register machine $M = (n, R, l_0, l_h)$, with u ADD instructions, v SUB instructions, and $N(M) \subseteq \mathbf{N}^k$, for some $k \geq 1$. We construct the tissue-like P system Π, of degree $k + 2 + u + 2v$, with the cells labeled with $1, 2, \ldots, k, k+1, k+2$, $add_1, \ldots, add_u, sub_1, sub'_1, \ldots, sub_v, sub'_v$, with

$$\begin{aligned}
O &= \{a_i \mid 1 \leq i \leq n\} \cup \{b_i, e_i \mid 1 \leq i \leq k\} \cup lab(M) \cup \{e, \#\},\\
T &= \{b_i \mid 1 \leq i \leq k\},\\
K &= \{s, s', s''\},\\
w_1 &= l_0,\\
w_i &= \lambda, \text{ for all } 2 \leq i \leq k+2,\\
w_{add_i} &= \#, \text{ for all } 1 \leq i \leq u,\\
w_{sub_i} &= \#, \text{ for all } 1 \leq i \leq v,\\
w_{sub'_i} &= e, \text{ for all } 1 \leq i \leq v,\\
E &= O,\\
syn &= \{(0,i),\ (i,k+2),\ (i,i+1) \mid 1 \leq i \leq k\}\\
&\cup \{(k+1,k+2),\ (0,k+2)\}\\
&\cup \{(k+2,add_i),\ (0,add_i) \mid 1 \leq i \leq u\}\\
&\cup \{(k+2,sub_i),\ (sub_i,sub'_i),\ (0,sub_i) \mid 1 \leq i \leq v\}.
\end{aligned}$$

The initial state of all synapses is s and the output cell is that with label $k+2$. In turn, the sets of rules associated with the synapses are as follows:

$$\begin{aligned}
R_{(0,i)} &= \{(s, a_i/\lambda, s'),\\
&\quad (s', b_i/\lambda, s),\\
&\quad (s, e_i/\lambda, s'')\},\\
R_{(i,k+2)} &= \{(s, a_i/\lambda, s),\\
&\quad (s, b_i/\lambda, s)\},\\
R_{(i,i+1)} &= \{(s, e_i/\lambda, s'),\\
&\quad (s', l_0/\lambda, s')\}, \text{ for all } i = 1, 2, \ldots, k,\\
R_{(k+1,k+2)} &= \{(s, l_0/\lambda, s)\},\\
R_{(0,k+2)} &= \{(s, \#/\#, s),\\
&\quad (s, \lambda/l_h, s)\},\\
R_{(k+2,add_i)} &= \{(s, l_1/\lambda, s'),\\
&\quad (s', \lambda/a_r, s''),\\
&\quad (s'', \lambda/l_2, s),\\
&\quad (s', \lambda/\#, s),\\
&\quad (s'', \lambda/\#, s)\},
\end{aligned}$$

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

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 Figure 3.

The system works as follows. The cells with labels from 1 to k are used for introducing in the system objects a_i, b_i, $1 \leq i \leq k$; with objects a_i we simulate the work of the register machine M, the copies of objects b_i remain in the end of halting computations in cell $k+2$, representing the result of the computation. After entering cell i, each object a_i and b_i is sent to cell $k+2$. Each cell $i = 1, 2, \ldots, k$ ends its communication with the environment by bringing inside the object e_i. These objects are used in order to open the channels among cells $i = 1, 2, \ldots, k, k+1$: by passing object e_i from cell i to cell $i+1$, the state of the synapse $(i, i+1)$ is changes to s'. This state makes possible the passage of object l_0, the initial label of M, step by step, from cell 1 to cell $k+1$. From cell $k+1$, the object l_0 is sent to cell $k+2$, for starting the simulation of a computation in M. Therefore, we can bring l_0 in cell $k+2$ if and only if all "input cells" $1, 2, \ldots, k$ has completed their work, and entered the "blocked state" s''.

The simulation of ADD instructions of M is done with the help of the cells $add_i, 1 \leq i \leq u$. Specifically, for each instruction add_i of the form $l_1 : (add(r), l_2)$ we proceed as follows. First, l_1 passes to cell add_i and the state of the synapse $(k+2, add_i)$ is changed to s'. This makes possible the passage of a_r from cell add_i to cell $k+2$; because the state of the synapse becomes s'', in the next step we can also bring l_2 in cell $k+2$, returning the state of the synapse to s. The objects a_r and l_2 must be available in cell add_i in the right moment, because otherwise the trap-object $\#$ is brought from cell add_i to cell $k+2$, and then the computation never stops (the rule $(s, \#/\#, s)$ will be used forever on the synapse $(0, k+2)$). The objects a_r, l_2 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 $l_1 : (add(r), l_2)$ is correctly simulated (the states of the used synapses are returned to the initial s, hence we can simulate other instructions).

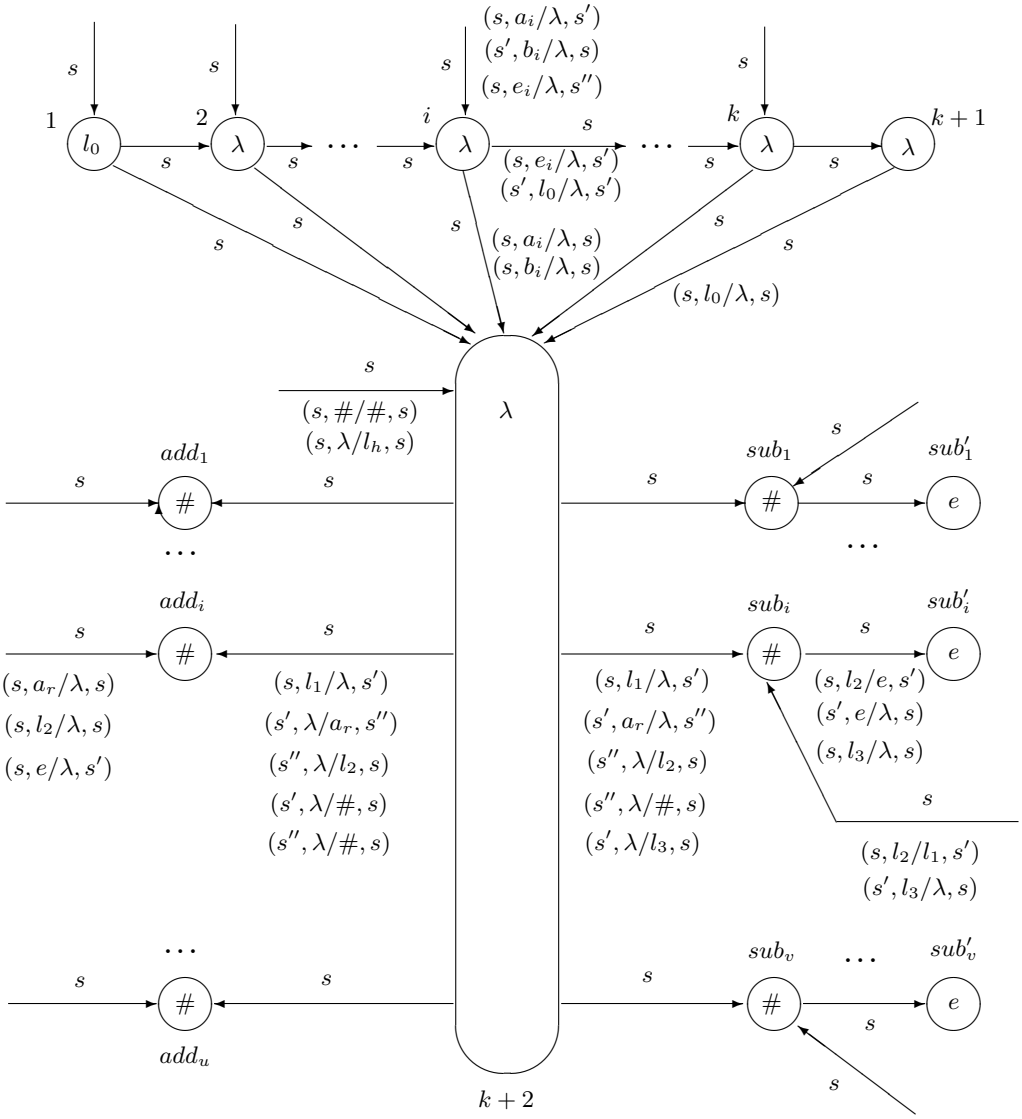

Figure 3. The structure of the system from the proof of Theorem 4.4

The SUB instruction sub_i, of the form $l_1 : (sub(r), l_2, l_3)$, is simulated through the interaction of cell $k+2$ with the cells sub_i and sub'_i, in the following way. First, the object l_1 is sent from cell $k+2$ to cell sub_i, and the state of the synapse $(k+2, sub_i)$ 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 $(0, sub_i)$ becomes s'. Simultaneously, if any copy of a_r is present in cell $k+2$, then the rule $(s', a_r/\lambda, s'')$ is used, hence one copy of a_r leaves cell $k+2$ and the state of the synapse $(k+2, sub_i)$ becomes s''. If no copy of a_r exists in cell $k+2$, then the state of the synapse remains s' and no rule is used here. In the third step, if the state of the synapse $(k+2, sub_i)$ is s'', then l_2 passes from cell sub_i to cell $k+2$, 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 $k+2$, the object l_2 can also pass to cell sub'_i, but in this case the trap-object should be

sent to cell $k+2$, 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, hence l_2 enters cell $k+2$, then l_3 will pass in the next step to cell sub'_i: the state of the synapse (sub_i, sub'_i) has remained s, hence 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 $k+2$, 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 with 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 will go to cell $k+2$, 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 is complete, the states of the synapses are again s, hence the simulation of instructions of M can continue.

In this process, it is essential that the labels l_1, l_2, l_3 from each instruction $l_1 : (sub(r), l_2, l_3)$ are mutually different.

When the halt label l_h is introduced in cell $k+2$, 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. □

5 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.

Because in the accepting mode we do not have to take care of the way the initial values of the register machine simulated by a P system as in Theorems 4.2, 4.3, 4.4 are introduced, we can save states in the constructions from the proofs of these theorems. This is especially of interest in the case of Theorem 4.3, where we use the two states only for introducing the input, and for the computation one state suffices; therefore, in the accepting case, the universality is obtained with only one state.

Another possibility is to consider as accepted the sequence of objects taken from the environment during a halting computation (as in [2] and [4]) and in this way we obtain language recognizing devices. The first example from Section 3 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 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., [8]), neuron interaction in the brain, distributed computing (internet included).

References

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

[2] E. Csuhaj-Varju, G. Vaszil, P automata or purely communicating accepting P systems. In [12], 219–233.

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

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

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

[6] R. Freund, Gh. Păun, On deterministic P systems. Submitted, 2003.

[7] P. Frisco, H.J. Hoogeboom, Simulating counter automata by P systems with symport/antiport. In [12], 288–301.

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

[9] C. Martín-Vide, J. Pazos, Gh. Păun, A. Rodríguez-Patón, Tissue P systems. *Theoretical Computer Sci.*, 296, 2 (2003), 295–326.

[10] M.L. Minsky, *Computation: Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, New Jersey, USA, 1967.

[11] Gh. Păun, *Computing with Membranes: An Introduction*. Springer-Verlag, Berlin, 2002.

[12] Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, eds., *Membrane Computing. International Workshop WMC 2002, Curtea de Argeş, Romania, Revised Papers. Lecture Notes in Computer Science* 2597, Springer-Verlag, Berlin, 2003.

[13] G. Rozenberg, A. Salomaa, eds., *Handbook of Formal Languages* (3 volumes), Springer-Verlag, Berlin, 1997.

[14] P. Sosik, R. Freund, P systems without priorities are computationally universal. In [12], 400–409.