P Systems with Adjoining Controlled Communication Rules

Mihai Ionescu[1], Dragoş Sburlan[2]

[1] Rovira i Virgili University
Research Group on Mathematical Linguistics
Tarragona, Spain
armandmihai.ionescu@urv.net
[2] Ovidius University
Faculty of Mathematics and Informatics
Constantza, Romania
dsburlan@univ-ovidius.ro

Summary. This paper proposes a new model of P systems where the rules are activated by objects present in the neighboring regions. We obtain the computational completeness considering only two membranes, external inhibitors and carriers. Leaving the carriers apart we obtain equality with ET0L systems in terms of number sets.

1 Introduction

Having as inspiration the way living cells are divided by membranes into compartments where various biochemical processes take place, *P systems* (also known as *membrane systems*) area grew rapidly since Gheorghe Păun, proposed the first model in 1998 ([4]). A complete bibliography of P systems can be found on the P system webpage ([8]).

Within the living cell there are several energy consuming activities. Among them there is the transport activity which is of three types: *diffusion*, *facilitated diffusion*, and *active transport*. Simple diffusion means that the molecules can pass directly through the membrane, always down a concentration gradient, while in the case of facilitated diffusion and active transport molecules can pass both down an up the concentration gradient. In the facilitated diffusion membrane protein channels are used to allow charged molecules (which otherwise could not diffuse across the cell membrane) to freely diffuse the cell, while active transport requires the expenditure of energy to transport the molecule from one side of the membrane to the other.

Hence, living cells get/expel from/to their environment many substances and for this aim they have developed specific transport systems across membranes, even against a concentration gradient. Often enough this necessity of the living

cell to expel or attract various molecules is triggered by the presence or the absence of certain chemicals in the immediate neighboring (inner or outer) regions.

Here we deal with P systems where the rules from a given region are activated precisely by the presence or the absence of certain symbols in the neighboring regions. This model has a biological counterpart and it is inspired by the chemicals that pass through the membranes of the cell, from one region to another, in the sense of polarization gradient. In this case, the electrical charge plays the role of the promoter.

Before going into the definition of the new model and its computational power (Section 3) let us briefly remind the reader some basic notions and notations (Section 2). Section 4 is dedicated to the conclusions and challenges for further research.

2 Preliminaries and Definitions

We assume familiarity with the basics of formal language theory (see [6]), as well as with the basics of membrane computing (see [5]).

An *alphabet* is a finite set of symbols (letters), and a word (string) over an alphabet Σ is a finite sequence of letters from Σ. We denote the empty word by λ, the length of a word w by $|w|$, and the number of occurrences of a symbol a in w by $|w|_a$. The (con)catenation of two words x and y is denoted by xy.

A *language* over Σ is a (possibly infinite) set of words over Σ. The language consisting of all words over Σ is denoted by Σ^*, and $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$. We denote by `REG`, `CF`, `ET0L`, `CS`, `RE` the families of languages generated by regular, context-free, table Lindemayer interactionless systems context-sensitive, and of arbitrary grammars, respectively (`RE` stands for recursively enumerable languages). The following strict inclusions hold: `REG` $\subset$ `CF` $\subset$ `ET0L` $\subset$ `CS` $\subset$ `RE`.

For a family `FL` of languages, `NFL` denotes the family of length sets of languages in `FL`. The following relations hold: `NREG` = `NCF` $\subset$ `NET0L` $\subset$ `NCS` $\subset$ `NRE`.

The multisets over a given finite support (alphabet) are represented by strings of symbols. The order of symbols does not matter, because the number of copies of an object in a multiset is given by the number of occurrences of the corresponding symbol in the string (see [1] for other ways to specify multisets).

3 The Model

Based on the biological observations mentioned in the introductory section we introduce the following new class of P systems.

3.1 Defining the Model

Definition 1. *A P system with adjoining controlled communication rules (called in short, a PACC system) is a construct*

$$\Pi = (V, C, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, i_0),$$

where:

- V *is the alphabet of objects;*
- $C \subseteq V$ *is the set of carriers;*
- μ *is a membrane structure with* m *membranes (labeled in a one-to-one manner by* $1, \ldots, m$*);*
- $w_1, \ldots, w_m$ *are the multisets of objects initially present in the regions of* Π*;*
- $R_1, \ldots, R_m$ *are finite sets of communication rules associated to membranes, that are of the following types:*
  - ◇ simple rules:
    $[A]_i \longrightarrow [\ ]_i\alpha$ *or* $A[\ ]_i \longrightarrow [\alpha]_i$*, for* $A \in V \setminus C$, $\alpha \in (V \setminus C)^*$,
  - ◇ promoted simple rules:
    $[A]_i B \longrightarrow [\ ]_i\alpha$ *or* $A[B]_i \longrightarrow [\alpha]_i$*, for* $A, B \in V \setminus C$, $\alpha \in (V \setminus C)^*$,
  - ◇ inhibited simple rules:
    $[A]_i \neg B \longrightarrow [\ ]_i\alpha$ *or* $A[\neg B]_i \longrightarrow [\alpha]_i$*, for* $A, B \in V \setminus C$, $\alpha \in (V \setminus C)^*$,
  - ◇ carrier rules:
    pairs of rules $[cA]_i \longrightarrow [\ \ ]_i c\alpha$ *and* $c[\ ]_i \longrightarrow [c]_i$*, for* $A \in V \setminus C$, $c \in C$, $\alpha \in (V \setminus C)^*$, *or*
    pairs of rules $cA[\ ]_i \longrightarrow [c\alpha]_i$ *and* $[c]_i \longrightarrow [\ \ ]_i c$ *for* $A \in V \setminus C$, $c \in C$, $\alpha \in (V \setminus C)^*$,
  - ◇ promoted carrier rules:
    pairs of rules $[cA]_i B \longrightarrow [\ ]_i c\alpha$ *and* $c[\ ]_i \longrightarrow [c]_i$*, for* $A, B \in V \setminus C$, $c \in C$, $\alpha \in (V \setminus C)^*$, *or*
    pairs of rules $cA[B]_i \longrightarrow [c\alpha]_i$ *and* $[c]_i \longrightarrow [\ ]_i c$*, for* $A, B \in V \setminus C$, $c \in C$, $\alpha \in (V \setminus C)^*$*;*
  - ◇ inhibited carrier rules:
    pairs of rules $[cA]_i \neg B \longrightarrow [\ ]_i c\alpha$ *and* $c[\ ]_i \longrightarrow [c]_i$*, for* $A, B \in V \setminus C$, $c \in C$, $\alpha \in (V \setminus C)^*$, *or*
    pairs of rules $cA[\neg B]_i \longrightarrow [c\alpha]_i$ *and* $[c]_i \longrightarrow [\ ]_i c$*, for* $A, B \in V \setminus C$, $c \in C$, $\alpha \in (V \setminus C)^*$*;*
- $i_0 \in \{1, \ldots, m\}$ *is an elementary membrane of* μ *(the output membrane).*

In a *simple rule* an object is rewritten in a string of objects, in the inner or outer region with respect to the initial object. A *promoted simple rule/inhibited simple rule* has the same action as a simple rule but it can be applied only in the presence/absence of certain objects (chemicals) called promoters/inhibitors. To be more precise we take as example the rule $[A]_i B \longrightarrow [\ ]_i\alpha$, which implies that object A is rewritten in α in the outer membrane only if *promoter* B is present there. If we replace B with $\neg B$, the object plays the role of the *inhibitor*, and only by its presence it blocks the execution of the rule.

In a *carrier rule* the objects can be rewritten only if they are guided by an object, the *carrier*. Note that the carrier is not actively participating in the reaction. Its role is to "accompany" the reaction and to inhibit the parallelism. As an

example, by rule $\left[cA\right]_i \longrightarrow \left[\ \right]_i c\alpha$ we mean that object A evolves to α (in the outer region of object A) iff there is an object c that helps A to be rewritten.

Promoted/Inhibited carrier rules can be applied if besides the carrier there is also a promoter/inhibitor which triggers/blocks the reaction.

As usual in membrane computing, the rules are used in a nondeterministic maximally parallel manner starting from an initial configuration. In this way, we obtain transitions between the configurations of the system. A configuration is described by the m-tuple of the multisets of objects present in the m regions of the system. The initial configuration is $(w_1, \ldots, w_m)$.

A sequence of transitions between configurations of the system constitutes a *computation*; a computation is successful if it *halts*, i.e., it reaches a configuration (the halting configuration) where no rule can be applied to any of the objects.

The *result* of a successful computation is the number of objects present within the membrane with the label i_o in the halting configuration. A computation which never halts yields no result.

We use the notation $NPACC_m(\alpha, \beta)$, where $\alpha \in \{smp\} \cup \{cat_k \mid k \geq 0\}$, $\beta \in \{proR_i, inhR_i\}$ to denote the family of sets of natural numbers generated by P systems with adjoining controlled communication rules having at most m membranes, communication rules that can be simple $\alpha = smp$, or carrier $\alpha = cat_k$, using at most k carriers, and external promoters $\beta = proR_i$ or external inhibitors $\beta = inhR_i$ of weight i at the level of rules.

3.2 An Example

Let us now exemplify the functioning of the model defined above throughout an **example**. Here it shown how such machines can be used to compute functions.

Consider the following system:

$$\Pi_1 = (\{A, B, D\}, C = \{c\}, [\ [\ ]_2]_1, w_1 = \{A^n\}, w_2 = \{c\}, R_1, R_2, 2),$$

where:

- R_1=$\emptyset$, R_2=$\{A\left[\ \right]_2 \longrightarrow \left[ABD\right]_2$, $\left[B\right]_2 \longrightarrow \left[\ \right]_2 B$, $\left[cD\right]_2 \longrightarrow \left[\ \right]_2 c$, $B\left[D\right]_2 \longrightarrow \left[AB\right]_2$, $c\left[\ \right]_2 \longrightarrow \left[c\right]_2\}$.

The system Π is fed with $n \geq 1$ copies of object A in region 1 and when it halts, the contents of the output region contains n^2 copies of A.

The functioning of the system is rather simple. The only rule we can apply in the initial configuration is the one which rewrites object A in ABD in the inner region, hence in the second step of the computation we will have all the objects of the system (n copies of A, n copies of B, n copies of D and the object initially present here, carrier c) in region 2. Then, we expel all objects B in region 1 and we start consuming objects D by applying the rule $\left[cD\right]_2 \longrightarrow \left[\ \right]_2 c$, hence object D is sent outside membrane 2 and is rewritten to λ having carrier c accompanying the reaction.

Note that object D plays the role of the counter and each time a copy of D is deleted (for example in step i of the computation), n more copies of A are produced (in step $i+2$ of the computation). One by one the n-th copies of D are consumed, adding for each of them n copies to object A. In the rule $B\big[D\big]_2 \longrightarrow \big[AB\big]_2$, object D plays also the role of promoter and object B can be rewritten into AB only in its presence. The computation ends with n^2 copies of A in region 2, hence the system computes the number-theoretic function $f(n) = n^2$, $n \geq 1$.

3.3 The Results

In what follows we will prove that the class of sets of numbers generated by P systems with external inhibitors equals the class of sets of numbers generated by P systems with external inhibitors and only two membranes.

Lemma 1. $NPACC_m(smp, inhR_1) = NPACC_2(smp, inhR_1), m \geq 2$.

Proof. Obviously, $NPACC_m(inh) \supseteq NPACC_2(inh)$. For the opposite inclusion we have to show that for any P system with external inhibitors $\overline{\Pi} = (\overline{V}, \overline{C}, \overline{\mu}, \overline{R}, \overline{i_0})$ generating a set of natural numbers, there exists an equivalent P system with external inhibitors $\Pi = (V, C, \mu, R, i_0)$ with only 2 membranes.

To this aim, we simulate the computation of $\overline{\Pi}$, with the system Π defined as follows.

Let us denote by $\mathcal{L} = \{1, 2, \ldots, m\}$ the set of labels of the regions in $\overline{\Pi_m}$. In addition, assume that $\overline{R} = \{R_1, \ldots, R_m\}$, and each $\overline{R_i} \in \overline{R}$, $1 \leq i \leq m$, contains all the rules that cross membrane i. Then, we define:

- $V = \{a_i \mid a \in \overline{V}, i \in \mathcal{L}\}$;
- $C = \overline{C} = \emptyset$;

Let $h : \overline{V}^* \times \mathcal{L} \to V^*$ be a mapping such that

1) $h(a, i) = a_i, a \in \overline{V}, i \in \mathcal{L}$,
2) $h(\lambda, j) = \lambda$, $j \in \mathcal{L}$,
3) $h(x_1x_2, j) = h(x_1, j)h(x_2, j)$, $x_1, x_2 \in \overline{V}^*$, $j \in \mathcal{L}$,

- denote $w = h(\overline{w_1})h(\overline{w_2}) \ldots h(\overline{w_m})$, where $\overline{w_i}$ is the multiset present in region $i \in \mathcal{L}$ of $\overline{\Pi_m}$ at the beginning of the computation.
- R is defined as follows.

  For each rule $A\big[\ \big]_i \longrightarrow \big[\alpha\big]_i \in R_i$, $A \in \overline{V}$, $\alpha \in \overline{V}^*$, $i \in \mathcal{L}$, we add to R the rule $h(A, j)\big[\ \big]_1 \longrightarrow \big[h(\alpha', i)\big]_1$, providing that j is the label of the outer membrane of membrane i.

  For each rule $A\big[\neg B\big]_i \longrightarrow \big[\alpha\big]_i \in R_i$, $A, B \in \overline{V}$, $\alpha \in \overline{V}^*$, $i \in \mathcal{L}$, we add to R the rule $h(A, j)\big[\neg h(B, i)\big]_1 \longrightarrow \big[h(\alpha', 2)\big]_1$, providing that j is the label of the outer membrane of membrane i.

  For each rule $\big[A\big]_i \longrightarrow \big[\ \ \big]_i\alpha \in R_i$, $A, B \in \overline{V}$, $\alpha \in \overline{V}^*$, $i \in \mathcal{L}$, we add to R the rule $\big[h(A, i)\big]_1 \longrightarrow \big[\ \ \big]_1 h(\alpha', j)$ providing that j is the outer membrane of membrane i.

For each rule $\left[A\right]_i \neg B \longrightarrow \left[\ \right]_i \alpha \in R_i$, $A, B \in \overline{V}$, $\alpha \in \overline{V}^*$, $i \in \mathcal{L}$, we add to R the rule $\left[h(A,i)\right]_1 \neg h(B,j) \longrightarrow \left[\ \right]_1 h(\alpha', j)$ providing that j is the outer membrane of membrane i.

Generally speaking, the purpose of membranes is to keep private the interior rules and objects from the neighboring ones and vice-versa. However, in our case we can express the passage of certain symbol through the membranes by using new symbols that we add to vocabulary and that encode both the crossed membrane label and the symbols from where they derive. In this way we can rewrite the rules, using the new symbols that perfectly describe the passage of objects in the membrane structure; consequently, in our case, we can shrink an arbitrarily membrane structure to only two membranes. The morphism used by the above construction accomplishes the encoding procedure.

The system Π simulates all the moves of $\overline{\Pi}$ and it stops whenever $\overline{\Pi}$ stops. However, in the halting configuration, in the designated output region of Π, there could be some objects representing the encoded version of the objects present in the regions of $\overline{\Pi}$. Therefore, we have to modify the above set of rules such that Π eliminates all these objects in order to generate the same set of numbers as $\overline{\Pi}$. This can be accomplish by producing an object D whenever a rule of $\overline{\Pi}$ is simulated (by adding the object D at the right hand side of each above rule), deleting it at each step (we add to R rules of type $D\left[\ \right]_1 \longrightarrow \left[\lambda\right]_1$ and $\left[D\right]_1 \longrightarrow \left[\ \right]_1 \lambda$). Finally, if $\overline{\Pi}$ stops, then Π will not produce the object D anymore, hence the absence of this object can trigger an inhibited rule that deletes all the unnecessary objects. Consequently, we have that $NPACC_m(smp, inhR_1) = NPACC_2(smp, inhR_1), m \geq 2$.

Here we will prove that the family of sets of vectors of numbers generated by P systems with external inhibitors equals the family of sets of numbers generated by ET0L systems.

Theorem 1. $NPACC_2(smp, inhR_1) = NET0L$.

Proof. We will prove the result by showing that communicative P systems with external inhibitors are equivalent with P systems with inhibitors, which at their turn, generates the same class of sets of numbers as the Parikh image of $ET0L$ as shown in [7]. Let $NP_1(smp, inhR_1)$ be the family of sets of numbers generated by P systems with inhibitors.

The proof of the inclusion $NP_1(smp, inhR_1) \supseteq NPACC_2(smp, inhR_1)$ is rather simple and is based on a similar encoding of regions into new objects as was presented above.

For the inclusion $NP_1(smp, inhR_1) \subseteq NPACC_2(smp, inhR_1)$ we will simulate the computation of a P system with one region $\Pi_{inh} = (V, C, \mu, w, R, i_0)$. We assume that the set of rules R contains rules of type $A \rightarrow \alpha$ or $A \rightarrow \alpha|_{\neg B}$, $A, B \in V$, $\alpha \in V^*$.

Let us consider the sets $\widetilde{V} = \{\widetilde{A} \mid A \in V\}$ and $\dot{V} = \{\dot{A} \mid A \in V\}$. In addition, let us define the morphisms:

$h_1 : V^* \to \widetilde{V}^*$, such that $h_1(A) = \widetilde{A}$ for all $A \in V$;
$h_2 : V^* \to \dot{V}^*$, such that $h_3(A) = \dot{A}$ for all $A \in V$.

We construct a P system $\Pi_{cc} = (\overline{V}, \overline{C}, \overline{\mu}, \overline{R}, \overline{i_0})$, simulating Π_{inh}, defined as follows:

$$\begin{array}{ll} \overline{V} = V \cup \widetilde{V} \cup \dot{V} \cup \{F\}; & w_1 = w; \\ C = \emptyset; & w_2 = w; \\ \mu = \left[\ \left[\ \right]_2 \right]_1; & i_0 = 1. \end{array}$$

The set of rules R is defined as follows[3]:

$$\begin{array}{rl} \text{step } i & A\big[\neg B\big] \longrightarrow \big[\mathbf{h_1}(\alpha)\mathbf{h_2}(\alpha)\big], \text{ for all rules } A \to \alpha|_{\neg B} \in R_{inh}, \\ \text{step } i & A\big[\ \big] \longrightarrow \big[\mathbf{h_1}(\alpha)\mathbf{h_2}(\alpha)\big], \text{ for all rules } A \to \alpha \in R_{inh}, \\ \text{step } i & \big[A\big] \longrightarrow \big[\ \big]F, \text{ if exists } A \to \alpha \in R_{inh}, \\ \text{step } i & \big[A\big]\neg B \longrightarrow \big[\ \big]F, \text{ if exists } A \to \alpha|_{\neg B} \in R_{inh}, \\ \text{step } i+1 & F\big[\ \big] \longrightarrow \big[\ \big], \\ \text{step } i+1 & \big[\mathbf{h_1}(A)\big] \longrightarrow \big[\ \big]\mathbf{h_1}(A), \text{ for all objects } A \in V, \\ \text{step } i+2 & \mathbf{h_1}(A)\big[\ \big] \longrightarrow \big[A\big], \text{ for all objects } A \in V, \\ \text{step } i+2 & \big[\mathbf{h_2}(A)\big]\neg R \longrightarrow \big[\ \big]A, \text{ for all } A \in V. \end{array}$$

Here is how the system Π_{cc} simulates the computation of Π_{inh}. First, remark that in order to correctly simulate the moves of Π_{inh}, we will maintain during the computation in both regions of Π_{cc} a copy of the multiset w – the multiset that represent the current configuration of Π_{inh}. This is especially useful when trying to simulate rules of type $A \to \alpha|_{\neg B} \in R_{inh}$ because we have to know whether or not the external inhibitor is present.

We assume that the system is in a configuration given by the strings $w_1 = w_2 = w$. The system attempts to execute simultaneously the rules of type

$$\begin{array}{rl} \text{step } i & A\big[\neg B\big] \longrightarrow \big[\mathbf{h_1}(\alpha)\mathbf{h_2}(\alpha)\big], \text{ for all rules } A \to \alpha|_{\neg B} \in R_{inh}, \\ \text{step } i & A\big[\ \big] \longrightarrow \big[\mathbf{h_1}(\alpha)\mathbf{h_2}(\alpha)\big], \text{ for all rules } A \to \alpha \in R_{inh}, \\ \text{step } i & \big[A\big] \longrightarrow \big[\ \big]F, \text{ if exists } A \to \alpha \in R_{inh}, \\ \text{step } i & \big[A\big]\neg B \longrightarrow \big[\ \big]F, \text{ if exists } A \to \alpha|_{\neg B} \in R_{inh}. \end{array}$$

Remark that the rules of first two types are used to generate inside the inner region, two copies of multiset α (represented by $\mathbf{h_1}(\alpha)$ and $\mathbf{h_2}(\alpha)$). In the same

[3] For the present proof, we will simplify the notation by not including the membrane labels into the syntax of the rules; this is possible here since we have only two membranes and we do not allow the interaction with the environment. In addition, we have specified on their left hand side the moment of their executions during the simulation of one computational step in Π_{inh}.

time, the rules of second type delete from region 2 the objects that were within the scope of rules of first type. In addition, remark that there are no other rules that can be applied in this step. Moreover, they produce in region 1 objects R; these objects will be used later for synchronizing the moments when multiset α appears in both regions.

Next, are executed the rules of type:

$$\begin{aligned}&\text{step } i+1 \quad F[\ ] \longrightarrow [\ ],\\ &\text{step } i+1 \quad [\mathbf{h_1}(A)] \longrightarrow [\ ]\mathbf{h_1}(A), \text{ for all objects } A \in V.\end{aligned}$$

Observe that the presence of object(s) R in this computational step inhibits the executions of rules of type $[\mathbf{h_2}(A)]\neg F \longrightarrow [\ ]A$, for all $A \in V$. Hence, in the third step, the rules of type

$$\begin{aligned}&\mathbf{h_1}(A)[\ ] \longrightarrow [A], \text{ for all objects } A \in V,\\ &[\mathbf{h_2}(A)]\neg F \longrightarrow [\ ]A, \text{ for all } A \in V,\end{aligned}$$

will be executed. The new objects appear at the same time in both regions of the system Π_{cc} and the simulation of the next computational step of Π_{inh} can start. Finally, if the system Π_{inh} stops because there are no rules to be applied, then also Π_{cc} halts.

Before we conclude, remark that the maximal parallelism as well as the universal clock is fundamental for the construction.

Consequently we have proved that the computation of an arbitrary P system with inhibitors can be simulated by a P system with external inhibitors, hence we have $NP_1(smp, inhR_1) \subseteq NPACC_2(smp, inhR_1)$. Therefore we have that $NP_1(smp, inhR_1) = NPACC_2(smp, inhR_1) = PsET0L$.

The following theorem shows that P systems with external inhibitors and carriers are computationally complete.

Theorem 2. $NPACC_2(cat, inhR_1) = NRE$.

Proof. The inclusion $NPACC_2(cat, inhR_1) \subseteq NRE$ is assumed true by invoking the Turing-Church thesis.

For the inclusion $NPACC_2(cat, inhR_1) \supseteq NRE$ we will simulate the computation of an arbitrary non-deterministic register machine $M = (n, \mathcal{P}, l_0, l_h)$. Such register machines are computational universal if $n \geq 3$.

We construct $\Pi = (V, C, \mu, w_1, w_2, R_1, i_0)$ as follows.

$$\begin{aligned}V &= \{a_i, A_i, S_i \mid 1 \leq i \leq n\} \cup \{l, \overline{l}, \overline{\overline{l}}, \widetilde{l}, \widetilde{\widetilde{l}}, L \mid l \in Lab(\mathcal{P})\} \cup \{c\}\\ &\quad \cup \{K, \overline{K}, \overline{\overline{K}}, \overline{\overline{\overline{K}}}, T_0, T_1, X, \overline{X}\};\\ C &= \{c\};\\ \mu &= [\ [\ ]_2\ ]_1;\end{aligned}$$

$$w_1 = l_0 L_0 a_1^{k_1} \dots a_n^{k_n} c;$$
$$w_2 = A_1 \dots A_n S_1 \dots S_n;$$
$$i_0 = 1.$$

The set of rules R is defined as follows:

• for each instruction $(l_1 : \mathrm{ADD}(j), l_2, l_3) \in \mathcal{P}$, the set R contains the rules:

$l_1[\ ] \longrightarrow [A_1 \dots A_{j-1} A_{j+1} \dots A_n S_1 \dots S_n a_j l_2]$, $l_1 \neq l_h$,

$l_1[\ ] \longrightarrow [A_1 \dots A_{j-1} A_{j+1} \dots A_n S_1 \dots S_n a_j l_3]$, $l_1 \neq l_h$,

$L_1[\neg A_j] \longrightarrow [A_1 \dots A_n S_1 \dots S_n]$,

$[l_2] \longrightarrow [\ ] l_2$,

$[l_3] \longrightarrow [\ ] l_3$,

$[a_j] \longrightarrow [\ ] a_j$,

$[A_i] \longrightarrow [\ ] \lambda$, $1 \leq i \leq n$,

$[S_i] \longrightarrow [\ ] \lambda$, $1 \leq i \leq n$;

• for each instruction $(l_1 : \mathrm{SUB}(r), l_2, l_3) \in \mathcal{P}$, the set R contains the rules:

$l_1[\ ] \longrightarrow [A_1 \dots A_n S_1 \dots S_{j-1} S_{j+1} \dots S_n \overline{l_1}]$, $l_1 \neq l_h$,

$ca_j[\neg S_j] \longrightarrow [A_1 \dots A_n S_1 \dots S_n X]$,

$L_1[\neg S_j] \longrightarrow [A_1 \dots A_n S_1 \dots S_n K]$,

$[\overline{l_1}] \longrightarrow [\ ] \overline{\overline{l_1}}$,

$[X] \longrightarrow [\ ] \overline{X}$,

$\overline{\overline{l_1}}[\ ] \longrightarrow [\overline{\overline{\overline{l_1}}} T_0 A_1 \dots A_n S_1 \dots S_n]$,

$[K] \longrightarrow [\ ] \overline{K}$,

$[\overline{\overline{\overline{l_1}}}] \neg X \longrightarrow [\ ] \widetilde{l_3}$,

$\overline{X}[\neg T_0] \longrightarrow [l_2]$,

$\overline{K}[\ ] \longrightarrow [A_1 \dots A_n S_1 \dots S_n \overline{\overline{K}}]$,

$[T_0] \longrightarrow [\ ] T_1$,

$[\overline{\overline{\overline{l_1}}}] \neg \overline{K} \longrightarrow [\ ] \lambda$,

$[l_2] \longrightarrow [\ ] l_2 L_2$,

$T_1[\ ] \longrightarrow [A_1 \dots A_n S_1 \dots S_n]$,

$\widetilde{l_3}[\ ] \longrightarrow [\widetilde{\widetilde{l_3}}]$,

$[\widetilde{\widetilde{l_3}}] \longrightarrow [\ ] l_3 L_3$,

$[\overline{\overline{K}}] \longrightarrow [\ ] \overline{\overline{\overline{K}}}$,

$\overline{\overline{\overline{K}}}[\ ] \longrightarrow [A_1 \dots A_n S_1 \dots S_n]$,

$[A_i] \longrightarrow [\ ]\lambda$, $1 \leq i \leq n$,

$[S_i] \longrightarrow [\ ]\lambda$, $1 \leq i \leq n$.

Here is how the P system Π simulates the computation of the register machine M. Observe for the beginning that in the P system Π we will represent the number stored into register j of M as the multiplicity of the object a_j. In addition, remark that objects A_j, S_j, $1 \leq j \leq n$, stand for the addition/subtraction command over register j – both in the simulation of an ADD or SUB instruction, the absence of symbol A_j or S_j allows the addition or deletion of one occurrence of object a_j. Objects A_j, S_j, $1 \leq j \leq n$, are produced all the time during the computation except the moment when we actually want to increment or subtract one occurrence of object a_j from the multiset; at that moment we generate all objects A_i, S_i, $1 \leq i \leq n$, such that $i \neq j$.

Let us see in more details how the simulation of the addition instruction $(l_1 : \mathrm{ADD}(j), l_2) \in \mathcal{P}$ works. Assume that at a certain moment during the computation, the current multisets in regions 1 and 2 are represented by the strings $w_1 = l_1 L_1 a_1^{k_1} \dots a_n^{k_n} c$ and $w_2 = A_1 \dots A_n S_1 \dots S_n$ respectively. Then, the rules that can be executed are:

$l_1[\ ] \longrightarrow [A_1 \dots A_{j-1} A_{j+1} \dots A_n S_1 \dots S_n a_j l_2]$ or the rule involving l_3,

$[A_i] \longrightarrow [\ ]\lambda$, $1 \leq i \leq n$,

$[S_i] \longrightarrow [\ ]\lambda$, $1 \leq i \leq n$.

As a consequence of executing the above rules the next configuration will be represented by $w_1 = L_1 a_1^{k_1} \dots a_n^{k_n} c$ and $w_2 = A_1 \dots A_{j-1} A_{j+1} \dots A_n S_1 \dots S_n a_j l_2$. Now, since in region 2 the object A_j is missing, then the rule

$L_1[\neg A_j] \longrightarrow [A_1 \dots A_n S_1 \dots S_n]$

can be executed; its role is to reestablish the initial configuration in region 2. Simultaneously, the system runs the rules

$[l_2] \longrightarrow [\ ]l_2$,

$[a_j] \longrightarrow [\ ]a_j$,

$[A_i] \longrightarrow [\ ]\lambda$, $1 \leq i \leq n$,

$[S_i] \longrightarrow [\ ]\lambda$, $1 \leq i \leq n$.

The rule $[l_2] \longrightarrow [\ ]l_2$ produces in region 1 the object l_2 that corresponds to register machine label l_2. In addition, by the execution of the rule $[a_j] \longrightarrow [\ ]a_j$, the number of objects a_j in region 1 (that corresponds to the number stored in register j of M) is incremented.

Concerning the simulation of the subtract instruction $(l_1 : \mathrm{SUB}(j), l_2, l_3) \in \mathcal{P}$, the system Π, being in a configuration represented by $w_1 = l_1 L_1 a_1^{k_1} \dots a_n^{k_n} c$ and $w_2 = A_1 \dots A_n S_1 \dots S_n$, executes first the rules:

$l_1[\ ] \longrightarrow [A_1 \dots A_n S_1 \dots S_{j-1} S_{j+1} \dots S_n \overline{l_1}]$,

$[A_i] \longrightarrow [\ ]\lambda$, $1 \leq i \leq n$,

$[S_i] \longrightarrow [\ ]\lambda$, $1 \leq i \leq n$.

In a similar manner as presented in the addition simulation, the rule $l_1[\ ] \longrightarrow [A_1 \dots A_n S_1 \dots S_{j-1} S_{j+1} \dots S_n \overline{l_1}]$ creates the context required for starting the simulation. The absence of object S_j in region 2 allows, in the second step, the (possible) execution of the rules

$ca_j[\neg S_j] \longrightarrow [A_1 \dots A_n S_1 \dots S_n X]$,

$L_1[\neg S_j] \longrightarrow [A_1 \dots A_n S_1 \dots S_n K]$.

Observe that in case there exists an object a_j in region 1, both rules are executed, while if there is not, only the rule $L_1[\neg S_j] \longrightarrow [A_1 \dots A_n S_1 \dots S_n K]$ will be executed.

In the same step, the rule $[\overline{l_1}] \longrightarrow [\ ]\overline{\overline{l_1}}$ performs. As we will see, the objects derived from object l_1 will be used later to check whether or not the rule $ca_j[\neg S_j] \longrightarrow [A_1 \dots A_n S_1 \dots S_n X]$ was executed. Moreover, they will be also used to introduce in region 2 objects $A_1 \dots A_n S_1 \dots S_n$ that forbids a new addition or subtraction of objects a_j.

Let us consider the first case, i.e. the region 1 contains at least one object a_j. Then, as a consequence of executing the above rules we will have the multisets $w_1 = a_1^{k_1} \dots a_j^{k_j - 1} \dots a_n^{k_n} c$ and $w_2 = A_1^2 \dots A_n^2 S_1^2 \dots S_n^2 X K$. The following rules will be further applied:

$\overline{\overline{l_1}}[\ ] \longrightarrow [\overline{\overline{l_1}} T_0 A_1 \dots A_n S_1 \dots S_n]$,

$[K] \longrightarrow [\ ]\overline{K}$,

and possibly the rule:

$[X] \longrightarrow [\ ]\overline{X}$.

Remark that the objects derived from $\overline{l_1}$ are within the scope of rules that introduce at each odd step objects $A_1 \cdots A_n S_1 \dots S_n$ (or $A_1 \dots A_{j-1} A_{j+1} \dots A_n S_1 \cdots S_n$ in the first step). In a similar manner the objects derived from K are within the scope of rules that introduce at each even step objects $A_1 \cdots A_n S_1 \dots S_n$. Anyway, at each step we delete by rules $A_i \to \lambda$ and $S_i \to \lambda$, $1 \leq i \leq n$ all objects A_i and S_i.

Now, since in the third step an object $\overline{X}$ was introduced in region 1 then, in the fourth step, the rule $[\overline{\overline{l_1}}]\neg X \longrightarrow [\ ]\widetilde{l_3}$ cannot be executed. Moreover, because in region 2 exists an object T_0 also the rule $\overline{X}[\neg T_0] \longrightarrow [l_2]$ cannot be executed. However, in the fourth step the rule $[T_0] \longrightarrow [\ ]T_1$ runs and it will allow, in the fifth step, the execution of the rule $\overline{X}[\neg T_0] \longrightarrow [l_2]$. In the same time, rule $[\overline{\overline{l_1}}]\neg \overline{K} \longrightarrow [\ ]\lambda$ is executed and so there will be no way to rewrite $\overline{\overline{l_1}}$ into $\widetilde{l_3}$ and furthermore into l_3. Finally, by rule $[l_2] \longrightarrow [\ ]l_2 L_2$ the label of the new register machine instruction to be simulated is generated.

Now let us see what how the simulation is done when the system Π attempts to simulate the instruction $(l_1 : \mathrm{SUB}(j), l_2, l_3) \in \mathcal{P}$ in the case when the register j is empty. Then, the simulation works in a similar manner as in the above presented

case with the main difference being that in the fourth step the rule $\left[\overline{\overline{l_1}}\right]\neg X \longrightarrow \left[\ \right]\widetilde{l_3}$ is executed because the object $\overline{X}$ was not produced (the rules $ca_j\left[\neg S_j\right] \longrightarrow \left[A_1 \dots A_n S_1 \dots S_n X\right]$ and $\left[X\right] \longrightarrow \left[\ \right]\overline{X}$ were not ran since the object a_j was missing from the initial multiset). So, the following rules are executed in sequence $\widetilde{l_3}\left[\ \right] \longrightarrow \left[\widetilde{\widetilde{l_3}}\right]$, $\left[\widetilde{\widetilde{l_3}}\right] \longrightarrow \left[\ \right]l_3 L_3$. As a consequence, the symbol that corresponds to the next instruction to be simulated is generated.

If l_h is generated then the computation stops, having in the output region a number of objects a_i, $1 \leq i \leq n$, equals with the contents of register i of M. In this way the execution of the entire register machine program is simulated.

Since one can easily construct a register machine, equivalent with M, that in a successful computation clears its registers except a special designated one (the output register) we have that $NPACC_2(cat, inhR_1) \supseteq NRE$.

Therefore, we have proved the equality $NPACC_2(cat, inhR_1) = NRE$.

4 Conclusions and Further Research

The model we introduced is based on the observation that various chemical reactions within a compartment of a living cell are activated from the neighboring compartments of the cell. We have proved that the family of sets of vectors of numbers generated by P systems with adjoining controlled communication rules when only simple inhibited rules are used equals the family of sets of numbers generated by ET0L systems. We have also proved the computational completeness if, in addition, carriers are used. As a plus, we want to emphasize that similar results can be obtained if, instead of inhibited simple rules, promoted ones are considered.

Trying to get more "realistic", we believe that it is worthwhile to investigate the power of the above systems to whom we add execution times for the rules and to study their properties (for more details we refer to [2]). Another possible line for further research is to investigate the power of the systems not considering the family of sets of vectors of numbers generated as we have done here, but considering the family of Parikh images generated by such systems.

References

1. C. Calude, Gh. Păun, G. Rozenberg, A. Salomaa (Eds.): *Multiset Processing*, LNCS 2235, Springer-Verlag, Berlin, 2001.
2. M. Cavaliere, D. Sburlan: Time and Synchronization in Membrane Systems, *Fundamenta Informaticae* 64(1–4), 65–77, 2005.
3. M. Ionescu, D. Sburlan: On P Systems with Promoters/Inhibitors, *Journal of Universal Computer Science*, 10(5), 581–599, 2004.
4. Gh. Păun: Computing with Membranes, *Journal of Computer and System Sciences*, 618(1), 108–143, 2000.

5. Gh. Păun: *Membrane Computing. An Introduction*, Springer-Verlag, Berlin, 2002.
6. A. Salomaa, G. Rozenberg (Eds.): *Handbook of Formal Languages*, Springer-Verlag, Berlin, 1997.
7. D. Sburlan: Further Results on P Systems with Promoters/Inhibitors. *International Journal of Foundations of Computer Science*, 17, 1 (2006), 205–221;
8. http://psystems.disco.unimib.it/