On the Expressive Power of Membrane Systems Working in Accepting Mode

Roberto Barbuti[1], Andrea Maggiolo-Schettini[1], Paolo Milazzo[1], and Simone Tini[2]

[1] Dipartimento di Informatica, Università di Pisa
Largo Pontecorvo 3, 56127 Pisa, Italy. Email: {*barbuti,maggiolo,milazzo*}*@di.unipi.it*
[2] Dipartimento di Informatica e Comunicazione, Università dell'Insubria
Via Carloni 78, 22100 Como, Italy. Email: *simone.tini@uninsubria.it*

Abstract. Membrane systems can be seen either as generators or as acceptors of multiset languages. The aim of this paper is to study the expressive power of membrane systems working in accepting mode, and to compare the results with those on membrane systems working in generating mode, which are mostly well-known in the literature. As regards the expressive power, determinism, presence of promoters and of cooperative rules are considered.

1 Introduction

Membrane systems (P systems) were introduced by Paun in [10] as distributed parallel computing devices inspired by the structure and the functioning of cells. In the extension of [3] the application of rules may be conditioned by the presence of *promoter objects*. A promoter does not participate in the application of rules, and a single promoter may enable the application of several rules and multiple applications of each one of these rules. P systems with promoters have been shown to be universal even when non-cooperative rules are considered [3]. The same holds for P systems without promoters, but with cooperative rules [11].

In universality proofs P systems based on multiset rewriting rules (transition P systems) are usually seen as multiset generators. A computation of a P system gives as output a multiset, and the set of all multisets given as output by different computations of a system are taken as the multiset language generated by the system. An alternative view considers P systems as multiset acceptors. Given a multiset as input, the computation of a P system may take to an accepting state. The set of accepted multisets constitutes the multiset language accepted by the P system. In this paper we are interested in studying the expressive power of P systems working in accepting mode. In particular, we consider classes of P systems that are discriminated by admitting, or not, promoters, cooperative rules and nondeterminism.

Different notions of P systems working in generating and accepting modes have been considered, for instance in [9, 12] from the point of view of complexity. A variant of P systems working in accepting mode and with symport/antiport rules, called P automata, have been introduced in [4, 8]. Generating and accepting modes have been considered and compared also in the context of regulated rewriting [5–7].

2 Membrane Systems with Promoters

In this section we recall the definition of P systems with promoters and introduce notions of P systems accepting and generating multiset languages.

2.1 Definition

A membrane system, also called P system, consists of a *hierarchy of membranes* that do not intersect, with a distinguishable membrane, called the *skin membrane*, surrounding them all. As usual, we assume membranes to be labeled by natural numbers. Given a set of objects V, a membrane m contains a multiset of *objects* in V, a set of *evolution rules*, and possibly other membranes, called *child* membranes (m is also called the *parent* of its child membranes). A rule in a membrane m can be applied only to objects in m. The rule must contain target indications, specifying the membranes where each object produced by applying the rule is sent. The new objects either remain in m, or can be sent out of m, or can be sent into one of its child membranes, precisely identified by its label. Formally, the products of a rule are denoted with a set of pairs of the following forms:

– $(v, here)$, meaning that the multiset of objects v produced by the rule remain in the same membrane m;
– (v, out), meaning that the multiset of objects v produced by the rule are sent out of m;
– (v, in_l), meaning that the multiset of objects v produced by the rule are sent into the child membrane l.

An evolution rule may have some *promoters* that are objects required to be present in the membrane m in order to enable the application of the rule. We can assume that all evolution rules have the following form:

$$u \to (v_h, here)(v_o, out)(v_1, in_{l_1}) \dots (v_n, in_{l_n})|_p$$

where u is the multiset of objects consumed by the rule, $\{l_1, \dots, l_n\}$ is a set of membrane labels, $v_h, v_o, v_1, \dots, v_n$ are the objects (grouped in multisets by target) produced by the rule, and p is the multiset of promoters of the rule. Notice that all objects mentioned in the rules are in V, therefore an object may appear in the reactants, in the products, and in the promoters of the rules. The size of the left–hand side u of an evolution rule is called the *radius* of such a rule. If a P System contains rules of radius greater than one, then it is called a *cooperative* system. Otherwise, it is called *non–cooperative.*

Application of evolution rules is done with maximal parallelism, namely at each evolution step a multiset of instances of evolution rules is chosen non–deterministically such that no other rule can be applied to the system obtained by removing all the objects necessary to apply all the chosen rules. The application of rules consists of removing all the reactants of the chosen rules from the system and adding the products of the rules by taking into account the target indications. Promoters are not consumed by the application of the corresponding evolution rule, thus implying that the presence of a single occurrence of a promoter can enable the application of more than one rule in each maximally parallel evolution step.

A P system has a tree-structure in which the skin membrane is the root and the membranes containing no other membranes are the leaves. We assume membranes labels to be unique: they are assigned at the beginning of the evolution by counting the membranes encountered during a breadth-first visit of the tree–structure, with 1 as the label of the skin membrane.

Now, we formally define P systems with promoters.

Definition 1. *A* membrane system Π *is given by*

$$\Pi = (V, \mu, w_1, \ldots, w_n, R_1, \ldots, R_n)$$

where:

- V *is an* alphabet *whose elements are called* objects*;*
- $\mu \subset \mathbb{N} \times \mathbb{N}$ *is a* membrane structure*, such that* $(l_1, l_2) \in \mu$ *denotes that the membrane labeled by* l_2 *is contained in the membrane labeled by* l_1*;*
- w_j *with* $1 \leq j \leq n$ *are strings from* V^* *representing multisets over* V *associated with the membranes* $1, \ldots, n$ *of* μ*;*
- R_j *with* $1 \leq j \leq n$ *are finite sets of* evolution rules *associated with the membranes* $1, \ldots, n$ *of* μ*.*

In this paper we assume P systems to be closed computational devices, namely we assume that objects cannot be sent out of the skin membrane (i.e.rules sending objects out are not allowed in the skin membrane) and objects cannot be received by the skin membrane from outside.

A *computation* of a P system is a sequence of maximally parallel evolution steps. A computation is *valid* if and only if the sequence of computation steps is finite and leads to a *final configuration*, namely a configuration in which no evolution rules can be further applied.

A P system is said to be *deterministic* if it may perfom only one computation (either valid or not). This happens when at each step there is only one maximal multiset of applicable evolution rules.

We show in Figure 1 an example of P system in which all the main features of the formalism are used. In the figure, membranes are depicted as boxes containing evolution rules, objects and inner membranes. The label of a membrane is at a corner of the corresponding box. Exponents are used to have a compact representation of multiple occurrences of an object in a multiset. For example,

$a^n p^k b$ represents the multiset consisting of n occurrences of a, k occurrences of p and one occurence of b. Symbol λ denotes the empty multiset.

The P system in the figure performs a computation consisting of k steps. At each step the number of a's is halved and one p is sent into membrane 2, where it is cancelled. The output of the computation is hence $a^{\lceil \frac{n}{2^k} \rceil} q$.

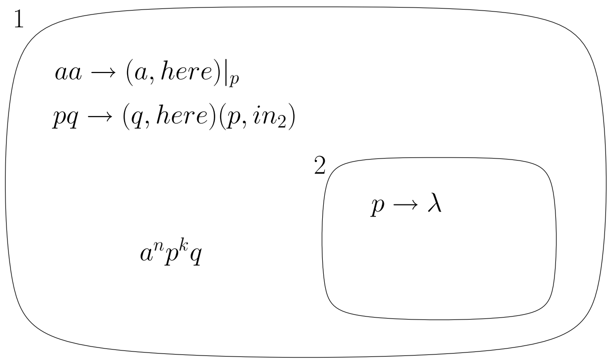

Fig. 1. An example of P system with promoters.

Let us assume that V is partitioned into sets Σ and $\mathcal{C}$, where $\mathcal{C}$ is called as the set of *control objects*. From [1] it follows that any P system with promoters Π can be translated into an equivalent P system Π' having a (flat) membrane structure that consists only of the skin. The idea is to obtain the alphabet of the control objects in Π' by enriching the alphabet of the control objects in Π with objects labeled with indexes of membranes in Π to represent objects of Π that are placed in some inner membrane. It turns out that Π and Π' are equivalent in the sense that each of them can mimic the behavior of the other, evolution step by evolution step. More precisely, the skin membranes of Π and Π' contain the same multisets over Σ after each computation step. An analogous technique was previously used in [2, 13] but with different classes of P systems. As an example, we show, in Figure 2, the result of flattening the P system with promoters given in Figure 1.

For the sake of precision, the class of P systems considered in [1] is slightly different: (i) promoters of an evolution rule are given as a set rather than a multiset, (ii) membranes are enriched with an interface that filters the objects that can be received from the external environment, and (iii) also rule inhibitors and dissolving rules are considered. As regards (i), it is easy to see that the way in which promoters are given does not influence the flattening technique. As regards (ii), membrane interfaces are used in [1] to ensure compositionality. Here, we consider closed computational systems, hence we do not allow interactions with the external environment. As regards (iii), it is easy to see that inhibitors and dissolving rules are not introduced by the flattening technique, hence the

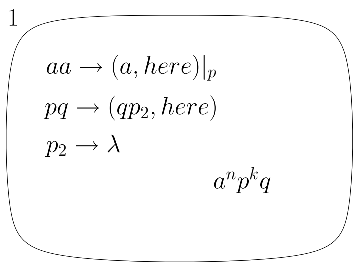

Fig. 2. The result of flattening the P system given in Figure 1.

flat version of a P system of the class we consider here is still in the same class. To sum up, we have that the following result holds.

Theorem 1. *Every P system with promoters can be translated into an equivalent P system whose membrane structure consists only of the skin membrane.*

Proof. Follows from results in [1]. □

If one gives a suitable notion of size of a P system taking into account the cardinality of the set of objects and of the set of rules, one can infer that the flattening technique in [1] takes a P system of size s to a P system of size $o(s)$.

We shall always assume flat P systems in our proofs, consequently we shall always assume the *here* particle for products of evolution rules (e.g. we will write $a \to b$ for $a \to (b, here)$).

In the following we shall study how determinism, presence of promoters, and cooperative rules have an influence on the expressive power of P systems. We shall use the following notations for the different classes of P systems: $P(coo, ndet, pro)$ denotes the class of P systems admitting cooperation, non determinism and promoters, we replace *coo* with *ncoo* in the classes where cooperative rules are not admitted, *ndet* with *det* in the classes where nondeterminism is not admitted, and *pro* with *npro* in the classes where promoters are not admitted.

2.2 Membrane Systems and Multiset Languages

P systems can deal with multiset languages. A multiset language [4] is a set of multisets over a given alphabet. It might be obtained from a language of strings by applying the Parikh mapping to each string in the language. The Parikh mapping takes a string into a vector of natural numbers, in which each element corresponds to the number of occurrences in the string of one of the alphabet symbols. After application of the Parikh mapping, information on the ordering of symbols in the strings of the language is lost.

Among the most important classes of multiset languages we mention the class obtained by applying the Parikh mapping to all context free languages of strings, denoted $PsCF$, that coincides with the class obtained by applying the Parikh mapping to all regular languages, denoted $PsREG$. We mention also the class obtained by applying the Parikh mapping to all recursively enumerable languages of strings, denoted $PsRE$. Note that $PsRE$ is the class of all recursively enumerable sets of multisets (represented as sets of vectors of natural numbers).

A P system can be used either as an acceptor or as a generator of a multiset language over Σ. In the first case, a multiset over Σ is inserted in the skin membrane of the P system and the result of its computations says whether such a multiset belongs to the multiset language accepted by the P system or not. In the second case the P system has a fixed initial configuration and can give as results (possibly in a non-deterministic way) all the possible multisets belonging to a given multiset language.

Let us formalise the notion of P system used as language acceptor.

Definition 2. *A* flat acceptor P system *over an alphabet Σ is a P system $\Pi = (\Sigma \cup \mathcal{C} \cup \{T\}, \emptyset, w_1, R_1)$, where:*

- *$\mathcal{C}$ is a set of* control objects *such that $\Sigma \cap \mathcal{C} = \varnothing$;*
- *T is a special object not contained in $\Sigma \cup \mathcal{C}$;*
- *w_1 is a multiset of objects in $\mathcal{C}$;*
- *A multiset w of objects over Σ is accepted by Π iff when we add w to w_1 then a final configuration can be reached with T appearing in the membrane.*

We remark that one could define equivalent notions of acceptor P systems by assuming that a multiset is accepted if and only if a final configuration can be reached (by ignoring the presence of T). We can simulate this simply by adding T to w_1 and by ensuring that there is no rule in R_1 using such a special object.

We denote the language accepted by a P system Π as $Ps(\Pi)$ (as Parikh set). Moreover, we denote the set of languages accepted by a class of P systems by adding prefix Ps and subscript a in the notation of the class itself as in the following example: we use $PsP_a(ncoo, ndet, pro)$ to denote the set of languages for which there exists an acceptor in the class $P(ncoo, ndet, pro)$.

Now we formalise the notion of P system used as language generator.

Definition 3. *A* flat generator P system *over an alphabet Σ is a P system $\Pi = (\Sigma \cup \mathcal{C}, \emptyset, w_1, R_1)$, where:*

- *$\mathcal{C}$ is a set of* control objects *such that $\Sigma \cap \mathcal{C} = \varnothing$;*
- *w_1 is a multiset of objects in $\mathcal{C}$;*
- *a multiset w of objects over Σ is generated by Π if and only if there exist a multiset $w_o^{\mathcal{C}}$ of objects over $\mathcal{C}$ and a final configuration that can be reached having $w \cup w_o^{\mathcal{C}}$ as multiset of objects.*

Note that also in the case of generator P systems there exist other equivalent definitions that could be considered (and that have been considered in the literature). For instance, one could use a special membrane to collect the output of the system, or could send the output out of the skin membrane.

$$\begin{array}{cccccccc}
 & PsP_a(coo, ndet, pro) & & PsP_a(coo, ndet, npro) & & & & PsP_a(ncoo, ndet, pro) \\
 & \| \Uparrow & & \| \Uparrow & & & & \| \Uparrow \\
PsRE =^* & PsP_g(coo, ndet, pro) & =^* & PsP_g(coo, ndet, npro) & =^* & & & PsP_g(ncoo, ndet, pro) \\
 & & & & & & & \cup^* \\
 & & & & & & & PsP_g(ncoo, ndet, npro) \\
 & & & & & & & \cup \\
 & & & & & & & \mathcal{L}_1 \\
 & & & & & & & \| \\
 & & & & & \mathcal{L}_3 & & PsP_a(ncoo, ndet, npro) \\
 & & & & & \| & & \| \\
PsRE & = PsP_a(coo, det, pro) & \overset{\Leftrightarrow}{=} & PsP_a(coo, det, npro) & \supset & PsP_a(ncoo, det, pro) & \supset & PsP_a(ncoo, det, npro) \\
 & \cup & & \cup & & \cup & & \cup \\
\mathcal{L}_2 & = PsP_g(coo, det, pro) & = & PsP_g(coo, det, npro) & = & PsP_g(ncoo, det, pro) & = & PsP_g(ncoo, det, npro)
\end{array}$$

Fig. 3. Relationships among classes of languages accepted and generated by the classes of P systems considered in this paper.

As for acceptor P systems, we denote the language generated by a P systems Π as $Ps(\Pi)$, and we introduce a notation for the set of languages generated by P systems of a certain class. In this case we replace subscript a with subscript g as in the following example: we use $PsP_g(ncoo, ndet, coo)$ to denote the set of languages for which there exists a generator in the class $P(ncoo, ndet, coo)$.

3 On the Power of Cooperation, Promoters and Non-determinism in Acceptor P Systems

In this section we study the expressiveness of several classes of acceptor P Systems. Our results are summarized in Figure 3, where we report original results over acceptor P systems and we recall results over generator P Systems that are well known in the literature (see, for instance, [3] and [11]). Results known in the literature are marked with "$*$". Classes $\mathcal{L}_1, \mathcal{L}_2$ and $\mathcal{L}_3$ will be defined later.

The proofs of some results will consist in the definition of an encoding of a class of P systems C into another class C'. In such a case, if $\mathcal{L}$ is the set of multiset languages that are accepted (resp. generated) by P Systems in C, and $\mathcal{L}'$ is the set of multiset languages that are accepted (resp. generated) by P Systems in C', we write $\mathcal{L} \Rightarrow \mathcal{L}'$. Moreover, we write $\mathcal{L} \Leftrightarrow \mathcal{L}'$ when both $\mathcal{L} \Rightarrow \mathcal{L}'$ and $\mathcal{L}' \Rightarrow \mathcal{L}$.

Our first three theorems (Thms. 2, 3, 4) show that $PsP_g(ncoo, ndet, pro) \Rightarrow PsP_a(ncoo, ndet, pro)$, $PsP_g(coo, ndet, pro) \Rightarrow PsP_a(coo, ndet, pro)$, and, finally, $PsP_g(coo, ndet, npro) \Rightarrow PsP_a(coo, ndet, npro)$. This implies the universality of these three classes of acceptors. If one gives a suitable notion of *size* of a P system taking into account the cardinality of the set of objects and of the set of rules, one can infer that these encodings take a generator of size s to an acceptor of size $o(s)$. Formal proofs of these results are omitted here for lack of space.

In the following, given a multiset of objects v, we shall write v' to denote the multiset $\{a' \mid a \in v\}$, v'' to denote the multiset $\{a'' \mid a \in v\}$, and so on.

Theorem 2. *$PsP_g(ncoo, ndet, pro) \Rightarrow PsP_a(ncoo, ndet, pro)$.*

Proof. Given any generator Π in $P(ncoo, ndet, pro)$, we construct an equivalent acceptor Π_a in $P(ncoo, ndet, pro)$. By Thm. 1 we can assume that Π is flat. Let $\Pi = (\Sigma \cup C, \emptyset, w, R)$. Also Π_a will be flat, of the form $(\Sigma_a \cup C_a \cup \{T\}, \emptyset, w_a, R_a)$. The idea is that Π_a embeds Π and, for any input multiset u for Π_a, we exploit Π to generate a multiset v in $Ps(\Pi)$ and, then, we compare u with v: if they coincide then Π_a accepts u. The non-determinism ensures that for every $u \in Ps(\Pi)$ there is an execution by Π_a accepting it.

Actually, when Π is embedded into Π_a, all initial objects in w and all objects mentioned in the rules in R are primed, in order to distinguish them from the input of Π_a, and are considered as control objects. Hence, we have that $\Sigma_a = \Sigma$, $C_a \supseteq \Sigma' \cup C'$ and $w_a \supseteq w'$. Then, in the construction of Π_a we have to face two problems. The first problem is that we must be able to check for termination of Π in order to ensure to compare with the input of Π_a an actual multiset in $Ps(\Pi)$ rather than something that is the result of a partial execution of Π. The second problem is that we must implement the comparison.

To these purposes, let us add two fresh control objects to C_a: s, which triggers the comparison, and g, which is initially in w_a, is produced by all rules derived from Π and prevents the production of s. So, first of all R_a contains all rules:

$$R_\Pi = \{a' \to v'g|_{p'} \mid a \to v|_p \text{ is a rule in } R\}$$

Then, R_a contains the following set of rules, denoted R_1, where also x, x', 1, 2 are fresh objects in C_a and x and 1 are also in w_a:

$$\begin{array}{ccc} x \to x'|_{1g} & x \to s|_2 & x' \to x|_2 \\ g \to \lambda & 1 \to 2 & 2 \to 1|_{x'} \end{array}$$

Sets R_Π and R_1 are such that some instances of g are in Π_a as long as rules in R_Π are applied. When no rule in R_Π can be applied, which simulates the termination of the execution by Π, no occurrence of g is anymore in Π_a. This makes the rule $x \to x'|_{1g}$ no longer applicable. Notice that Π_a either contains both 1 and x, or it contains both 2 and x'. In the former case, since $x \to x'|_{1g}$ cannot be applied and 2 is produced by $1 \to 2$, after one computation step s is produced by $x \to s|_2$ and the comparison is triggered. In the latter case, x and 1 are produced by $x' \to x|_2$ and $2 \to 1|_{x'}$, respectively, and we come back to the previous case. Finally, let us add to C_a also the objects $\overline{0}, \overline{1}, \overline{2}$ and $\overline{3}$ and the set $\hat{C}$ consisting of the capitalized versions of the symbols in $\Sigma \cup \Sigma'$. Moreover, let us add $\overline{0}$ and T to w_a. The comparison of the multiset generated by the rules in

R_Π with the input is performed by the following set of rules (denoted R_2):

$$\overline{0} \to \overline{1}|_s \qquad \overline{1} \to \overline{2} \qquad \overline{2} \to \overline{3} \qquad \{\,\overline{3} \to \overline{1}|_{Taa'} \mid a \in \Sigma\}$$

$$\{\,a \to a|_{\overline{1}} \mid a \in \Sigma\,\} \qquad \{\,a \to A|_{\overline{1}} \mid a \in \Sigma\,\}$$
$$\{\,a' \to a'|_{\overline{1}} \mid a \in \Sigma\,\} \qquad \{\,a' \to A'|_{\overline{1}} \mid a \in \Sigma\,\}$$

$$\{\,T \to \lambda|_{AB\overline{2}} \mid A, B \in \hat{C}\,\} \qquad \{\,T \to \lambda|_{A'B'\overline{2}} \mid A', B' \in \hat{C}\,\}$$

$$\{\,T \to \lambda|_{A\overline{3}} \mid A \in \hat{C}\,\} \qquad \{\,T \to \lambda|_{A'\overline{3}} \mid A' \in \hat{C}\,\} \qquad \{\,T \to T|_{AA'\overline{3}} \mid A, A' \in \hat{C}\,\}$$
$$\{\,A \to \lambda|_{\overline{3}} \mid A \in \hat{C}\,\} \qquad \{\,A' \to \lambda|_{\overline{3}} \mid A' \in \hat{C}\,\}$$

Objects $\overline{0}, \overline{1}, \overline{2}$ and $\overline{3}$ are used to sequentialize different phases of the comparison. In particular, the rule consuming $\overline{0}$ starts the comparison, and it is triggered by s when the rules in R_Π are no longer applicable.

Rules promoted by $\overline{1}$ transform a non-deterministically chosen portion of the objects of the input multiset and of the multiset generated by R_Π into their capitalized version. Rules promoted by $\overline{2}$ check that at most one object of each of the two multisets to be compared has been capitalized, otherwise they replace T with λ. Rules promoted by $\overline{3}$ check that at least one object of each of the two multisets to be compared has been capitalized, and that such two objects are one the primed version of the other. Moreover, the capitalized objects are deleted. These three phases remove one object from the input multiset and the corresponding primed one from the multiset generated by R_Π, and they are repeated until there are pairs of corresponding objects in the two multisets and T has not been removed. Such a control is performed by the promoters of rules consuming $\overline{3}$.

Summarizing, $\Sigma_a = \Sigma$, $C_a = \Sigma' \cup C' \cup \hat{C} \cup \{\overline{0}, \overline{1}, \overline{2}, \overline{3}, x, x', 1, 2, g, s\}$, $w_a = w' \cup \{\overline{0}, 1, x, g, T\}$ and $R_a = R_\Pi \cup R_1 \cup R_2$. □

Theorem 3. *$PsP_g(coo, ndet, pro) \Rightarrow PsP_a(coo, ndet, pro)$.*

Proof. Following the proof of Thm. 2, given any generator Π in $P(coo, ndet, pro)$ we construct an acceptor Π_a in $P(coo, ndet, pro)$ that embeds Π. The acceptor Π_a can be constructed exactly as in the proof of Thm. 2. However, since here we can exploit cooperative rules, the set of rules R_2 used in the proof of Thm. 2 to implement the comparison between the input of Π_a and the multiset generated by R_Π can be replaced by the following rules:

$$\{aa' \to \lambda|_s \mid a \in \Sigma\} \cup \{aT \to \lambda|_s \mid a \in \Sigma\} \cup \{a'T \to \lambda|_s \mid a \in \Sigma\}$$

In this case the comparison requires only one computation step. □

Theorem 4. *$PsP_g(coo, ndet, npro) \Rightarrow PsP_a(coo, ndet, npro)$.*

Proof. Also in this case we exploit the generator Π in $P(coo, ndet, npro)$ to build an equivalent acceptor Π_a in $P(coo, ndet, npro)$. By Thm. 1 we can assume that Π is flat. Let $\Pi = (\Sigma \cup C, \emptyset, w, R)$. Also Π_a will be flat, of the form $(\Sigma_a \cup C_a \cup \{T\}, \emptyset, w_a, R_a)$. As in the proof of Thm. 2, we rename all objects in Π so that $\Sigma_a = \Sigma$, $C_a \supseteq \Sigma' \cup C'$, $w_a \supseteq w'$, and we introduce a fresh control object s in C_a triggering the comparison between the input multiset of Π_a and the multiset generated by the rules in Π_a derived from those in Π. Here s is triggered by another control object $t \in C_a$, trough the rule

$$t \to rs$$

where also r is a fresh object in C_a. The idea is that t is initially in w_a and the other rules in R_a will ensure that in all computations leading to a final configuration with T in the membrane, then this rule fires only after the rules derived from those in Π have generated their multiset.

Acceptor Π_a performs a loop with 3 steps, until s is produced from t by rule $t \to rs$. The sequence of these 3 steps simulates a single computation step by Π. At the first step in the loop, the following set of rules R^1_Π may fire:

$$\{u' \to v''v''', tu' \to v''v'''t' \mid u \to v \text{ is a rule in } R\}$$

Firing $tu' \to v''v'''t'$ prevents firing $t \to rs$ and, as a consequence, the production of s. Object t' is in C_a and serves to produce t once more.
At the second step, the following set of rules R^2_Π may fire:

$$\{a'''rT \to \lambda \mid a \in \Sigma\} \cup \{t' \to t''\} \cup \{a'' \to a'''' \mid a \in \Sigma\}$$

The rules in the first set check that s has not been produced (note that s can be produced only together with r) if the computation by Π has not terminated yet. More precisely, if s has been already produced and the computation by Π has not terminated yet, T is removed.
At the third step the following set of rules R^3_Π may fire:

$$\{a'''a'''' \to a' \mid a \in \Sigma\} \cup \{t'' \to t\}$$

so that the first step can begin once more.
When Π_a exits from the loop, which simulates the termination by Π, s can be exploited for the comparison, which is implemented by the following set of rules R_1:

$$\{saa' \to s \mid a \in \Sigma\} \cup \{saT \to \lambda \mid a \in \Sigma\} \cup \{sa'T \to \lambda \mid a \in \Sigma\}$$

Summarizing, $\Sigma_a = \Sigma$, $C_a = \{a', a'', a''', a'''' \mid a \in \Sigma \cup C\} \cup \{r, s, t, t', t''\}$, $w_a = w' \cup \{t, T\}$, $R_a = \{t \to rs\} \cup R^1_\Pi \cup R^2_\Pi \cup R^3_\Pi \cup R_1$. $\square$

From our first three theorems, the following results follow.

Corollary 1. *It holds that:*

- $PsP_g(ncoo, ndet, pro) \subseteq PsP_a(ncoo, ndet, pro)$,
- $PsP_g(coo, ndet, pro) \subseteq PsP_a(coo, ndet, pro)$,
- $PsP_g(coo, ndet, npro) \subseteq PsP_a(coo, ndet, npro)$.

Let us prove now that if we admit neither promoters nor cooperative rules, then nondeterministic acceptors and deterministic acceptors have the same expressive power, and are less expressive than nondeterministic generators. To this purpose, we characterize both $PsP_a(ncoo, ndet, npro)$ and $PsP_a(ncoo, det, npro)$.

Given a set of objects Σ and $A, N \subseteq \Sigma$, let $L_{A,N}$ and L_N denote the following multiset languages:

$$L_{A,N} = \{u \mid A \cap u \neq \emptyset \text{ and } N \cap u = \emptyset\}, \quad L_N = \{u \mid N \cap u = \emptyset\} .$$

Let $\mathcal{L}_1$ be the class $\mathcal{L}_1 = \{L_{A,N} \mid A, N \subseteq \Sigma$ for some set of objects $\Sigma\} \cup \{L_N \mid N \subseteq \Sigma$ for some set of objects $\Sigma\}$.

Theorem 5. *$PsP_a(ncoo, ndet, npro) = PsP_a(ncoo, det, npro) = \mathcal{L}_1$.*

Proof. First of all we prove that $\mathcal{L}_1 \subseteq PsP_a(ncoo, det, npro)$. Given a set of objects Σ and sets $A, N \subseteq \Sigma$, an acceptor for $L_{A,N}$ has no control object and rules $\{a \to T \mid a \in A\}$ and $\{b \to b \mid b \in N\}$. An acceptor for L_N contains initially an occurrence of T and has rules $\{b \to b \mid b \in N\}$.
It remains to prove that $PsP_a(ncoo, ndet, npro) \subseteq \mathcal{L}_1$. Assume any acceptor $\Pi \in P(ncoo, ndet, npro)$. If it contains a rule of the form $T \to u$, for any $u \in \Sigma^*$, then $Ps(\Pi) = \emptyset$, and $\emptyset \in \mathcal{L}_1$ ($\emptyset = L_\Sigma$). Otherwise, let G be the graph having a node for each object in $\Sigma \cup C$ and an arch from a to b if there is a rule $a \to u$ with $b \in u$. Let N be the set of the objects $a \in \Sigma$ such that all paths from a are infinite, meaning that there exists an object a' such that $a \to \cdots \to a'$ and $a' \to \cdots \to a'$, and let A be the set of the objects $a \in \Sigma$ such that at least one path from a is finite and leads to T, namely has the form $a \to \cdots \to T$. If T is an initial object in Π then a multiset is accepted iff it gives rise to a finite computation, because no rule can remove T and the final configuration, if reached, contains T for sure. Therefore, $Ps(\Pi) = L_N$. If T is not initially in Π, then a multiset is accepted iff it gives rise to a finite computation that introduces T in one of its steps. Therefore, $Ps(\Pi) = L_{A,N}$. □

Corollary 2. *$PsP_g(ncoo, ndet, npro) \supset PsP_a(ncoo, ndet, npro)$.*

Proof. Follows from results in [11], where P systems without cooperative rules and with the output interpreted as a natural number are proven to be able to generate semilinear set of numbers (Theorem 3.3.2). In the proof of the theorem a translation of context free grammars into P systems without cooperation is given which implies that $PsREG \subseteq PsP_g(ncoo, ndet, npro)$. It is obvious that $\mathcal{L}_1 \subset PsREG$. (Recall that $PsREG = PsCF$.) □

Let us switch to deterministic generators and acceptors. In this case the expressive power of generators is quite poor, and equivalent to the class of multiset languages consisting of at most one multiset. Let $\mathcal{L}_2 = \{\{w\} \mid w$ is a multiset$\} \cup \{\varnothing\}$.

Proposition 1. *$PsP_g(ncoo, det, pro) = PsP_g(coo, det, pro) = PsP_g(ncoo, det, npro) = PsP_g(coo, det, npro) = \mathcal{L}_2$.*

Proof. The initial configuration of a generator P system is fixed. Determinism implies that there is one only possible execution. If such an execution terminates, then it gives the only multiset of the language as output, otherwise the generated language is empty. Any language consisting of one multiset u can be generated by means of a non-cooperative rule without promoters $a \to v$ at the first step, where a is a control object initially in the P system. Cooperation and promoters do not increase expressiveness. $\square$

We already know that $PsP_a(ncoo, det, npro) = \mathcal{L}_1$. Let us characterize the class $PsP_a(ncoo, det, pro)$, which turns out to be more expressive.

Let $\mathcal{L}_3$ denote the least class of multiset languages including all sets $\{a^n | n \geq k\}$ for every object a and every $k \in \mathbb{N}$, closed by complementation, finite union and finite intersection.

Let us write $u \xrightarrow{R} v$ to denote that by performing a multiset of rules R we rewrite a multiset of objects u into a multiset of objects v. Let us write $u \models r$ if a multiset of objects u triggers a rule r.

Theorem 6. *$PsP_a(ncoo, det, pro) = \mathcal{L}_3$.*

Proof. Let us prove first that $PsP_a(ncoo, det, pro) \subseteq \mathcal{L}_3$. Let Π be an acceptor in $P(ncoo, det, pro)$. By Thm. 1 we can assume that Π is flat. Let $\Pi = (\Sigma \cup C \cup \{T\}, \emptyset, w, R)$. Assume that $R = \{r_i \mid i \in I\}$, with each r_i of the form $a_i \to u_i|_{p_i}$. Let $m = \max\{h \mid \exists i \in I, b \in \Sigma.\, b^h \in p_i\}$.
To prove that $Ps(\Pi) \in \mathcal{L}_3$ it is enough to prove that if $a^h u \in Ps(\Pi)$ for some $a \in \Sigma$, $u \in \Sigma^*$ and $h \geq m$, then also $a^{h+1}u \in Ps(\Pi)$.
Assume that $a^h u$ is accepted by Π by performing n computation steps, for some $n \in \mathbb{N}$. More precisely, there exist n multisets $T_1, \ldots, T_n$, with $T_j = \{r_i^{n_{i,j}} \mid i \in I\}$, and $n+1$ multisets of objects $u_0, u_1, \ldots, u_n$ such that $u_0 \xrightarrow{T_1} u_1 \xrightarrow{T_2} \cdots \xrightarrow{T_n} u_n$, $u_0 = a^h u$, $T \in u_n$ and $u_n \not\models r_i$ for any $i \in I$.
Let $v_0 = a^{h+1}u$. We can prove that there exist n multisets $T'_1, \ldots, T'_n$, with $T'_j = \{r_i^{n'_{i,j}} \mid i \in I\}$, and n multisets of objects $v_1, \ldots, v_n$ such that $v_0 \xrightarrow{T'_1} v_1 \xrightarrow{T'_2} \cdots \xrightarrow{T'_n} v_n$, $v_n \not\models r_i$ for any $i \in I$, $n_{i,j} \leq n'_{i,j}$ and $n_{i,j} = 0 \Rightarrow n'_{i,j} = 0$ for each $i \in I$ and $1 \leq j \leq n$, and, finally, $v_j \supseteq u_j$ and $b^x \in u_j$ and $b^{x+y} \in v_j$ with $y > 0$ and $b^{x+y} \not\subseteq u_j$ imply $x \geq m$ for each $0 \leq j \leq n$.
In fact, we know that $v_0 \supseteq u_0$ and that $b^x \in u_0$ and $b^{x+y} \in v_0$ with $y > 0$ and $b^{x+y} \not\subseteq u_0$ imply that b is a, and, therefore, $x = h \geq m$. Now, given any $0 \leq j \leq n$, assume that $v_j \supseteq u_j$ and that $b^x \in u_j$ and $b^{x+y} \in v_j$ with $y > 0$ and $b^{x+y} \not\subseteq u_j$ imply $x \geq m$. This implies that v_j and u_j promote the same rules. Therefore, if $j = n$, since $u_j \not\models r_i$ for any $i \in I$, we infer that also $v_j \not\models r_i$ for any $i \in I$. If $j < n$ then $v_j \supseteq u_j$ implies that a computation step $v_j \xrightarrow{T'_j} v_{j+1}$ with $n'_{i,j} \geq n_{i,j}$ actually exists. Since u_j and v_j promote the same rules and all rules are non-cooperative, we also infer that $n_{i,j} = 0$ implies $n'_{i,j} = 0$. Moreover, $u_{j+1} = (u_j \cap \{b \mid b \neq a_i$ for any i with $n_{i,j} > 0\}) \cup \{u_i^{n_{i,j}} \mid i \in I\}$ and $v_{j+1} = (v_j \cap \{b \mid b \neq a_i$ for any i with $n_{i,j} > 0\}) \cup \{u_i^{n'_{i,j}} \mid i \in I\}$. The relation

$u_{j+1} \subseteq v_{j+1}$ follows immediately. It remains to prove that it cannot happen that $b^x \in u_{j+1}$, $b^{x+y} \in v_{j+1}$, $y > 0$, $b^{x+y} \not\subseteq u_{j+1}$ and $x < m$ for any $b \in \Sigma$. If, by contradiction, this happens for some b, then there is a rule $r_i = a_i \to u_i|_{p_i}$ with $b \in u_i$ such that $n'_{i,j} > n_{i,j}$ and $n_{i,j} < x + y$. We infer that $a_i^{n'_{i,j}} \in v_j$ and $a_i^{n_{i,j}} \in u_j$. Since we know that for all c it holds that $c^k \in u_j$ and $c^{k+h} \in v_j$ with $h > 0$ imply $k \geq m$, we infer that $n_{i,j} \geq m$. Having $n_{i,j} \geq m$ and $x < m$ is a contradiction, since $x \geq n_{i,j}$.

Let us prove now that $PsP_a(ncoo, det, pro) \supseteq \mathcal{L}_3$. Actually, we prove that each multiset in $\mathcal{L}_3$ is accepted by an acceptor P system in $P(ncoo, det, pro)$ that always terminates and that consumes neither objects in Σ nor the acceptance symbol T.
The multiset $\{a^n \mid n \geq k\}$ is accepted by a P system with a control object b, initial multiset b and a rule $b \to T|_{a^k}$.
The multiset $\{a^n \mid n < k\}$ is accepted by a P system with control objects $b, 1, 2$, initial multiset $b1$ and rules $b \to \lambda|_{a^k}$, $1 \to 2$ and $2 \to T|_b$.
Finally, assume two multiset languages L_1 and L_2 in $\mathcal{L}_3$ and let Π_1 and Π_2 be the P systems accepting them. Assume that the set of control objects in Π_1 and Π_2 are disjoint. Let Π'_1 and Π'_2 be the P systems obtained from Π_1 and Π_2 by replacing T with T_1 and T with T_2, respectively. The language $L_1 \cap L_2$ is built by joining all control objects and rules in Π'_1 with all control objects and rules in Π'_2 and by adding the rule $T_1 \to T|_{T_2}$. The language $L_1 \cup L_2$ is built by joining all objects and rules in Π'_1 with all objects and rules in Π'_2 and by adding the rules $T_1 \to T$ and $T_2 \to T$. □

Let us prove now the universality of the class $PsP_a(coo, det, npro)$. In this case the corresponding class $PsP_g(coo, det, npro)$ is less expressive, therefore looking for an encoding of $PsP_g(coo, det, npro)$ into $PsP_a(coo, det, npro)$ as done in Thms. 2, 3, 4 is useless. We directly simulate 3-register machines, for which universality has been proven without the need of any complicated representation of data and instructions (as it happens with 2-register machines).

Theorem 7. *$PsP_a(coo, det, npro) = PsRE$.*

Proof. We provide a map assigning to a 3-register machine M an equivalent acceptor P system Π_M in $P(coo, det, npro)$. Let R_1, R_2 and R_3 be the three registers of M, and $0 \leq i \leq m$ be the labels of its instructions. A state of M is a triple (i, A, B, C), with $0 \leq i \leq m$ and $A, B, C \in \mathbb{N}$, the initial state is $(1, A', B', C')$ for some $A', B', C' \in \mathbb{N}$, and the pair (A', B', C') is accepted if M starting from $(1, A', B', C')$ reaches $(0, 0, 0, 0)$. The idea is that Π_M uses objects i with $0 \leq i \leq n$, a, b and c, and represents a configuration (i, A, B, C) with multiset $(ia^A b^B c^C)$.

Instruction $i : R_1+, j$ is simulated by rule $i \to aj$.

Instruction $i : R_1-, j, k$ is simulated by rules

$$i \to x_i y_i \qquad ax_i \to x'_i \qquad y_i \to y'_i \qquad y'_i x'_i \to j \qquad y'_i x_i \to k\,.$$

Instructions over R_2 and R_3 are analogous, we simply replace any occurrence of a with b or c, respectively.

Finally, we need these rules:

$$0 \to T \qquad Ta \to \lambda \qquad Tb \to \lambda \qquad Tc \to \lambda\,.$$

□

All results over deterministic acceptors proved so far can be summarized in the following corollary.

Corollary 3. $PsP_a(ncoo, det, npro) \subset PsP_a(ncoo, det, pro) \subset PsP_a(coo, det, npro) = PsP_a(coo, det, pro)$.

Proof. Directly from Thm. 5, Thm. 6 and Thm. 7 □

Moreover, we have that each class of deterministic generators is strictly included in the corresponding class of acceptors.

Corollary 4. *It holds that:*

– $PsP_g(ncoo, det, npro) \subset PsP_a(ncoo, det, npro)$;
– $PsP_g(ncoo, det, pro) \subset PsP_a(ncoo, det, pro)$;
– $PsP_g(coo, det, npro) \subset PsP_a(coo, det, npro)$;
– $PsP_g(coo, det, pro) \subset PsP_a(coo, det, pro)$.

Proof. Directly from Thm. 5, Prop. 1, Cor. 3 and $\mathcal{L}_2 \subset \mathcal{L}_1$.

Finally, we provide an encoding of deterministic acceptors with cooperative rules and promoters into the subclass without promoters.

Theorem 8. $PsP_a(coo, det, pro) \Leftrightarrow PsP_a(coo, det, npro)$.

Proof. The encoding $PsP_a(coo, det, npro) \Rightarrow PsP_a(coo, det, pro)$ is obvious. As regards the other direction, given any acceptor $\Pi \in P(coo, det, pro)$, we derive an equivalent acceptor $\hat{\Pi} \in P(coo, det, npro)$. By Thm. 1 we can assume that Π is flat. Let $\Pi = (\Sigma \cup C, \emptyset, w, R)$. Assume that $R = \{r_1, \ldots, r_k\}$ and $\Sigma \cup C = \{a_1, \ldots, a_n\}$. Also $\hat{\Pi}$ will be flat, of the form $\hat{\Pi} = (\hat{\Sigma} \cup \hat{C}, \emptyset, \hat{w}, \hat{R})$, with $\hat{\Sigma} = \Sigma$, $\hat{C} \supset C$ and $\hat{w} \supseteq w$.

The starting idea is that for any rule $r_i \equiv u_i \to v_i|_{p_i}$ in R we have a rule r'_i without promoters in $\hat{R}$ of the form $r'_i \equiv u_i p_i \to v_i p_i$, so that performing a step S by Π with n_i occurrences of r_i in parallel for each $1 \le i \le k$ is simulated by performing n_i occurrences of r'_i in sequence, for each $1 \le i \le k$.

Rules r_i as above do not work, for three reasons. The first point is that if $u_i \cap p_i \neq \emptyset$ then r_i is triggered by $(u_i \cup p_i) \setminus (u_i \cap p_i)$, whereas r'_i requires the whole multiset $u_i \cup p_i$. This can be repaired by rewriting all r_i as $u_i \to v_i|_{p_i \setminus u_i}$, without modifying the behavior of Π, before deriving $\hat{\Pi}$ from Π. The second point is that by moving promoters to left hand sides of rules we may introduce nondeterminism. (For example, by trasforming rules $a \to d|_c$ and $b \to e|_c$ into $ac \to dc$ and $bc \to ec$.) This can be repaired by rewriting r'_i as $iu_ip_i \to v_ip_i$, where $1 \le i \le k$ are new control objects in $\hat{C}$ that must be introduced in sequence. The third point is that if $v_i \cap u_i \neq \emptyset$ then performing r'_i may trigger r'_i itself, which should be prevented when we are simulating a single evolution step S by Π. This can be repaired by rewriting r'_i as $iu'_ip_i \rightarrow v''_ip_i$, provided that new control objects a', a'' are introduced in $\hat{C}$ for each $a \in \Sigma \cup C$, objects a are rewritten into a' before the sequence of steps by $\hat{\Pi}$ simulating S and objects a'' are rewritten into a after the same sequence.

The initial multiset $\hat{w}$ contains two fresh control objects $s, s' \in \hat{C}$. Acceptor $\hat{\Pi}$ may start by performing the following rules:

$$ss' \rightarrow 1s'\hat{1}\overline{1} \qquad R_a = \{a \rightarrow a' \mid a \in \Sigma \cup C\}$$

so that the object 1 triggering r'_1 is introduced and all objects in $\Sigma \cup C$ are primed. Also s', $\hat{1}$ and $\overline{1}$ are new control objects in $\hat{C}$, whose role will be clarified later.

Then, for each $1 \leq i \leq k$, $\hat{C}$ contains also objects i', i'', i''', i'''', and r'_i is adjusted once more as

$$r'_i \equiv i'u'_ip'_i \rightarrow v''_ip'_ii'''$$

For each $1 \leq i < k$ the following set of rules are added to $\hat{R}$

$$R_i = \{i \rightarrow i'i'', i'' \rightarrow i'''', i'''i'''' \rightarrow i, i'i'''' \rightarrow i+1\}$$

so that i is rewritten into $i+1$ as soon as r'_i remains without $u'_ip'_i$. Moreover, the following rules are added to $\hat{R}$

$$R_k = \{k \rightarrow k'k'', k'' \rightarrow k'''', k'''k'''' \rightarrow k, k'k'''' \rightarrow v_1\}$$

where the role of the control object $v_1 \in \hat{C}$ will be explained later.

Notice that a step S by Π consisting of n_i occurrences of r_i for each $1 \leq i \leq n$ is simulated by performing n_1 occurrences of r'_1 in sequence, then n_2 occurences of r'_2 in sequence and so on. This is correct only if Π is deterministic, as in our case. In fact, when Π is nondeterministic, our strategy solves the nondeterminism and, therefore, does not simulate some valid behaviors by Π.

It remains to map any a' that has not been consumed by $r'_1, \ldots, r'_k$ and any a'' that has been introduced by $r'_1, \ldots, r'_k$ to a. To this purpose, first of all we add to $\hat{C}$ new control objects $v_1, \ldots, v_n$ and $t_1, \ldots, t_n$, respectively.

Then, for each $1 \leq j \leq n$, $\hat{R}$ contains the following set of evolution rules

$$R^j = \{v_j \rightarrow v'_jv''_j, v'_ja'_j \rightarrow v'''_ja_j, v''_j \rightarrow v''''_j, v'''_jv''''_j \rightarrow v_j, v'_jv''''_j \rightarrow t_j\}$$

mapping all a'_js to a_j, where also $v'_j, v''_j, v'''_j, v''''_j$ are new objects in $\hat{C}$. When this task has been completed, t_j is introduced. Moreover, for all $1 \leq j < n$, $\hat{R}$ contains the following set of evolution rules

$$R^j_* = \{t_j \rightarrow t'_jt''_j, t'_ja''_j \rightarrow t'''_ja_j, t''_j \rightarrow t''''_j, t'''_jt''''_j \rightarrow t_j, t'_jt''''_j \rightarrow v_{j+1}\}$$

mapping all a''_js to a_j, where also $t'_j, t''_j, t'''_j, t''''_j$ are new objects in $\hat{C}$. Finally, $\hat{R}$ contains also the following set of evolution rules

$$R^n_* = \{t_n \rightarrow t'_nt''_n, t'_na''_n \rightarrow t'''_na_n, t''_n \rightarrow t''''_n, t'''_nt''''_n \rightarrow t_n, t'_nt''''_n \rightarrow t\}$$

mapping all a_n''s to a_n. When this task has been completed, object $t \in \hat{C}$ is introduced, which is exploited to trigger the rule

$$t \to s$$

so that s and s' can reintroduce 1 once more and the subsequent step by Π can be simulated. It remains to solve a point: When no rule $r_1', \dots, r_k'$ is triggered, $\hat{\Pi}$ does not terminate but enters an infinite loop. Notice that this happens when object v_1 is introduced in $3k$ steps after object 1. So, let us rewrite the rule $k'k'''' \to v_1$ introducing v_1 as $k'k'''' \to v_1 z z'$, where z, z' are new control objects in $\hat{C}$, and let us add to $\hat{R}$ the following set of rules R':

$$\{\widehat{i} \to \widehat{i+1} \mid 1 \le i < 3k\} \quad \{\overline{i} \to \overline{i+1} \mid 1 \le i \le 3k\}$$

$$z\widehat{3k}s' \to \lambda \quad \overline{3k+1}\widehat{3k} \to \lambda \quad z' \to z'' \quad zz'' \to \lambda$$

so that when $\widehat{3k}$ and z appear at the same step then s' is removed and the infinite loop is prevented. Objects $\overline{i}$ are needed since $\overline{3k+1}$ removes $\widehat{3k}$ when $\widehat{3k}$ appears before z. □

4 Conclusions

The paper has studied relationships among classes of multiset languages accepted and generated by P systems. In particular, the role of determinism, presence of promoters and cooperative rules have been considered.

In the nondeterministic case, when either promoters or cooperative rules are allowed, acceptor P systems have shown to be universal. The same is known to hold for the corresponding classes of nondeterministic generator P systems.

In the deterministic case, acceptor P systems have been shown to be universal only if cooperative rules are allowed. Universality has been shown not to hold for the corresponding classes of generator P systems.

All the considered classes of languages have been characterized, and results of strict inclusion among some of them have been proved. Moreover, in some cases, we have been able to construct mappings taking P systems working in generating mode into the corresponding P systems working in accepting mode.

References

1. Barbuti, R., Maggiolo-Schettini, A., Milazzo, P., Tini, S.: A P Systems Flat Form Preserving Step-by-step Behaviour. Fundamenta Informaticae 87, 1-34 (2008)
2. Bianco, L., Manca, V.: Encoding–Decoding Transitional Systems for Classes of P Systems. In: Workshop on Membrane Computing (WMC 2005). LNCS, vol. 3850, pp. 134-143, Springer (2006)
3. Bottoni, P., Martin-Víde, C., Păun, G., Rozenberg, G.: Membrane Systems with Promoters/Inhibitors. Acta Informatica 38, 695-720 (2002)
4. Csuhaj-Varjú, E.: P Automata. In: Workshop on Membrane Computing 2004. LNCS, vol. 3365, pp. 19-35, Springer (2005)

5. Fernau, H.: Graph-controlled Grammars as Language Acceptors. Journal of Automata, Languages and Combinatorics 2, 79-91 (1997)
6. Fernau, H., Holzer, M.: Accepting Multi-Agent Systems II. Acta Cybernetica 12, 361-380 (1996)
7. Fernau, H., Holzer, M., Bordihn, H.: Accepting Multi-Agent Systems. Computers and Artificial Intelligence 15, 123-139 (1996)
8. Freund, R and Oswald, M: A short note on analysing P systems. Bulletin of the EATCS, 79, 2002, 231–236.
9. Ibarra, O.H.: On the Computational Complexity of Membrane Systems. Theoretical Computer Science 320, 89-109 (2004)
10. Păun, G.: Computing with Membranes. Journal of Computer and System Sciences 61, 108-143 (2000)
11. Păun, G.: Membrane computing. An introduction. Springer-Verlag, Berlin, (2002)
12. Porreca, A.E., Mauri, G., Zandron, C.: Complexity Classes for Membrane Systems. Theoretical Informatics and Applications 40, 141-162 (2006)
13. Qi, Z., You,J., Mao, H.: P systems and Petri nets. In: Workshop on Membrane Computing 2003. LNCS, vol. 2933, pp. 286–303, Springer (2004)
14. Shapiro, E.Y.: The Family of Concurrent Logic Programming Languages. ACM Comput. Surv. 21, 1989, 412–450