
On a Class of P Automata as a Machine Model for Languages over Infinite Alphabets ^{*}

György Vaszil

Computer and Automation Research Institute
Hungarian Academy of Sciences
Kende utca 13-17, 1111 Budapest, Hungary
E-mail: vaszil@sztaki.hu

Summary. We show how P automata having a finite description and working with a finite object-alphabet can be used to describe languages over countably infinite alphabets. We propose to relate the language classes characterized by different types of P automata to some of the existing characterizations of language classes over infinite alphabets, and give an upper bound for the class of languages accepted by the class of one of the most straightforward and least complicated variants of these types of P automata.

1 Introduction

Membrane systems, or P systems are distributed computing models inspired by the functioning of the living cell. Their main components are membrane structures consisting of membranes hierarchically embedded in the outermost skin membrane. Each membrane encloses a region containing a multiset of objects and possibly other membranes. Each region has an associated set of operators working on the objects contained by the region. These operators can be of different types, they can change the objects present in the regions or they can provide the possibility of transferring the objects from one region to another one. The system might also have additional capabilities such as, for example, a dynamically changing membrane structure, or special constraints (like context conditions or priority relations) added to the sets of operators. The evolution of the objects inside the membrane structure from an initial configuration to a somehow specified end configuration corresponds to a computation having a result which is derived from some properties of the specific end configuration. Several variants of the basic notion have been introduced and studied proving the power of the framework, see the monograph [11] for a summary of notions and results of the area. As the reader might notice, this machinery has relatively strong capabilities, so it might seem reasonable to

^{*} Research supported in part by the Hungarian Scientific Research Fund “OTKA” grants T042529 and F037567.

look for as simple P system variants as possible, as for example systems using communication rules only. For more details on these variants, see [8], [10].

Besides their simplicity, the introduction of P automata in [2] was motivated by the idea of using P systems as language acceptors. The objects in a P automaton may move through the membranes from region to region, but they may not be modified during the functioning of the systems, and furthermore, the “words” accepted by a P automaton correspond to the sequences of multisets containing the objects entering from the environment in each step of the evolution of the system.

To characterize formal languages, the strings of multisets read by the P automaton have to be mapped to strings of symbols of an alphabet. If this mapping uses erasing, that is, some possible input multisets are mapped to the empty symbol, ε , then the characterization of recursively enumerable languages can be obtained. This was already established in [2] stating that for any recursively enumerable language, there is a P automaton accepting the image of the language under a certain mapping. Similar results were also obtained in [4], [5], for P automata with different features and different mappings to obtain the recursively enumerable language.

If the correspondence between the input multisets and the alphabet symbols is established by a mapping which is non-erasing and “simple”, that is, each possible input multiset is mapped to a symbol of a finite alphabet in some computationally “easy” way (except the empty multiset which is still mapped to ε), then the situation is different as shown in [3]. In the case of sequential rule application, the number of possible input multisets is finite, so the correspondence between the input multisets and the alphabet symbols is natural. P automata with these types of mappings and sequential application of symport/antiport rules characterize a subclass of the languages accepted by Turing machines reading the input tape one-way and using logarithmic space on the work-tape. (The interested reader is referred to [3] for the exact definitions and challenging open problems concerning this computational complexity class.) If the symport/antiport rules are applied in the maximal parallel manner, then the number of potential inputs is infinite, thus, the infinite set of possible inputs has to be mapped to a finite alphabet. In this case (if the mapping is non-erasing and “simple”, as above) the characterization of the class of context-sensitive languages is obtained.

Because of the infinite number of potential input multisets, it is rather natural to consider a P automaton with parallel rule application as a machine working with strings of symbols over infinite alphabets. The only thing necessary to obtain such a model is to define a mapping establishing the correspondence between the input multisets and the symbols of the infinite alphabet. In the following we begin the exploration of this idea. First we introduce the model, then present two examples which also give a hint towards the relationship of the language classes described by P automata and those proposed in [9] and [6]. Finally we present a theorem on the limits of one of the “easiest” infinite alphabet language class described by P automata.

2 Preliminaries and Definitions

We first recall the notions and the notations we use. The reader is assumed to be familiar with the basics of formal language theory, for details see [12]. Let Σ be a not necessarily finite, but countable set of symbols called alphabet. Let Σ^* be the set of all words over Σ , that is, the set of finite strings of symbols from Σ , and let $\Sigma^+ = \Sigma^* - \{\varepsilon\}$ where ε denotes the empty word.

Let U be a set - the universe - of objects, and let \mathbb{N} denote the set of natural numbers. A multiset is a pair $M = (V, f)$, where $V \subseteq U$ and $f : U \rightarrow \mathbb{N}$ is a mapping which assigns to each object $a \in V$ its multiplicity, if $a \notin V$ then $f(a) = 0$. The support of $M = (V, f)$ is the set $supp(M) = \{a \in V \mid f(a) \geq 1\}$. If $supp(M)$ is a finite set, then M is called a finite multiset. The set of all finite multisets over the set V is denoted by V° .

The number of objects in a finite multiset $M = (V, f)$, the cardinality of M , is defined by $card(M) = \sum_{a \in V} f(a)$. We say that $a \in M = (V, f)$ if $a \in supp(M)$. $M_1 = (V_1, f_1) \subseteq M_2 = (V_2, f_2)$ if $supp(M_1) \subseteq supp(M_2)$ and for all $a \in V_1$, $f_1(a) \leq f_2(a)$. The union of two multisets is defined as $(M_1 \cup M_2) = (V_1 \cup V_2, f')$ where for all $a \in V_1 \cup V_2$, $f'(a) = f_1(a) + f_2(a)$, the difference is defined for $M_2 \subseteq M_1$ as $(M_1 - M_2) = (V_1 - V_2, f'')$ where $f''(a) = f_1(a) - f_2(a)$ for all $a \in V_1 - V_2$, and the intersection of two multisets is $(M_1 \cap M_2) = (V_1 \cap V_2, f''')$ where for $a \in V_1 \cap V_2$, $f'''(a) = \min(f_1(a), f_2(a))$, $\min(x, y)$ denoting the minimum of $x, y \in \mathbb{N}$. We say that M is empty, denoted by ϵ , if its support is empty, $supp(M) = \emptyset$.

A multiset M over the finite set of objects V can be represented as a string w over the alphabet V with $|w|_a = f(a)$ where $a \in V$ and where $|w|_a$ denotes the number of occurrences of the symbol a in the string w , and with ϵ representing the empty multiset. In the following we sometimes identify the finite multiset of objects $M = (V, f)$ with the word w over V representing M , thus we write $w \in V^\circ$, or sometimes we enumerate the not necessarily distinct elements a_1, \dots, a_n of a multiset as $M = \{\{a_1, \dots, a_n\}\}$, by using double brackets to distinguish from the usual set notation.

A P system is a structure of hierarchically embedded membranes, each having a label and enclosing a region containing a multiset of objects and possibly other membranes. The out-most membrane which is unique and usually labelled with 1, is called the skin membrane. The membrane structure is denoted by a sequence of matching parentheses where the matching pairs have the same label as the membranes they represent. If $x \in \{\{i, \}_i \mid 1 \leq i \leq n\}^*$ is such a string of matching parentheses of length $2n$, denoting a structure where membrane i contains membrane j , then $x = x_1 [i x_2 [j x_3]_j x_4]_i x_5$ for some $x_k \in \{\{l, \}_l \mid 1 \leq l \leq n, l \neq i, j\}^*$, $1 \leq k \leq 5$. If membrane i contains membrane j , and there is no other membrane, k , such that k contains j and i contains k (x_2 and x_4 above are strings of matching parentheses themselves), then we say that membrane i is the parent membrane of j .

The evolution of the contents of the regions of a P system is described by rules associated to the regions. Applying the rules synchronously in each region, the system performs a computation by passing from one configuration to another one.

In the following we concentrate on communication rules called symport or antiport rules.

A symport rule is of the form (x, in) or $(x, out), x \in V^\circ$. If such a rule is present in a region i , then the objects of the multiset x must enter from the parent region or must leave to the parent region, respectively. An antiport rule is of the form $(x, in; y, out), x, y \in V^\circ$, in this case, objects of x enter from the parent region and in the same step, objects of y leave to the parent region. All types of these rules might be equipped with a promoter or inhibitor multiset, denoted as $(x, in)|_Z, (x, out)|_Z$, or $(x, in; y, out)|_Z, x, y \in V^\circ, Z \in \{z, \neg z \mid z \in V^\circ\}$, in which case they can only be applied if region i contains the objects of multiset z , or if $Z = \neg z$, then region i must not contain the elements of z . (For more on symport/antiport see [10], for the use of promoters see [8].)

Now we recall the formal definition of a P automaton.

Definition 1 A *P automaton* is a construct $\Gamma = (V, \mu, (w_1, P_1, F_1), \dots, (w_n, P_n, F_n))$ where $n \geq 1$ is the number of membranes, V is a finite set of objects, μ is a membrane structure of n membranes with membrane 1 being the skin membrane, and for all $i, 1 \leq i \leq n$,

- $w_i \in V^\circ$ is the initial contents (state) of region i , that is, it is the finite multiset of all objects contained by region i ,
- P_i is a finite set of communication rules associated to membrane i , they can be symport rules or antiport rules, with or without promoters or inhibitors, as above, and
- $F_i \subseteq V^\circ$ is a finite set of finite multisets over V called the set of final states of region i . If $F_i = \emptyset$, then all the states of membrane i are considered to be final.

To simplify the notations we denote symport and antiport rules with or without promoters/inhibitors as $(x, in; y, out)|_Z, x, y \in V^\circ, Z \in \{z, \neg z \mid z \in V^\circ\}$ where we also allow x, y, z to be the empty string. If $y = \varepsilon$ or $x = \varepsilon$, then the notation above denotes the symport rule $(x, in)|_Z$ or $(y, out)|_Z$, respectively, if $Z = \varepsilon$, then the rules above are without promoters or inhibitors.

The n -tuple of finite multisets of objects present in the n regions of the P automaton Γ describes a *configuration* of Γ , the n -tuple $(w_1, \dots, w_n) \in (V^\circ)^n$ is the initial configuration.

The application of the rules can take place in a sequential, or in a maximally parallel manner. Here we only consider parallel rule application, but we keep the subscript “*par*” in order to emphasize that maximal parallel application is just one of the different possibilities.

Definition 2 The transition mapping of a P automaton is a partial mapping $\delta_{par} : V^\circ \times (V^\circ)^n \rightarrow 2^{(V^\circ)^n}$. These mappings are defined implicitly by the rules of the rule sets $P_i, 1 \leq i \leq n$. For a configuration (u_1, \dots, u_n) ,

$$(u'_1, \dots, u'_n) \in \delta_{par}(u, (u_1, \dots, u_n))$$

holds, that is, while reading the input $u \in V^\circ$ the automaton may enter the new configuration $(u'_1, \dots, u'_n) \in (V^\circ)^n$, if there exist rules as follows.

- For all $i, 1 \leq i \leq n$, there is a multiset of rules $R_i = \{\{r_{i,1}, \dots, r_{i,m_i}\}\}$, where $r_{i,j} = (x_{i,j}, in; y_{i,j}, out)|_{Z_{i,j}} \in P_i$ with $z \subseteq u_i$ for $Z_{i,j} = z \in V^\circ$, and $z \cap u_i = \epsilon$ for $Z_{i,j} = \neg z, z \in V^\circ, 1 \leq j \leq m_i$, satisfying the conditions below, where x_i, y_i denote the multisets $\bigcup_{1 \leq j \leq m_i} x_{i,j}$ and $\bigcup_{1 \leq j \leq m_i} y_{i,j}$, respectively. Furthermore, there is no rule occurrence $r \in P_j$, for any $j, 1 \leq j \leq n$, such that if $r \in P_1$ then $r \neq (x, in)|_Z$, but the rule multisets R'_i with $R'_i = R_i$ for $i \neq j$ and $R'_j = \{\{r\}\} \cup R_j$, also satisfy the conditions.

The conditions are given as

1. $x_1 = u$, and
2. $\bigcup_{parent(j)=i} x_j \cup y_i \subseteq u_i, 1 \leq i \leq n$,

and then the new configuration is obtained by

$$u'_i = u_i \cup x_i - y_i \cup \bigcup_{parent(j)=i} y_j - \bigcup_{parent(j)=i} x_j, 1 \leq i \leq n.$$

The definition of the transition given above differs from the one presented [3]. The difference comes from the use of symport rules of the form $(x, in)|_Z$ in the region of the skin membrane. In [3] (and other previous work on P automata), the environment is considered to have an infinite supply of objects in every step of the computation, so the maximal parallel use of a symport rule of the above form would result in the import of the objects of x in an infinite number of copies, thus, the use of these types of rules is simply not allowed. Here, for the sake of being as general as possible, we allow the use of rules of type $(x, in)|_Z$ in the skin membrane, and say that the use of any number of these rules is considered to be maximally parallel.

We define the sequence of multisets of objects accepted by the P automaton as the sequence of input multisets which appear in the environment and are consumed by the skin membrane in each computational step while the system reaches a final state, a configuration where for all j with $F_j \neq \emptyset$, the contents $u_j \in V^\circ$ of membrane j is “final”, i.e., $u_j \in F_j$.

Definition 3 Let us extend δ_{par} to $\bar{\delta}_{par}$, a function mapping $(V^\circ)^*$, the sequences of finite multisets over V , and $(V^\circ)^n$, the configurations of Γ , to new configurations. We define $\bar{\delta}_{par}$ as

1. $\bar{\delta}_{par}(v, (u_1, \dots, u_n)) = \delta_{par}(v, (u_1, \dots, u_n)), v, u_i \in V^\circ, 1 \leq i \leq n$, and
2. $\bar{\delta}_{par}((v_1) \dots (v_{s+1}), (u_1, \dots, u_n)) = \bigcup \delta_{par}(v_{s+1}, (u'_1, \dots, u'_n))$
for all $(u'_1, \dots, u'_n) \in \bar{\delta}_{par}((v_1) \dots (v_s), (u_1, \dots, u_n)), v_j, u_i, u'_i \in V^\circ,$
 $1 \leq i \leq n, 1 \leq j \leq s + 1$.

Note that we use brackets in the multiset sequence $(v_1) \dots (v_{s+1}) \in (V^\circ)^*$ in order to distinguish it from the multiset $v_1 \cup \dots \cup v_{s+1} \in V^\circ$.

Definition 4 Let Γ be a P automaton as above with initial configuration (w_1, \dots, w_n) , let Σ be an alphabet, and let $f : V^\circ \rightarrow \Sigma \cup \{\epsilon\}$ be a mapping with $f(x) = \epsilon$ if and only if $x = \epsilon$.

The language accepted by Γ is defined as

$$L(\Gamma, f) = \{f(v_1) \dots f(v_s) \in \Sigma^* \mid (u_1, \dots, u_n) \in \bar{\delta}_{par}((v_1) \dots (v_s), (w_1, \dots, w_n)) \\ \text{with } u_j \in F_j \text{ for all } j \text{ with } F_j \neq \emptyset, 1 \leq j \leq n, 1 \leq s\}.$$

Obviously, the choice of the mapping f is essential. It has to be “easily” computable because the power of the P automaton should be provided by the underlying membrane system and not by f itself. The notion of “easiness”, however, greatly depends on the context we are working in, so we do not give it a general specification here.

3 Examples

Before continuing the study of our model, we recall two previous attempts to define the generalization of the notion of “regular languages” to the case of infinite alphabets. The language class denoted here as $\mathcal{L}(REG_1^\infty)$ is characterized in [9] through an encoding of the words over the infinite alphabet $\Sigma = \{a_1, a_2, \dots\}$ to words over the binary alphabet $\{0, 1\}$ defined as $\gamma(a_i) = 0^{i-1}1$ for each $i \geq 1$. For a language $L \subseteq \Sigma^*$, the property of $L \in \mathcal{L}(REG_1^\infty)$ holds if $\gamma(L) = \{\gamma(w) \mid w \in L\} \subseteq \{0, 1\}^*$ is a regular language. The class of $\mathcal{L}(REG_1^\infty)$ has several nice properties, but also strong limitations, for example, the simple language $L_1 = \{a_i a_1 a_i \mid i \geq 1\} \notin \mathcal{L}(REG_1^\infty)$.

A different language class, denoted here as $\mathcal{L}(REG_2^\infty)$, is proposed in [6] by generalizing the notion of finite automata to so called finite memory automata which process inputs over infinite alphabets. Without going into the details, a finite memory automaton is a finite automaton equipped with a finite number of registers which are used for storing certain symbols of the input word in some well defined way during the computation. To be able to give a finite description of the machine, the transitions are based on the finitely many possible states of the internal control and the equality or non-equality of the input with the contents of one or more of the finitely many registers. The class of $\mathcal{L}(REG_2^\infty)$ also has its limitations, for example, $L_2 = \{a_{2^i} \mid i \geq 1\} \notin \mathcal{L}(REG_2^\infty)$.

Note also that $L_2 \in \mathcal{L}(REG_1^\infty)$ and $L_1 \in \mathcal{L}(REG_2^\infty)$, thus, $\mathcal{L}(REG_1^\infty)$ and $\mathcal{L}(REG_2^\infty)$ are incomparable.

Now we give two examples of P automata by showing how L_1 and L_2 can be accepted.

Example 1 Let $\Gamma_1 = (\{a, \#, \$, @\}, [1 \ 2 \ 2]_1, (w_1, P_1, F_1), (w_2, P_2, F_2))$ with

$$\begin{aligned} w_1 &= \{\{\#\}\}, \\ P_1 &= \{(a, in)|_\#, (a, in; \$, out), (a, in; a, out)|_\@\}, \\ F_1 &= \emptyset, \text{ and} \\ w_2 &= \{\{\$, @\}\}, \end{aligned}$$

$$P_2 = \{(\#, in; \$, out), (a, in; @, out)|_\#, (a, out)\},$$

$$F_2 = \{ \{ \{ \# \} \} \}.$$

In the first step, an arbitrary number of a symbols, say k , is imported into the system and the symbol $\#$ in region 1 is exchanged to the symbol $\$$ from region 2. In the second step, one a symbol from the environment is exchanged to the symbol $\$$, and one a symbol and the $@$ symbol is exchanged between region 1 and 2. Now the number of a symbols in region 1 is k , so using the rule $(a, in; a, out)|_@$, the same number, k , of a symbols are imported to the system from the environment while one a symbol is sent out of the second region, thus, the system reaches a final state.

If for $x \in V^\circ$ and $\Sigma = \{a_1, a_2, \dots\}$ we have $f(x) = a_{card(x)}$, then $L(\Gamma_1, f) = L_1 = \{a_i a_1 a_i \mid i \geq 1\}$.

Example 2 Let $\Gamma_2 = (\{a, \#\}, [1 \ 2]_2]_1, (w_1, P_1, F_1), (w_2, P_2, F_2))$ with

$$w_1 = \{ \{ \# \} \},$$

$$P_1 = \{(a, in)|_\#\},$$

$$F_1 = \{ \{ \epsilon \} \}, \text{ and}$$

$$w_2 = \epsilon,$$

$$P_2 = \{(\#, in), (aa, in)|_\#\},$$

$$F_2 = \emptyset.$$

This system inputs an arbitrary number of a symbols from the environment in the first computational step while also moving the symbol $\#$ to region 2. After the first step, no more input from the environment is possible, but using the rules $(aa, in)|_\#$ the system moves an even number of a symbols from the first region to the second. If there is nothing left in the first region, thus, if the number of a symbols input in the first step was even, the system enters the final state.

If for $x \in V^\circ$ and $\Sigma = \{a_1, a_2, \dots\}$ we again have $f(x) = a_{card(x)}$, then $L(\Gamma_2, f) = L_2 = \{a_{2i} \mid i \geq 1\}$.

Before we abandon this line of investigation, at least for the present paper, we mention that similar approaches to the ones described above also exist for the definition of the class of context-free languages over infinite alphabets, see [9] and [1]. To establish the precise relationship of the language classes described by P automata and these infinite alphabet analogues of regular and context-free languages would certainly be an interesting and fruitful research topic (not to mention the comparison with the other – not too many – classes of infinite alphabet languages which can be found in the literature).

4 A Preliminary Result

For now, we would like to present a theorem providing an upper bound to the power of those P automata which use mappings between the input multisets and

the infinite alphabet that are similar to those of the previous examples, namely, if Σ is an infinite alphabet with $\Sigma = \{a_1, a_2, \dots\}$, then let F be the class of mappings $f : V^\circ \longrightarrow \Sigma \cup \{\varepsilon\}$ defined as $f(M) = a_j$ where $j = \text{card}(M)$, and furthermore, if $f^{-1}(a_i) = M$, then $M = (\{a\}, g)$ for a fixed element $a \in V$ and $g(a) = j$, thus, the possible input multisets consist of different number of occurrences of a single object.

Now let us denote by $\mathcal{L}(PA, \text{card})$ the class of languages $L(\Gamma, f)$ over Σ for some P automata Γ and mapping $f \in F$, and let also the value of the integer expressed by the binary string $x \in \{1\}\{0, 1\}^*$ be denoted by $\text{val}(x)$.

We present the following theorem mainly for the purpose of demonstrating a type of the results which can be expected in this framework and to invite the reader to continue this chain of thoughts by trying to relax one or more of the very strict conditions of the statement.

Theorem 1. *If $L \in \mathcal{L}(PA, \text{card})$ then $\gamma(L) \in \mathcal{L}(CS_3)$ where $\gamma : \Sigma^* \rightarrow \{0, 1, \$\}^*$ defined by $\gamma(a_i) = x\$$ with $x \in \{1\}\{0, 1\}^*$, $\text{val}(x) = i$ for all $a_i \in \Sigma$ and $\mathcal{L}(CS_3)$ denotes the class of context-sensitive languages over the three letter alphabet $\{0, 1, \$\}$.*

Proof. Let Γ be a P automaton and let $L = L(\Gamma, f)$ for some $f \in F$ as above. Consider now a Turing machine having one read only input tape and n^2 work-tapes where n is the number of membranes of the P automaton. The input alphabet is $\{0, 1, \$\}$, the work-tape alphabet is $\{0, 1\}$. What this Turing machine does, is keeping track of the configurations of Γ by writing (in binary notation) the number of objects in each membrane on the first n work-tapes and simulating the configuration changes using the rest of the tapes for keeping track of intermediate results.

Without entering any details of the construction (which is very similar to the proof of Theorem 1 in [3]) we intend to convince the reader that this is not only possible, but it is possible using a workspace which is a linear function of the length of the input. The rules and the position of the finite number of input symbols can be encoded in the finite control and all the additional information necessary to record the configurations of Γ is stored on the first n work-tapes. The Turing machine first writes the number of symbols leaving from each membrane to all of the others using the rest of the work-tapes making sure that the simulated rule application is maximally parallel, and also checking whether the number of objects given by the next binary sequence of the input word can be read from the environment. Then it updates the values corresponding to the configuration of Γ stored on the first n work-tapes.

Since the number of work-tapes is fixed, and the numbers which need to be written on the individual tapes are bounded by the sum of the values of binary sequences of the input, not to mention the simple computation of f and f^{-1} , this whole process can be realized in linear space. \square

5 Conclusion

We have proposed the idea of using P automata for the description and classification of languages over infinite alphabets which is a so far unexplored area with a whole lot of research topics yet to be studied.

For easier orientation we would like to stress two important problems which deserve to be the first among the questions that need to be investigated. First, it would be necessary to try to relate the model to existing characterizations of languages over infinite alphabets, some of which are mentioned in the earlier sections. Secondly, to investigate the role of the choice of f , the mapping between the input multisets and the infinite alphabet would be of great importance. It would be necessary to find mappings which are easily computed but complicated enough in order to not suppress the power of the underlying membrane framework (as it is probably suppressed by members of the class F of Theorem 1).

References

1. E.H.Y. Cheng, M. Kaminski: Context-free languages over infinite alphabets. *Acta Informatica*, 35 (1998), 245–267.
2. E. Csuhaj-Varjú, Gy. Vaszil: P Automata. In *Pre-Proceedings of the Workshop on Membrane Computing WMC-CdeA 2002* (Gh. Păun, C. Zandron, eds.), Curtea de Argeş, Romania, August 19-23, 2002. Pub. No. 1 of MolCoNet-IST-2001-32008 (2002) 177–192, and also in *Membrane Computing*, (Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, eds.), LNCS 2597, Springer-Verlag, Berlin, 2003, 219–233.
3. E. Csuhaj-Varjú, O.H. Ibarra, Gy. Vaszil: On the computational complexity of P automata. In *Preliminary Proceedings of DNA 10, Tenth International Meeting on DNA Computing* (C. Ferretti, G. Mauri, C. Zandron, eds.), June 7-10, 2004, University of Milano-Bicocca, 2004, 97–106.
4. R. Freund, C. Martín-Vide, A. Obtułowicz, Gh. Păun: On three classes of automata-like P systems. In *Developments in Language Theory. 7th International Conference, DLT 2003, Szeged, Hungary, July 2003* (Z. Ésik, Z. Fülöp, eds.), LNCS 2710, Springer-Verlag, Berlin, 2003, 292–303.
5. R. Freund, M. Oswald: A short note on analysing P systems. *Bulletin of the EATCS*, 78 (October 2002), 231–236.
6. M. Kaminski, N. Francez: Finite-memory automata. *Theoretical Computer Science*, 134, (1994), 329–363.
7. M. Madhu, K. Krithivasan: On a class of P automata. *Intern. J. Computer Math.*, 80, 9 (2003), 1111–1120.
8. C. Martín-Vide, A. Păun, Gh. Păun: On the power of P systems with symport rules. *Journal of Universal Computer Science*, 8, 2, (2002), 317–331.
9. F. Otto: Classes of regular and context-free languages over countably infinite alphabets. *Discrete Applied Mathematics*, 12, (1985), 41–56.
10. A. Păun, Gh. Păun: The power of communication: P systems with symport/antiport. *New Generation Computing*, 20, 3, (2002), 295–306.
11. Gh. Păun: *Membrane Computing. An Introduction*. Springer-Verlag, Berlin, 2002.
12. G. Rozenberg, A. Salomaa, eds.: *Handbook of Formal Languages*. Springer-Verlag, Berlin, 1997.

