# On the Computational Complexity of *P* Automata[⋆]

Erzsébet Csuhaj-Varjú[1], Oscar H. Ibarra[2], and György Vaszil[1]

[1] Computer and Automation Research Institute, Hungarian Academy of Sciences, Kende utca 13-17, 1111 Budapest, Hungary
{csuhaj, vaszil}@sztaki.hu

[2] Department of Computer Science, University of California, Santa Barbara, CA 93106, USA
ibarra@cs.ucsb.edu

**Abstract.** We characterize the classes of languages described by $P$ automata, i.e., accepting $P$ systems with communication rules only. Motivated by properties of natural computing systems, we study computational complexity classes with a certain restriction on the use of the available workspace in the course of computations and relate these to the language classes described by $P$ automata. We prove that if the rules of the $P$ system are applied sequentially, then the accepted language class is strictly included in the class of languages accepted by one-way Turing machines with a logarithmically bounded workspace, and if the rules are applied in the maximal parallel manner, then the class of context-sensitive languages is obtained.

## 1 Introduction

Membrane systems, or $P$ systems, are biomolecular computing devices working in a distributed and parallel manner inspired by the functioning of the living cell. The main ingredient of a $P$ system is a hierarchically embedded structure of membranes with rules associated to the regions describing the evolution of the objects present in the membranes. The evolution of the system corresponds to a computation. $P$ systems have intensively been studied in the past few years, the interested reader might consult the monograph [10] for a systematic study of the area.

The introduction of $P$ automata in [1] was motivated by an idea recently attracting researchers, namely, to use $P$ 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 described languages are obtained as the set of accepted sequences of

⋆ The research of E. Csuhaj-Varjú and Gy. Vaszil was supported in part by the Hungarian Scientific Research Fund "OTKA" grant no. T 042529. The research of O. H. Ibarra was supported in part by NSF Grants IIS-0101134 and CCR02-08595.

multisets containing the objects entering from the environment during the evolution of the system. The environment is considered to have an infinite supply of objects, any number of symbols may be requested by the application of one or more rules associated to the skin membrane, and these symbols may traverse this membrane and enter when they are requested to do so.

A result on the accepting power of $P$ automata was already established in [1] 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 [3], [4], and [7], for $P$ automata with different features and different mappings to obtain the recursively enumerable language.

In the present paper we continue the study of the power of $P$ automata, but this time we are interested in the exact characterization of the languages of the multiset sequences entering the system through the skin membrane during a computation. We do this by establishing a correspondence between the symbols of an alphabet and the multisets that might ever enter the $P$ automaton, and characterize the set of words corresponding to the set of accepted multiset sequences. This approach differs from the ones mentioned above because we do not allow erasing, that is, each nonempty multiset corresponds to a symbol of the alphabet when defining the string represented by the multiset sequence. This means that the workspace of the $P$ automaton is provided only by the objects of the input obtained from the environment during a computation, which is a very natural way of restricting the use of the resources: As the processing of the input progresses, additional parts of the workspace become available with each step. Thus, the workspace which can be used in the course of the computation is provided in accordance with the number of symbols actually read from the input, in other words, the computation is made possible by manipulating what is already obtained from the input. This idea agrees very well with the behaviour of natural systems where the result of a computation is also obtained by using the resources provided by the input, the object of the computation itself.

We consider the sequential and the so-called maximal parallel way of rule application. In the sequential case, the number of different multisets that may ever enter the system is finite which means that there is a natural one-to-one correspondence between these multisets and the symbols of a finite alphabet. This is not necessarily so when the rules are applied in the maximal parallel way, in this case $P$ automata can be considered as devices accepting finite strings over an infinite alphabet. In this paper we do not study the case of infinite alphabets, we use instead a mapping that maps the infinite set of different multisets to a finite alphabet, thus we will be able to speak of languages accepted by $P$ automata using the rules in the sequential or in the maximal parallel manner, the languages being in both cases over a finite alphabet.

We show that the languages which can be characterized by $P$ automata in this sense using the rules in the sequential manner are strictly included in the class of languages accepted by one-way Turing machines using logarithmically bounded workspace, while if the rules are used in the maximal parallel way, the class of context-sensitive languages is obtained.

## 2 Definitions

We first recall the notions and the notations we use. Let $V$ be an alphabet, let $V^*$ be the set of all words over $V$, and let $V^+ = V^* - \{\varepsilon\}$ where $\varepsilon$ denotes the empty word. We denote the length of a word $w \in V^*$ by $|w|$, and the number of occurrences of a symbol $a \in V$ in $w$ by $|w|_a$. The set of natural numbers is denoted by $\mathbb{N}$.

A multiset is a pair $M = (V, f)$, where $V$ is an arbitrary (not necessarily finite) set of objects and $f : V \to \mathbb{N}$ is a mapping which assigns to each object its multiplicity. The support of $M = (V, f)$ is the set $supp(M) = \{a \in V \mid f(a) \geq 1\}$. If $V$ is a finite set, then $M$ is called a finite multiset. The set of all finite multisets over the set $V$ is denoted by $V^\circ$.

We say that $a \in M = (V, f)$ if $a \in supp(M)$, and $M_1 = (V_1, f_1) \subseteq M_2 = (V_2, f_2)$ 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 $\epsilon$, 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 $|w|_a = f(a)$, $a \in V$, and with $\varepsilon$ representing the empty multiset $\epsilon$. 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 elements of $w = a_1 \dots a_t \in V^\circ$ in double brackets (to distinguish from the usual set notation) as $\{\{a_1, \dots, a_t\}\}$.

Now we present the basic notions of membrane computing; the interested reader may find more detailed information on the theory of $P$ systems in the monograph [10]. 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 $j$, denoted by $i = parent(j)$, and at the same time, membrane $j$ is one of the child membranes of $i$.

By the contents of a region we mean the multiset of objects which is contained by the corresponding membrane excluding those objects which are contained by any of its child membranes.

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, $parent(i)$. 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 [9], for the use of promoters see [8].)

The rules can be applied in the maximal parallel or in the sequential manner. When they are applied in the sequential manner, one rule is applied in each region in each derivation step, when they are applied in the parallel manner, as many rules are applied in each region as possible. See Definition 2 for the formal description of these modes.

The end of the computation is defined by halting: A $P$ system halts when no more rules can be applied in any of the regions. In the case of $P$ automata, however, we consider predefined accepting configurations called final states, by associating a finite set of multisets to each region. The $P$ automaton accepts the input sequence when the contents of each region coincides with one element of these previously given finite sets of multisets.

The result of the computation can also be given in several ways, see [10] for more details. In the case of $P$ automata, the result of the computation is an accepted multiset sequence, the sequence of multisets entering the skin membrane during a successful computation.

Now we present the formal definition of a $P$ automaton.

**Definition 1.** A $P$ *automaton* with $n$ membranes is defined as $\Gamma = (V, \mu, (w_1, P_1, F_1), \ldots, (w_n, P_n, F_n))$ where $n \geq 1$, $V$ is a finite alphabet of objects, $\mu$ 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 $\Gamma$ describes a *configuration* of $\Gamma$; $(w_1, \ldots, w_n) \in (V^\circ)^n$ is the initial configuration.

**Definition 2.** The transition mapping of a $P$ automaton is a partial mapping $\delta_X : V^\circ \times (V^\circ)^n \to 2^{(V^\circ)^n}$, with $X \in \{seq, par\}$ for sequential or for parallel rule application. These mappings are defined implicitly by the rules of the rule sets $P_i,\ 1 \leq i \leq n$. For a configuration $(u_1, \ldots, u_n)$,

$$(u_1', \ldots, u_n') \in \delta_X(u, (u_1, \ldots, u_n))$$

holds, that is, while reading the input $u \in V^\circ$ the automaton may enter the new configuration $(u_1', \ldots, u_n') \in (V^\circ)^n$, if there exist rules as follows.

- If $X = seq$, then for all $i, 1 \leq i \leq n$, there is a rule $(x_i, in; y_i, out)|_{Z_i} \in P_i$ with $z \subseteq u_i$ for $Z_i = z \in V^\circ$, and $z \cap u_i = \epsilon$ for $Z_i = \neg z, z \in V^\circ$, satisfying the conditions below, or
- if $X = par$, then for all $i, 1 \leq i \leq n$, there is a multiset of rules $R_i = \{\{r_{i,1}, \ldots, r_{i,m_i}\}\}$, 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 $r \in P_j$, for any $j,\ 1 \leq j \leq n$, such that 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

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

We define the sequence of multisets of objects accepted by the $P$ automaton as an input sequence which is consumed by the skin membrane 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., $u_j \in F_j$.

Note that in the case of parallel rule application, the set of multisets which may enter the system in one step is not necessarily bounded, thus, this type of automata may work with strings over infinite alphabets. In this paper however, we study languages over finite alphabets, so we apply a mapping to produce a finite set of symbols from a possibly infinite set of multisets, and in order not to "encode" the computational power in this mapping, we assume that it is computable by a linear space bounded Turing machine.

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

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

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

**Definition 4.** Let $\Gamma$ be a *P* automaton as above with initial configuration $(w_1, \ldots, w_n)$ and let $\Sigma$ be a finite alphabet. The *language accepted by* $\Gamma$ in the sequential way of rule application, $L_{seq}$, or in the maximal parallel way of rule application, $L_{par}$, is

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

for $X \in \{seq, par\}$, and for a linear space computable mapping $f : V^\circ \longrightarrow \Sigma \cup \{\varepsilon\}$ with $f(x) = \varepsilon$ if and only if $x = \epsilon$. Let us denote the class of languages accepted by *P* automata with sequential or parallel rule application as $\mathcal{L}_X(PA)$, $X \in \{seq, par\}$.

## 3 The Power of *P* Automata

Now we consider the accepting power of *P* automata. We follow ideas from [5] and [6] in relating this power to well-known machine based complexity classes. There, among other similar models, the so-called symport/antiport *P* system acceptors are studied. These are accepting membrane systems similar to *P* automata, the main difference in the two models is the fact that the alphabet of symport/antiport acceptors is divided into a set of terminals and nonterminals. During the work of these systems both types of objects may leave or enter the membrane structure but only the objects which are terminal constitute the part of the input sequence which is accepted in a successful computation. Thus, the nonterminal objects are used to provide additional workspace for the computation.

This feature motivated the introduction of so-called $S(n)$ space bounded symport/antiport acceptors, systems where the total number of objects used in an accepting computation on a sequence of length $n$ is bounded by a function $S(n)$. As shown in [5] and [6], a language $L$ is accepted by an $n^k$ space bounded symport/antiport acceptor, if and only if, it is accepted by a nondeterministic $\log n$ space bounded one-way Turing machine, or by a $c^n$ space bounded symport/antiport acceptor, if and only if it is accepted by a one-way linear space bounded Turing machine, that is, if and only if it is context-sensitive.

Since in $P$ automata the workspace is provided by the objects of the accepted or rejected input only, the maximal number of objects present inside the membrane structure during a computation is bounded by the length of the input sequence. Furthermore, even this "space" can only be used with a strong restriction since it becomes available step-by-step, as more and more symbols of the input are read. Thus, when looking for a Turing machine model corresponding to $P$ automata, some restriction on the use of the available workspace is necessary.

**Definition 5.** A nondeterministic one-way Turing machine is *restricted* $S(n)$ *space bounded* if for every accepted input of length $n$, there is an accepting computation where the number of nonempty cells on the work-tape(s) is bounded by $S(d)$ where $d \leq n$, and $d$ is the number of input tape cells already read, that is, the *distance* of the reading head from the left end of the one-way input tape.

Let $\mathcal{L}(1LOG)$, $\mathcal{L}(1LIN)$, $\mathcal{L}(\mathit{restricted}{-}1LOG)$, and $\mathcal{L}(\mathit{resticted}{-}1LIN)$ denote the class of languages accepted by one-way nondeterministic Turing machines with logarithmic space bound, linear space bound, restricted logarithmic space bound, and restricted linear space bound, respectively.

Let $L$ denote the language

$$L = \{xy \mid x \in \{1, 2, \ldots, 9\}\{0, 1, \ldots, 9\}^*, y \in \{\#\}^+, \text{ with } val(x) = |y|\}$$

where $val(x)$ is the value of $x$ as a decimal number.

As we shall see later, $\mathcal{L}(\mathit{restricted} - 1LIN) = \mathcal{L}(1LIN)$, but the class $\mathcal{L}(\mathit{restricted} - 1LOG)$ is strictly a subclass of $\mathcal{L}(1LOG)$ since $L$, the language defined above, is in the latter class but not in the former. Still, $\mathcal{L}(\mathit{restricted} - 1LOG)$ contains some very interesting languages, e.g., $\{a^n b^n c^n \mid n \geq 1\}$ and $\{a^{2^n} \mid n \geq 0\}$ are both in $\mathcal{L}(\mathit{restricted} - 1LOG)$, as they can be accepted by Turing machines capable of recording the distance of the reading head from the left-end of the one-way input tape which can be achieved in restricted logarithmic space.

**Theorem 1.**

$$\mathcal{L}_{seq}(PA) = \mathcal{L}(\mathit{restricted} - 1LOG) \text{ and } \mathcal{L}_{par}(PA) = \mathcal{L}(\mathit{restricted} - 1LIN).$$

*Proof.* First we prove the inclusions from left to right in both equations. Consider the $P$ automaton $\Gamma = (V, \mu, (w_1, P_1, F_1), \ldots, (w_n, P_n, F_n))$, $n \geq 1$, with $L(\Gamma) \subseteq \Sigma^*$ where $f : V^\circ \longrightarrow \Sigma \cup \{\varepsilon\}$ is a linear space computable mapping with $f(x) = \varepsilon$ if and only if $x = \epsilon$. We show how to construct a one-way Turing machine $M$ which simulates the work of $\Gamma$ using restricted logarithmic space if $\Gamma$ applies the rules sequentially, or restricted linear space if $\Gamma$ applies the rules in the maximal parallel manner. Let $M = (k, \Sigma, A, Q, q_0, q_F, \delta_M)$ be a Turing machine with a one-way read only input tape where

- $k = (|V| \cdot n^2 + |V|)$ is the number of work-tapes,
- $\Sigma$ is the finite input alphabet,
- $A = \{0, \ldots, 9\}$ is the work-tape alphabet,

– $Q$ is the set of internal states, $q_0, q_F \in Q$ are the starting and the final states, and
– $\delta_M$is the transition function of $M$.

Let $M$ have $n$ work-tapes assigned to each region and symbol pair $(i, a) \in \{1, \dots, n\} \times V$, and an additional tape for each symbol of $V$. Using the digits of the tape alphabet, $\{0, \dots, 9\}$, $M$ keeps track of the configurations of $\Gamma$ by having an integer written on the first one of the work-tape $n$-tuple assigned to $(i, a)$ denoting the number of $a$ objects present in region $i$.

Let these configurations of $M$ be denoted as

$$(q, w, \alpha_{1,1}, 0^{n-1}, \dots, \alpha_{n,|V|}, 0^{n-1}, 0^{|V|})$$

where $q \in Q$ is the current state, $w \in \Sigma^*$ is the part of the input that is not yet read, $(\alpha_{i,j}, 0^{n-1}) \in (A^+)^n, 1 \leq i \leq n,\ 1 \leq j \leq |V|$, are the values written on the work-tape $n$-tuple corresponding to region $i$ and symbol $a_j \in V$, the value of $\alpha_{i,j}$ denoting the number of such objects present in region $i$. For the sake of notational convenience, in the following we will use $\alpha_{i,j}$ to represent both the string of digits on the work-tape and the decimal value of this string.

Now let us consider the transitions of $\Gamma$. Let $\delta_\Gamma$ be $\delta_{seq}$ or $\delta_{par}$ as defined above for the case of sequential or maximal parallel way of rule application. The transition function of $M$ is defined in such a way that if and only if

$$(u'_1, \dots, u'_n) \in \delta_\Gamma(v, (u_1, \dots, u_n)),$$

then

$$(q, w, \alpha_{1,1}, 0^{n-1}, \dots, \alpha_{n,|V|}, 0^{n-1}, 0^{|V|}) \longrightarrow \\ (q, w', \alpha'_{1,1}, 0^{n-1}, \dots, \alpha'_{n,|V|}, 0^{n-1}, 0^{|V|})$$

is a possible transition in $M$ where $\alpha_{i,j} = |u_i|_{a_j}$ and $\alpha'_{i,j} = |u'_i|_{a_j}$ for all $1 \leq i \leq n,\ 1 \leq j \leq |V|$, and if $f(v) = a$ then $w = aw'$, or if $v = \epsilon$, then $w = w'$. Note that this is possible because the finite set of rules can be encoded in the finite control, and all the information necessary to record a configuration of the $P$ automaton is stored on the work-tapes. First, for each region and symbol pair, $(i, a)$, $M$ writes the number of $a$ objects leaving from region $i$ to region $j$ to the $j$th tape of the work-tape $n$-tuple corresponding to $(i, a)$, and also records the number of symbols entering from the environment using the $|V|$ additional work-tapes. Then in a final round, it creates the description of the new configuration by adding the appropriate values to the integers stored on the first tapes of each work-tape $n$-tuple, and using the collection of objects, $v \in V^\circ$, which enter from the environment, $M$ computes $f(v) = a \in \Sigma$ and reads $a$ from the input tape.

If $\Gamma$ is a sequential $P$ automaton, then this whole process can be realized in restricted logarithmic space since the number of cells used on the work-tapes is the logarithm of the number of objects present in the $P$ system which is at most $c \cdot d$ where $c$ is some constant and $d$ is the number of nonempty multisets read by the $P$ automaton, or in terms of the Turing machine, the distance of the reading head from the left end of the input tape. Furthermore, the function

$f$ maps a finite domain to a finite set of values, so its computation does not require any additional space. If $\Gamma$ works in the maximal parallel manner, then the computation of $M$ requires restricted linear space because the number of symbols inside the $P$ systems is at most $c^d$ with $c, d$ as above, so the integers describing the configurations can be represented by decimal numbers in restricted linear space. The values of $f(x) \in \Sigma \cup \{\varepsilon\}$ can also be computed inside the restricted linear space bound, since the cardinality of any $x \in V^\circ$ for which the computation is needed is at most $c^d$, and the computation of $f(x)$ itself uses linear space measured in the size of the argument, so it is still restricted linear.

The transition function of $M$ should also enable an initialization phase,

$$(q_0, w, \varepsilon, \ldots, \varepsilon) \longrightarrow (q, w, \alpha_{1,1}, 0^{n-1}, \ldots, \alpha_{n,m}, 0^{n-1}, 0^{|V|})$$

where $\alpha_{i,j} = |w_i|_{a_j},\ 1 \leq i \leq n,\ 1 \leq j \leq |V|$.

The input is accepted by $M$ if and only if it is accepted by $\Gamma$, that is,

$$(q, \varepsilon, \alpha_{1,1}, 0^{n-1}, \ldots, \alpha_{n,m}, 0^{n-1}, 0^{|V|}) \longrightarrow (q_F, \varepsilon, \alpha_{1,1}, 0^{n-1}, \ldots, \alpha_{n,m}, 0^{n-1}, 0^{|V|})$$

where for each $F_i \neq \emptyset$, there is an $u_i \in F_i$ such that for all $a_j \in V,\ \alpha_{i,j} = |u_i|_{a_j}$. The precise construction of $M$ is left to the reader.

Now we prove the inclusions from right to left. To do this we need the notion of a two-counter automaton. A *two-counter machine* is an automaton with a one-way read only input tape and two counters capable of storing any non-negative integer. Formally it can be given as $M = (\Sigma, Q, q_0, q_F, \delta_M)$ where $\Sigma$ is an input alphabet, $Q$ is a set of internal states containing the initial and accepting states $q_0, q_F \in Q$ respectively, and $\delta_M$ is a transition function which maps the quadruple of state, input symbol, and zero or non-zero counter contents to the triple of a new state and two instructions to increment, unchange, or decrement the counters. As the work-tapes of any Turing machine can be simulated with two counters, two-counter machines accept the class of recursively enumerable languages (see [2]).

However, if the sum of the counter contents is bounded, that is, the two-counter automaton has limited workspace, then its power is decreased. If we define $S(n)$ space bounded two-counter machines as $S(n)$ being the bound on the sum of the counter contents during any accepting computation on an input of length $n$, then we obtain a model equivalent to $\log S(n)$ space bounded one-way Turing machines because an integer $i$ stored in a counter can be written on the work-tapes using $\log i$ tape cells. We may also introduce the restricted $S(n)$ space bounded variant exactly as above, in which case we obtain a machine equivalent to restricted $\log S(n)$ space bounded one-way Turing machines.

Consider now a two-counter machine $M$. If $x$ is an element of the domain of $\delta_M$, then a transition is given by the pair $(x, \delta_M(x))$. Let these pairs be labelled by elements of the finite set of labels $TRANS$.

Let $\Gamma = (V, \mu, (w_1, P_1, \emptyset), (w_2, P_2, F_2), (w_3, P_3, \emptyset), (w_4, P_4, \emptyset))$ be a $P$ automaton with the membrane structure $\mu = [_1\ [_2\ ]_2\ [_3\ ]_3\ [_4\ ]_4\ ]_1$. Inside the skin membrane, it has a controlling region, region 2, for storing and manipulating symbols corresponding to the states of $M$, and a pair of membranes, 3 and 4, for maintaining the values of the two counters.

Let $V = \Sigma \cup \{\langle q\rangle, \langle t\rangle, \langle t\rangle_a \mid q \in Q,\ t \in \mathit{TRANS},\ a \in \Sigma\}$. The symbols of $V - \Sigma$ govern the work of $\Gamma$. The presence of $\langle q\rangle \in V$ in the skin membrane indicates that $\Gamma$ simulates a configuration of the two-counter machine when it is in state $q \in Q$. While simulating a transition from state $q$ to state $q'$ labelled by $t \in \mathit{TRANS}$, $\Gamma$ needs extra steps for reading the input and manipulating the symbols which keep track of the counter values. During these steps the skin membrane contains one of the symbols $\langle t\rangle$, $\langle t\rangle_a$, for some $a \in \Sigma$, and when the simulation of the transition is complete, $\langle q'\rangle \in V$ appears in the skin membrane.

The simulation of $M$ starts in the initial state with $w_1 = \langle q_0\rangle\langle q_0\rangle\langle q_0\rangle a$, for some $a \in \Sigma$, and for $2 \le i \le 4$, $w_i = \{\{\ \langle q\rangle, \langle t\rangle, \langle t\rangle_a \mid q \in Q,\ q \ne q_0,\ t \in \mathit{TRANS},\ \ a \in \Sigma\ \}\}$. The rule sets belonging to the regions are as follows. Let

$$\begin{aligned} P_1 = \{&(\epsilon, in)|_{\langle q\rangle\langle q\rangle\langle q\rangle},\ (\epsilon, in)|_{\langle t\rangle_a\langle t\rangle_a\langle t\rangle_a} \mid q \in Q,\ t \in \mathit{TRANS},\ a \in \Sigma\} \cup \\ &\{(x^k, in; y, out)|_{\langle t\rangle\langle t\rangle\langle t\rangle} \mid x \in \Sigma,\ y \in \Sigma,\ t \in \mathit{TRANS},\ x \text{ is read} \\ &\quad \text{during the transition } t\} \cup \\ &\{(\epsilon, in)|_{\langle t\rangle\langle t\rangle\langle t\rangle} \mid t \in \mathit{TRANS},\ \varepsilon \text{ is read during the transition } t\}, \end{aligned}$$

and $k \ge 1$ is a suitable constant. In the case of sequential rule application, if transition $t$ is simulated, as indicated by the presence of $\langle t\rangle$, then $k$ copies of the corresponding input symbol are read into the skin membrane, and one other symbol is sent out in the first simulating step. If the rules are applied in the maximal parallel manner, then after the first simulating step, the number of symbols in the skin region is $k$ times as much, as the number that was already present.

Before the simulation starts, that is, when a state symbol $\langle q\rangle$ is present, or in the later simulating steps, when the symbols $\langle t\rangle_a$, for some $a \in \Sigma$, are present, then nothing is read from the input.

In what follows, let us assume that $\Sigma = \{a_1, \ldots, a_m\}$. Now let

$$\begin{aligned} P_2 = \{&(\langle q\rangle, in; \langle t\rangle, out), (\langle t\rangle, in; \langle t\rangle_{a_1}, out), (\langle t\rangle_{a_i}, in; \langle t\rangle_{a_{i+1}}, out), \\ &(\langle t\rangle_{a_m}, in; \langle q'\rangle, out) \mid 1 \le i \le m-1,\ t \in \mathit{TRANS} \text{ is a} \\ &\text{transition from } q \text{ to } q'\}, \end{aligned}$$

and let $F_2 = \{\ \{\{\ \langle q\rangle, \langle t\rangle, \langle t\rangle_a \mid q \in Q,\ q \ne q_F,\ t \in \mathit{TRANS},\ \ a \in \Sigma\ \}\}\ \}$. The second region is responsible for keeping track of the simulated states and transitions. The simulation is finished if $q_F$, the final state is exported from this region to the first one as indicated by $F_2$ above. Now for $3 \le i \le 4$, let

$$\begin{aligned} P_i = \{&(\langle q\rangle, in; \langle t\rangle x, out), (\langle t\rangle, in; \langle t\rangle_{a_1}, out), (\langle t\rangle_{a_j}, in; \langle t\rangle_{a_{j+1}}, out), \\ &(\langle t\rangle_{a_m}, in; \langle q'\rangle, out) \mid x \in \Sigma,\ 1 \le j \le m-1,\ t \in \mathit{TRANS} \\ &\text{is a transition from } q \text{ to } q' \text{ decreasing the } i\text{th counter } \} \cup \\ &\{(\langle q\rangle, in; \langle t\rangle, out), (\langle t\rangle, in; \langle t\rangle_{a_1}, out)|_{Z_1}, (\langle t\rangle_{a_j}, in; \langle t\rangle_{a_{j+1}}, out)|_{Z_{j+1}}, \\ &(\langle t\rangle_{a_m} x, in; \langle q'\rangle, out) \mid 1 \le j \le m-1,\ t \in \mathit{TRANS} \text{ is a transition} \end{aligned}$$

from $q$ to $q'$, and
$x \in \Sigma$ if the $(i-2)$th counter is increased during $t$, or
$x = \varepsilon$ if the $(i-2)$th counter is not changed during $t$, or
$Z_j = \varepsilon$ for all $a_j \in \Sigma$ if the $(i-2)$th counter can be nonempty before $t$, or
$Z_j = \neg a_j$ for all $a_j \in \Sigma$ if the $(i-2)$th counter must be empty before $t\}$.

Region 3 and 4 keep track of the values stored in the counters of $M$ by the help of the rules above. Together with moving the transition symbols $\langle t \rangle_a$, for some $a \in \Sigma$, they import and export the symbols originating from the input as necessary to maintain the correct counter contents, the emptiness of the counters are checked by the use of the forbidding promoters.

In the case of sequential rule application, the possible inputs are the multisets containing the elements of $\Sigma$ in $k$ copies, so we can use the mapping $f_{seq}(\{\{x^k\}\}) = x,\ x \in \Sigma$ to map $V^\circ$ to $\Sigma$. In the parallel case, the input multisets are of the form $x^{ik}$ for some $x \in \Sigma,\ i \geq 1$, so we might use $f_{par}(x^{ik}) = x$, for all $i \in \mathbb{N}$, to produce the string corresponding to an input sequence.

Now, if $M$ is restricted $S(n)$ space bounded with $S(n) = c \cdot n$ for a constant $c$, then it can be simulated by the $P$ automaton $\Gamma$ with sequential rule application, since if $k \geq c+1$, then the number of objects (the bound on the sum of the counter values) is $d(k-1)+1 \geq d \cdot c$ where $d$ is the number of already read nonempty multisets. If $M$ is restricted $S(n)$ space bounded with $S(n) = c^n$ for a constant $c$, then it can be simulated by $\Gamma$ with parallel rule application, since if $k \geq 2 \cdot c$, then the amount of imported objects during the computation is sufficient to make sure that in each step the number of available objects which can be used to keep track of the counter values is $c^d$, where $d$ is the number of nonempty multisets read. Since restricted $S(n)$ space bounded two-counter machines are equivalent to restricted $\log S(n)$ space bounded one-way Turing machines, our statement is proved. □

Based on this theorem, we can show that the class of languages accepted by $P$ automata in the sequential way is strictly included in the class of languages accepted by logarithmic space bounded one-way Turing machines, that is, in $\mathcal{L}(1LOG)$.

**Theorem 2.** $\mathcal{L}_{seq}(PA) \subset \mathcal{L}(1LOG)$.

*Proof.* To prove our statement we show that $\mathcal{L}(restricted - 1LOG)$ is strictly included in $\mathcal{L}(1LOG)$. The inclusion is obvious, so it is enough to show that the difference of the two classes is nonempty. Consider the language $L$ as defined above in Section 2. It is clear that $L$ can be accepted by a one-way Turing machine using logarithmic space: First the initial part, $x \in \{1,\ldots,9\}\{0,\ldots,9\}^*$, of the input is copied to a work-tape then the number of # symbols are checked against this integer by reading further and decreasing the stored integer by one in each step.

To see that $L$ cannot be accepted by a one-way restricted logarithmic space bounded Turing machine, suppose that $M$ is such a machine accepting $L$. Without loss of generality, assume that $M$ has only one work-tape. Then there is a constant $c$ (which only depends on the specification of $M$) such that for every $k$, after reading $k$ input symbols, $M$ uses at most $c \cdot \log k$ cells on the work-tape. (All logarithms are base 10.) Also, the number of possible configurations on a work-tape of at most $c \cdot \log k$ cells (a configuration is a triple consisting of the state, work-tape contents, work-tape head position) is at most $d^{\log k} = k^{\log d} \leq k^e$ for some constants $d$ and $e$. Now for any given $k$, consider the set of all strings of the form: $w\#^{val(w)}$, where $|w| = k$. Clearly, $k \leq g \cdot \log n$ for some constant $g$, where $n = |w\#^{val(w)}|$. Since the machine is one-way restricted logarithmic space bounded, there exist an $n$ (and therefore $k$) big enough and two inputs $w\#^{val(w)}$ and $w'\#^{val(w')}$ with $|w| = |w'| = k$ and $w \neq w'$ for which $M$ will be in the same configuration after reading $w$ and after reading $w'$. This is because $M$ can be in one of at most $k^e \leq (g \cdot \log n)^e$ configurations after reading a string of length $k$. Now there are $10^k \geq h \cdot n$ strings $w$ whose lengths are $k$ for some positive constant $h$. If we choose $n$ big enough, $(g \cdot \log n)^e < h \cdot n$. It follows that the machine when given the string $w'\#^{val(w)}$ will also accept. This is a contradiction since $w'\#^{val(w)}$ is not in $L$. □

It is interesting to look at the closure properties of $\mathcal{L}_{seq}(PA)$ which, as shown in Theorem 1 is identical to $\mathcal{L}(restricted - 1LOG)$.

**Theorem 3.** *$\mathcal{L}_{seq}(PA)$ is closed under union, intersection, concatenation, Kleene $^*$ and $^+$, inverse homomorphism, and $\varepsilon$-free homomorphism. It is not closed under (unrestricted) homomorphism, reversal, and complementation.*

*Proof.* It is straightforward to verify that $\mathcal{L}_{seq}(PA) = \mathcal{L}(restricted - 1LOG)$ is closed under union, intersection, concatenation, Kleene $^*$ and $^+$, inverse homomorphism, and $\varepsilon$-free homomorphism. By using $L' = \{xyz \mid x \in \{\&\}^+, y \in \{1, \ldots, 9\}\{0, \ldots 9\}^*, z \in \{\#\}^+, |x| = |z| = val(y)\}$, a variant of the language defined above, it is easy to see that $\mathcal{L}(restricted - 1LOG)$ is not closed under unrestricted homomorphism, since erasing the symbol & produces $L$ from $L'$. It is also not closed under reversal since the reverse of $L$ is clearly in $\mathcal{L}(restricted - 1LOG)$, but as we have seen, $L$ is not. To see that $\mathcal{L}(restricted - 1LOG)$ is not closed under complementation, consider $\bar{L}$, the complement of $L$. $\bar{L}$ can be accepted by a one-way Turing machine $M'$ using restricted logarithmic space, as follows. Inputs that do not have the form $xy$ with $x \in \{1, \ldots, 9\}\{0, \ldots, 9\}^*$, $y \in \{\#\}^+$ can easily be accepted by $M'$. For an input of the form $xy$, $M'$ needs to check that $val(x) \neq |y|$. To do this, $M'$ scans the segment $x$ using its work-tape as a counter (in base 10) to record the position of the input head as it scans $x$. At some point, nondeterministically chosen, $M'$ stops incrementing the counter and remembers the symbol $d$ under the input head. Note that at this point, the counter has value $\log i$ and $d$ is the $i$-th symbol of $x$. $M'$ then scans the segment $y$ while recording the length of $y$ on another work-tape (again use as a counter in base 10). Let $w$ be the count after processing $y$. $M'$ then checks that the symbol in position $i$ of $w$ is not equal to $d$. Clearly $M'$ accepts $\bar{L}$. Since $L$ is not in $\mathcal{L}(\mathit{restricted}-1LOG)$, it follows that $\mathcal{L}(\mathit{restricted}-1LOG)$ is not closed under complementation. □

Now consider the *deterministic* version of a one-way restricted logarithmic space bounded Turing machine. It is easy to show that $\mathcal{L}(\mathit{restricted}{-}1DLOG)$ is closed under complementation. Hence, from Theorem 2 we have $\mathcal{L}(\mathit{restricted}{-}1LOG)-\mathcal{L}(\mathit{restricted}-1DLOG) \neq \emptyset$. This is an interesting example of nondeterminism being better than determinism for a restricted type of space-bounded Turing machine.

Unlike in the case of the logarithmic space bound, the restricted use of linear space does not influence the power of a linear space bounded Turing machine. To see this, consider the machine which first copies its input to an additional work-tape, then works with it as with the input tape, and with the rest of its work-tapes exactly as before. This machine uses restricted linear space, and it clearly accepts the same set of input words as before. Thus, since linearly bounded Turing machines characterize the class of context-sensitive languages, we have:

**Theorem 4.** $\mathcal{L}_{par}(PA) = \mathcal{L}(1LIN) = \mathcal{L}(CS)$.

## References

1. Csuhaj-Varjú, E., Vaszil, Gy.: *P* Automata. In: Păun, Gh., Zandron, C. (eds.): Pre-Proceedings of the Workshop on Membrane Computing WMC-CdeA 2002, Curtea de Argeş, Romania, August 19-23, 2002. Pub. No. 1 of MolCoNet-IST-2001-32008 (2002) 177-192, and also in
   Păun, Gh., Rozenberg, G., Salomaa, A., Zandron, C. (eds.): Membrane Computing. Lecture Notes in Computer Science, Vol. 2597. Springer, Berlin (2003) 219-233
2. Fischer, P. C.: Turing Machines with Restricted Memory Access. Information and Control **9** (1966) 364-379
3. Freund, R., Martín-Vide, C., Obtułowicz, A., Păun, Gh.: On Three Classes of Automata-like P Systems. In: Ésik, Z., Fülöp, Z. (eds.): Developments in Language Theory. 7th International Conference, DLT 2003, Szeged, Hungary, July 2003. Proceedings. Lecture Notes in Computer Science, Vol. 2710. Springer, Berlin (2003) 292-303
4. Freund, R., Oswald, M.: A Short Note on Analysing P Systems. Bulletin of the EATCS **78** (October 2002) 231-236
5. Ibarra, O. H.: On the Computational Complexity of Membrane Systems. To appear in Theoretical Computer Science C
6. Ibarra, O. H.: The Number of Membranes Matters. In: Alhazov, A., Martín-Vide, C., Păun, Gh. (eds.): Workshop on Membrane Computing, WMC-2003, Tarragona, July 17-22, 2003. Technical Report 28/03 of the Research Group on Mathematical Linguistics, Rovira i Virgili University, Tarragona, Spain (2003) 273-285
7. Madhu, M., Krithivasan, K.: On a Class of P Automata. Submitted
8. Martín-Vide, C., Păun, A., Păun, Gh.: On the Power of P Systems with Symport Rules. Journal of Universal Computer Science **8**(2) (2002) 317-331

9. Păun, A., Păun, Gh.: The Power of Communication: P Systems with Symport/Antiport. New Generation Computing **20**(3) (2002) 295-306
10. Păun, Gh.: Computing with Membranes: An Introduction. Springer, Berlin, (2002)
11. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages. Springer-Verlag, Berlin, vol. 1-3, (1997)