On Three Classes of Automata-Like P Systems

Rudolf FREUND[1], Carlos MARTÍN-VIDE[2], Adam OBTUŁOWICZ[3], and Gheorghe PĂUN*[42]

[1] Department of Computer Science, Technische Universität Wien
Favoritenstraße 9, A-1040 Wien, Austria
E-mail: rudi@emcc.at
[2] Research Group on Mathematical Linguistics, Rovira i Virgili University
Pl. Imperial Tárraco 1, 43005 Tarragona, Spain
E-mail: cmv@astor.urv.es
[3] Institute of Mathematics of the Polish Academy of Sciences
Śniadeckich 8, PO Box 137, 00-950 Warsaw, Poland
E-mail: adamo@impan.gov.pl
[4] Institute of Mathematics of the Romanian Academy
PO Box 1-764, 70700 Bucureşti, Romania
E-mail: gpaun@imar.ro, gp@astor.urv.es

Abstract. We investigate the three classes of accepting P systems considered so far, namely the P automata of Csuhaj-Varju, Vaszil [3], their variant introduced by Madhu [10], and the related machinery of Freund, Oswald [5]. All three are based on symport/antiport rules. For slight variants of the first two classes we prove that any recursively enumerable language can be recognized by systems with only two membranes (this considerably improves the result from [3], where systems with seven membranes were proved to be universal). We also introduce the initial mode of accepting strings (the strings are introduced into the system, symbol by symbol, at the beginning of a computation), and we briefly onvestigate this mode for the three classes of automata. Some open problems are formulated too.

1 Introduction

Membrane computing is a branch of natural computing which aims to devise distributed parallel computing devices that are abstracted from cell functioning. We refer to [13] and to the web page http://psystems.disco.unimib.it for details and for references. In short, in the compartments of a membrane structure we place multisets of symbol-objects which evolve by means of local rules. Applying the rules in a non-deterministic, maximally parallel way, the system passes from one configuration to another one, thereby performing a computation. Only halting computations produce a result, which consists of the objects present in a specified output membrane.

* Work done in the framework of the Contract No ICA1-CT-2000-70024 between IMPAN, Warsaw, Poland, and the European Community

An interesting class of P systems is that based on purely communicative rules, using so-called symport/antiport rules, [12]. Specifically, we use rules for transfering objects through membranes corresponding to the symport and antiport known in biology: symport is the process of passing two molecules through a membrane, in the same direction, while antiport refers to the passage of two molecules in opposite directions. In mathematical (generalized) terms, a symport rule is of the form (x, in) or (x, out), where x is a string representing the multiset of objects which enter, respectively exit, the membrane, while an antiport rule is of the form $(x, out; y, in)$, with x indicating the objects which exit the membrane at the same time when the objects indicated by y enter the membrane. Such rules can be also used in a conditional manner. For instance, a symport rule written in the form $(x, in)|_z$ is used to introduce the elements of x in a membrane i only if the elements of z are already present in membrane i; we say that z acts as a *promoter*.

As sketched above, a P system is used to *generate* a set of numbers or of strings. In the present paper we deal with the automata-like approach, using a P system as an acceptor, more precisely, as a string acceptor. The idea is not new, it was already explored for the first time in [3], and then in [10] and [5].

In [3] the so-called P automata are introduced, as membrane systems using conditional symport rules of the form $(y, in)|_x$, with a very important difference in defining the computation steps: the multisets are processed in a sequential manner, at most one rule is applied in each region of the system. This is a feature very useful in programming computations in such systems. On the other hand, note the strong restriction that all rules use the command *in*, hence the communication is done in a one-way manner, top-down. Modulo a projection of sequences of multisets to strings (one selects only the symbols from a given terminal alphabet), these devices are shown in [3] to be universal: any recursively enumerable language can be recognized by a P automaton (with at least seven membranes). Actually, some further features are present in the systems from [3], but we do not consider them here. We improve the result from [3]: two membranes suffice (and this is an optimal result, because one-membrane systems can never halt after starting a computation).

A variant of the devices from [3] was proposed in [10], this time closer to automata style: one considers both objects and states, and rules of the form $(qy, in)|_{px}$, where p, q are states and x, y are mutisets of objects. Each region has associated only one state-object. A multiset which enters the system during a computation is accepted if the system halts in a final state. Systems of this type with two membranes are proven in [10] to accept all recursively enumerable sets of natural numbers. We extend here this result to languages, even for systems with restricted forms of rules.

Note the similarities and differences of the systems of the two types above with the multiset automata from [2], where one uses multiset rewriting-like rules of the form $px \to qy$. It is also worth mentioning that the conditional symport rules are a particular form of "boundary rules", as introduced in [1], where rules of the form $y[_i x \to [_i xy$ were considered, with the obvious meaning that y enters

region i providing that x is already there. Thus, our universality results extend also to systems from [1].

A similar type of string accepting devices was considered in [5], but with a simpler (standard) definition: take usual P systems with symport/antiport rules, and accept the multiset of terminal symbols taken from the environment during a halting computation. The universality is again obtained, this time by systems with only one membrane, using antiport rules only.

In the present paper we also consider a restricted version of the devices of all these three types, which brings the accepting P systems closer to the way the usual automata work and which we call *initial*: the symbols of a string are introduced into the system one by one, in the first steps of the computation; further symbols can be introduced, at the same time or later; the string is accepted if the computation eventually halts.

For the devices from [3] and for a simplified variant of systems from [10] where however we allow several state-objects to be present in the same region, we prove that any one-letter recursively enumerable language can be recognized in the initial mode by systems with only two membranes, and with only two objects present in each symport rule.

The case of one-letter alphabet is also considered in [9], where the one-letter recursively enumerable languages are characterized by means of P automata as in [3] with two (three) membranes and symport rules of weight three (two, respectively), without using promoters.

2 Prerequisites; Register Machines

We use the standard formal language theory notation and terminology. In particular, by $|x|$ we denote the length of the word x over V. For $x \in V^*$ and $U \subseteq V$, by $|x|_U$ we denote the number of occurrences in x of symbols from U. For any family of languages FL we denote by $1FL$ the family of one-letter languages from FL and by NFL the family of length sets of languages in FL.

Several times in the paper we will make use of representing a string over $V = \{a_1, \ldots, a_k\}$, by its value in base $k+1$. Specifically, if $w = a_{i_1} a_{i_2} \ldots a_{i_n}$, $1 \leq i_j \leq k$ for all $1 \leq j \leq n$, then $val_{k+1}(w) = i_1 \cdot (k+1)^{n-1} + \ldots + i_{n-1} \cdot (k+1) + i_n$. Note that $val_{k+1}(w) = val_{k+1}(a_{i_1} \ldots a_{i_{n-1}}) \cdot (k+1) + i_n$.

Convention. When comparing two language generating/accepting devices we ignore the empty string and when comparing two natural numbers computing/accepting devices we ignore the number 0.

A universal computational model very useful in our framework are register machines (see [11] for some original definitions, [9] for variants (called counter automata), and [7], [8] for definitions like that we use in this paper).

An *n-register machine* is a construct $M = (n, R, q_s, q_h)$, where:

- n is the number of registers,
- R is a set of labelled instructions of the form $q_1 : (op(r), q_2, q_3)$, where $op(r)$ is an operation on register r of M, and q_1, q_2, q_3 are labels from a given set $lab(M)$ (the labels are associated in a one-to-one manner to the instructions),

- q_s is the initial/start label, and
- q_h is the final label.

The machine is capable of the following instructions:

- $(A(r), q_2, q_3)$: Add 1 to the contents of register r and proceed to any of the instructions (labelled with) q_2 and q_3.
- $(S(r), q_2, q_3)$: If register r is not empty, then subtract 1 from its contents and go to the instruction q_2, otherwise proceed to instruction q_3.
- *halt*: Stop the machine. This instruction can only be assigned to the final label q_h.

A register machines M is said to compute a partial function $f : \mathbf{N} \longrightarrow \mathbf{N}$ if, starting with any number m in register 1 and with instruction q_s, M halts in the final label q_h with register 1 containing $f(m)$; if the final label cannot be reached, then $f(m)$ remains undefined.

A register machine can also recognize an input number $m \in \mathbf{N}$ placed in register 1: m is accepted if the machine stops by the halt instruction. If the machine does not halt, then the analysis was not successful.

The register machines (with a small number of registers, but this fact is not of interest for what follows) are know to be computationally universal, equal in power to Turing machines. In our proofs, we will make use of the following two propositions.

Lemma 1. *Each recursively enumerable set of natural numbers can be recognized by a register machine.*

Lemma 2. *For any recursively enumerable language $L \subseteq T^*$, with $card(T) = k$, there exists a register machine M recognizing L in the following way: for every $w \in T^*$, $w \in L$ if and only if M halts when started with $val_{k+1}(w)$ in its first register; in the halting step, all registers of the machine are empty.*

3 P Systems with Symport/Antiport

We start by recalling the "standard" definition of a P system with symport/antiport, as introduced in [12] (hence used to generate natural numbers).

A *P system with symport/antiport rules* is a construct Π of the form

$$\Pi = (V, T, \mu, w_1, \ldots, w_m, E, R_1, \ldots, R_m, i_o),$$

where: (1) V is an alphabet of *objects*, (2) $T \subseteq V$ is the terminal alphabet; (3) μ is a membrane structure (with the membranes labelled by natural numbers $1, \ldots, m$ in a one-to-one manner); (4) $w_1, \ldots, w_m$ are multisets over V associated with the regions $1, \ldots, m$ of μ; (5) $E \subseteq V$ is the set of objects which are supposed to appear in an arbitrarily large number of copies in the environment; (6) $R_1, \ldots, R_m$ are finite sets of symport and antiport rules associated with the membranes $1, \ldots, m$; a symport rule is of the form (x, in) or (x, out), where

$x \in V^+$ (with the meaning that the objects specified by x enter, respectively exit, the membrane), and an antiport rule is of the form $(x, out; y, in)$, where $x, y \in V^+$, which means that the multiset x is sent out of the membrane and y is taken into the membrane region from the surrounding region; (7) i_o is the label of the output membrane, an elementary one in μ.

Starting from the *initial configuration*, which consists of μ and $w_1, \ldots, w_m, E$, the system passes from one configuration to another one by applying the rules from each R_i in a non-deterministic and maximally parallel way. (The environment is supposed inexhaustible, at each moment all objects from E are available in any number of copies we need.) A sequence of transitions is called a *computation*; a computation is *successful* if and only if it halts.

With a successful computation we associate a *result*, in the form of the number of objects from T present in membrane i_0 in the halting configuration. The set of all such numbers computed (we also say *generated*) by Π is denoted by $N(\Pi)$, and the family of all sets $N(\Pi)$, computed by P systems with at most m membranes, with symport rules $(x, in), (x, out)$ with $|x| \leq r$, and antiport rules $(x, out; y, in)$ with $|x|, |y| \leq t$ is denoted by $NOP_m(sym_r, anti_t)$, $m \geq 1$ and $r, t \geq 0$.

Proofs of the following results, which improve previously know results in this area, can be found in [9], [6].

Theorem 1. *$NRE = NOP_m(sym_r, anti_t)$, for $(m, r, t) \in \{(1, 1, 2), (3, 2, 0), (2, 3, 0)\}$.*

With the symport/antiport rules we can also associate *promoters*, in the form $(x, in)|_z$, $(x, out)|_z$, $(x, out; y, in)|_z$, where z is a multiset of objects. Such a rule, associated with membrane i, is applied only if z is present in the region of membrane i. The use of promoters z of length at most u is indicated by adding p_u in front of $sym, anti$, respectively, in the notation of families $NOP_m(sym_r, anti_t)$. The uncontrolled case corresponds to having p_0.

A language accepting version of these systems was considered in [5]: A string w over the alphabet T is recognized by the analysing P system Π if and only if there is a successful computation of Π such that the sequence of terminal symbols taken from the environment during the computation is exactly w. If more than one terminal symbol is taken from the environment in one step, then any permutation of these symbols constitutes a valid subword of the input string. The language of all strings $w \in T^*$ recognized by Π is denoted by $A(\Pi)$. (In this case the output membrane i_o plays no role, hence it can be omitted.)

In the previous mode of analysing a string, if a string w is recognized by Π, then its symbols should exist in E; because each element of E appears in the environment in arbitrarily many copies, we cannot introduce a symbol of w into the system by using a symport rule, because an arbitrarily large number of copies would be introduced, hence the string will not be finite. Antiport rules are thus obligatory. In order to cope with this difficulty, and in order to recognize strings in a way closer to automata style, we consider a restricted mode of accepting strings, called *initial*: take a string $x = x(1)x(2)\ldots x(n)$, with

$x(i) \in T, 1 \leq i \leq n$; in the steps $1, 2, \ldots, n$ of a computation we place one copy of $x(1), x(2), \ldots, x(n)$, respectively, in the environment (together with the symbols of E); in each step $1, 2, \ldots, n$ we request that the symbol $x(1), x(2), \ldots, x(n)$, respectively, is introduced into the system (by a symport or an antiport rule, alone or together with other symbols); after exhausting the string x, the computation may continue, maybe introducing further symbols into the system, but without allowing to the symbols of x to leave the system; if the computation eventually halts, then the string x is recognized. If the system halts before ending the string, or at some step i the symbol $x(i)$ is not taken from the environment, then the string is not accepted. The language of strings accepted by Π in the initial mode is denoted by $A_I(\Pi)$.

It is important to note that in the initial mode of using a P system we can use, say, a rule of the form (a, in) for introducing a symbol of the analysed string into the system: a is not necessarily an element of E, it is only present on "the tape". Contrast this with the non-initial case, when such a rule will cause the computation to continue forever: if a is present in the environment, then it is present in arbitrarily many copies, thus the rule (a, in) can be used indefinitely, because the environment is inexhaustible.

The family of all languages $A_I(\Pi)$, accepted by systems Π with at most m membranes, with symport rules $(x, in), (x, out)$ with $|x| \leq r$, and antiport rules $(x, out; y, in)$ with $|x|, |y| \leq t$, is denoted by $A_I LP_m(sym_r, anti_t), m \geq 1, r, t \geq 0$.

4 The Universality of P Automata

The above-mentioned analysing P systems are a less restrictive variant of the devices considered in [3], where one considers constructs of the form $\Pi = (V, T, \mu, w_1, \ldots, w_m, E, R_1, \ldots, R_m)$, with all components as above, but with the rules from the sets $R_i, 1 \leq i \leq m$, of the form $(y, in)|_x$. Several rules can have the same promoter multiset x or can share objects from their promoting multisets. Moreover, objects which appear in the promoter x of a rule associated with a membrane i may also appear in the "moving multiset" y of a rule associated with a membrane placed inside membrane i.

However, an important difference with the general case of P systems with symport/antiport rules is that in each step, in each region, at most one rule is used; if *at least* one rule can be used, then *one* of them is chosen and used.

As for the analysing P systems defined in the previous section, a string w over T^* is accepted if its symbols are taken from the environment during a halting computation, in the order they appear in w. We denote by $A(\Pi)$ the language of strings accepted by Π in the arbitrary mode, and by $A_I(\Pi)$ the language of strings accepted in the initial mode. We denote by $ALP_m(p_u sym_r, owts)$, $A_I LP_m(p_u sym_r, owts)$ the obtained families of languages; the meaning of all parameters is as usual, while the indication $owts$ refers to the specific form (one-way communication, using only top-down symport rules) and working mode (sequential, one rule per region) of the systems we use.

With our notation, in [3] it is proved that $RE = ALP_7(p_4 sym_6, owts)$. In [9] it is also pointed out that we have $1RE = 1ALP_3(p_0 sym_2, owts) = 1ALP_2(p_0 sym_3, owst)$.

We improve here the result from [3] from the point of view of all parameters (number of membranes, size of promoters, size of moved multisets) for languages over arbitrary alphabets. However, we consider first the simpler case of one-letter alphabets, for which we obtain a result not contained in [9] (and also looking for the power of the initial mode of introducing the string in the system).

Theorem 2. $1RE = 1ALP_2(p_2 sym_2, owts) = 1A_I LP_2(p_2 sym_2, owts)$.

Proof. We use Lemma 1. Let $L \subseteq \{a_{n+1}\}^*$ be a language whose length set is recognized by a register machine $M = (n, R, q_s, q_h)$. We construct the register machine $M' = (n+1, R', \bar{q}_s, q_h)$ with $lab(M') = lab(M) \cup \{\bar{q}_s, p_1\}$ and

$$R' = R \cup \{\bar{q}_s : (S(n+1), p_1, q_s),\ p_1 : (A(1), \bar{q}_s, \bar{q}_s)\}.$$

The register machine M' start by moving the contents of its register $n+1$ into register 1 and after that it simulates the machine M.

Starting from M' we construct the P system

$$\begin{aligned}
\Pi &= (V, \{a_{n+1}\}, [_1[_2\ ]_2]_1, b, \lambda, V, R_1, R_2),\\
V &= \{a_j \mid 1 \le j \le n+1\} \cup \{q, q' \mid q \in lab(M')\} \cup \{b, c\},\\
R_1 &= \{(a_{n+1}, in)|_b,\ \ (a_{n+1}\bar{q}_s, in)|_b,\ \ (c, in)|_{a_{n+1}},\ \ (c, in)|_{\bar{q}_s q_h},\ \ (c, in)|_c\}\\
&\quad \cup \{(q_2 a_r, in)|_{q_1},\ \ (q_3 a_r, in)|_{q_1} \mid \text{ for } q_1 : (A(r), q_2, q_3) \in R'\}\\
&\quad \cup \{(q_2, in)|_{q_1 a_r},\ \ (q'_3, in)|_{q_1},\\
&\qquad (c, in)|_{q'_3},\ \ (q_3, in)|_{q_1 q'_3} \mid \text{ for } q_1 : (S(r), q_2, q_3) \in R'\},\\
R_2 &= \{(q, in) \mid q \in lab(R') - (\{\bar{q}_s\} \cup \{q_1 \mid \text{ for } q_1 : (S(r), q_2, q_3) \in R'\})\}\\
&\quad \cup \{(\bar{q}_s b, in)\}\\
&\quad \cup \{(q_1 a_r, in),\ \ (q_1 q'_3, in) \mid \text{for } q_1 : (S(r), q_2, q_3) \in R'\}.
\end{aligned}$$

We start with the object b in region 1. As long as b is here, in each step one symbol a_{n+1} is introduced into the system by the rule $(a_{n+1}, in)|_b$. At any moment we can also use the rule $(a_{n+1}\bar{q}_s, in)|_b$. It introduces the label $\bar{q}_s$ in the system, hence at the next step $\bar{q}_s b$ will enter membrane 2, and simultaneously from the set R_1 we can either use again one of the rules $(a_{n+1}, in)|_b$, $(a_{n+1}\bar{q}_s, in)|_b$, or we can start simulating a computation in M.

If we use the rule $(a_{n+1}, in)|_b$, then, because $\bar{q}_s$ is "lost" in region 2, the only rule which can be used in region 1 is $(c, in)|_{a_{n+1}}$ and then the computation continues forever by means of the rule $(c, in)|_c$. If we use the rule $(a_{n+1}\bar{q}_s, in)|_b$, then the new copy of $\bar{q}_s$ will remain forever in region 1 of the system (note that we do not have a rule of the form $(\bar{q}_s, in)$ in R_2, while the unique copy of b was already introduced in membrane 2). Because $\bar{q}_s$ is present, the system will start to simulate the work of M' (see the details below). If the computation in M never stops, then the computation in Π never stops and the string is not recognized.

If the computation in M' stops, hence the label q_h will be introduced, then the computation in Π again continues forever: in the presence of $\bar{q}_s q_h$, we can bring c into the system, and then we can use forever the rule $(c, in)|_c$ from R_1.

Therefore, in order to have a halting computation, after introducing the first $\bar{q}_s$ into the system, we have to start simulating a computation in M'; as we have noted above (see the passing from M to M'), this means the simulation of a computation in M, after moving the contents of register $n+1$ into register 1. Therefore, the contents of register $n+1$ is only once emptied and after that we only work on registers $1, \ldots, n$.

Thus, any string $a_{n+1}^m, m \geq 1$, can be introduced into the system, with $\bar{q}_s$ also present in region 1, and no copy of a_{n+1} will ever be taken from the environment. Hence, the arbitrary and the initial modes of working are identical for Π.

The simulation of the instructions of M' is done as follows. After bringing $\bar{q}_s$ into the system, this object goes into region 2 together with b, at the same time promoting a rule from R_1. In general, let us assume that we have a label q_1 in region 1.

In the case of an add-instruction $q_1 : (A(r), q_2, q_3)$, the simulation is simply done by bringing inside one of the labels q_2, q_3, together with a copy of a_r, by means of rules $(q_2 a_r, in)|_{q_1}$ and $(q_3 a_r, in)|_{q_1}$ of R_1.

Assume that in region 1 we have the label q_1 associated with the subtract-instruction $q_1 : (S(r), q_2, q_3)$. The rule $(q_1 a_r, in) \in R_2$ can be used or not, depending on the existence of at least one copy of a_r, the same with the rule $(q_2, in)|_{q_1 a_r} \in R_1$, while $(q_3', in)|_{q_1} \in R_1$ can be always used. Thus, we distinguish two cases:

Case (i): At least one copy of a_r is present in region 1. Then q_1 is introduced in region 2 together with a copy of a_r, and from R_1 we use either one of the rules $(q_2, in)|_{q_1 a_r}, (q_3', in)|_{q_1}$. If the first one is used, this is the correct simulation of the instruction $q_1 : (S(r), q_2, q_3)$ in the case when the subtraction is possible. If we use the second rule from R_1, then the object q_3' enters region 1 and remains here. At the next step, the only applicable rule from R_1 is $(c, in)|_{q_3'}$, and the computation will continue forever. This is again consistent with the work of M', where the continuation with q_3 is not correct if the subtraction is possible.

Case (ii): If no copy of a_r is present in region 1, then q_1 remains in region 1, and the rule $(q_2, in)|_{q_1 a_r}$ cannot be used. Thus, we have to use the rule $(q_3', in)|_{q_1}$. Because both q_1 and q_3' are present, we may use the rule $(q_3, in)|_{q_1 q_3'} \in R_1$, which brings inside the label q_3; at the same time, q_1 and q_3' enter region 2. We have correctly simulated the instruction $q_1 : (S(r), q_2, q_3)$ in the case when the subtraction is not possible. Instead of the rule $(q_3, in)|_{q_1 q_3'} \in R_1$, we can also use the rule $(c, in)|_{q_3'}$, but this will lead to a non-halting computation.

Therefore, the computation continues without introducing the trap object c only if the instructions of M' are correctly simulated. Clearly, we stop only if the number m is recognized by M' (hence by M), and a_{n+1}^m is thus accepted by Π. Consequently, $A_I(\Pi) = A(\Pi) = L$, and the proof is complete. □

In the case of the arbitrary mode, this result can be extended to arbitrary alphabets.

Theorem 3. $RE = ALP_2(p_2sym_2, owts)$.

Proof. This time we use Lemma 2. Consider a language $L \subseteq V^*$, for some $V = \{b_1, \ldots, b_k\}$, which is accepted by a register machine $M = (n, R, q_s, q_h)$.

We will construct a P system Π, of degree 2, which will work as follows. Starting in the initial configuration, a symbol b_j is introduced in region 1. Then j copies of a_{n+1} are also introduced. Assume that we also have some number α of copies of a_{n+2} present in the system. We multiply this number α with $k+1$, passing from the α copies of a_{n+2} to $k\alpha$ copies of a_{n+1}. In this way, we have $k\alpha + j$ copies of a_{n+1}, which corresponds to the value in base $k+1$ of the string we introduce in the system. After completing this step, we change all objects a_{n+1} with objects a_{n+2} and we continue. At any step, we can also switch to simulating the work of M over the (value in base $k+1$ of the) string already introduced into the system. To this aim, we first change all objects a_{n+2} with objects a_1 and introduce the label q_s in the system.

The introduction of b_j and of j copies of a_{n+1} is done by rules as follows:

Assume that in the skin region we have the object $q_{s,any}$. In R_1 we introduce

$(b_j q_{s,j}, in)|_{q_{s,any}}$,
$(q_{s,j,1} a_{n+1}, in)|_{q_{s,j}}$,
$(q_{s,j,t+1} a_{n+1}, in)|_{q_{s,j,t}}$, for all $1 \leq t \leq j-1$,

and in R_2 we introduce

$(q_{s,any}, in)$,
$(q_{s,j,t}, in)$, for all $1 \leq t \leq j$,

for all $j = 1, 2, \ldots, k$.

The "state-objects" q (with subscripts) are immediately sent into region 2, hence they correctly control the process and then "disappear". When $q_{s,j,j}$ is introduced, we pass to the above described multiplication by $k+1$ of the available copies of a_{n+2}. This can be done by a register machine $M_j = (2, R_j, q_{s,j,j}, q_{s,any})$, with the following instructions:

$$
\begin{aligned}
&q_{s,j,j} : (S(n+2), q_{j,1}, q'_j),\\
&q_{j,t} : (A(n+1), q_{j,t+1}, q_{j,t+1}), \text{ for all } 1 \leq t \leq k,\\
&q_{j,k+1} : (A(n+1), q_{s,j,j}, q_{s,j,j}),\\
&q'_j : (S(n+1), q''_j, q_{s,any}),\\
&q''_j : (A(n+2), q'_j, q'_j).
\end{aligned}
$$

The two registers of M_j were numbered with $n+1, n+2$ because M_j will be integrated in a large machine, with $n+2$ registers.

The instruction $q'_j : (S(n+1), q''_j, q_{s,any})$ can be also replaced with

$$q'_j : (S(n+1), q''_j, q_{move})$$

(let us denote by M'_j the obtained machine), and q_{move} will trigger the register machine M_{move} which changes all symbols a_{n+2} with a_1, and ends in the label q_s (which is the initial label of M).

Now, consider the procedure of passing from the add- and subtract-instructions of a register machine to conditional symport rules as used in the proof of Theorem 2; the obtained rules are placed in two sets, R_1 and R_2. We apply this procedure for all instructions of the register machines $M_j, M'_j, 1 \leq j \leq k$, and M_{move}, as specified above. We obtain a "subsystem" of Π which introduces a string $w \in \{b_1, \ldots, b_k\}^*$ into the skin membrane, and at the same time produces here $val_{k+1}(w)$ copies of a_1; in the end of this process, the object q_s is also placed in the skin region. Therefore, the work of M can now be simulated as in the proof of Theorem 2. In total, we get a system Π which accepts exactly the strings which are accepted by M, hence $A(\Pi) = L$. □

We do not know whether or not the result in Theorem 2 can be extended to languages over arbitrary alphabets also in the case of the initial mode.

5 The Universality of P Automata with States

Consider now the variant of P automata introduced in [10], in a simplified version. Specifically, we deal with systems of the form $\Pi = (V, K, T, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m)$, where all components are as usual, $K \subseteq V - T$ is the set of *states*, and the rules from sets $R_i, 1 \leq i \leq m$, are of the form $(qy, in)|_{px}$, where $p, q \in K$ and $x, y \in V^*$. The rules are used in the maximally parallel manner. Using a rule $(qy, in)|_{px} \in R_i$ is possible only if px is included in the multiset present in region i, and the effect is that the multiset y is introduced in region i from the surrounding region, while p is replaced by q. Thus, in each region we can use at most as many rules as occurrences of state-objects exists in that region. As usual, a string over T is accepted if it is the sequence of symbols from T taken into the system during a halting computation. The language of all strings accepted by Π is denoted by $A(\Pi)$ and the language of strings accepted in the initial mode is denoted by $A_I(\Pi)$. The family of all languages $A(\Pi)$, accepted by systems Π as above with at most m membranes, using rules $(qy, in)|_{px}$ with $|qy| \leq r, |px| \leq u$, and with at most t states is denoted by $ALP_m(p_u sym_r, state_t)$. If the strings are recognized in the initial mode, then we write $A_I LP$ instead of ALP.

Clearly, the automata from [10] are extensions of those from [3]: we can consider a unique state, p, and instead of a rule $(y, in)|_x$ we consider the rule $(py, in)|_{px}$. Thus, the following results are consequences of Theorems 2, 3. Direct proofs can be also given, much simpler than the proofs of Theorems 2, 3, but we leave this as a task for the reader.

Theorem 4. $1RE = 1ALP_2(p_3 sym_3, state_1) = 1A_I LP_2(p_3 sym_3, state_1)$.

Theorem 5. $RE = ALP_2(p_3 sym_3, state_1)$.

The optimality of these results, as well as whether Theorem 5 holds also for the initial mode, remains as an open problem.

6 Accepting P Systems with Symport/Antiport

Consider now systems as those from [5]. In the arbitrary case they can recognize all recursively enumerable languages even when using only one membrane. More precisely, the following result was proven in [5]:

Theorem 6. *Each language $L \in RE$ can be recognized by an analysing P system with only one membrane using antiport rules $(x, out; y, in)$ with $(|x|, |y|) \in \{(1,2),(2,1)\}$ only (and no symport rule).*

Thus, there is nothing else to do about the general case; what remains to do is to investigate the initial mode of accepting strings. We will return to this topic in a forthcoming paper, here we only mention two preliminary results.

Theorem 7. *Each regular language can be recognized by an analysing P system in the initial mode with only one membrane using antiport rules $(x, out; y, in)$ with $(|x|, |y|) \in \{(1,2),(1,1)\}$ only (and no symport rule). The converse assertion is not true: there are systems as above which recognize non-regular languages.*

Proof. Let $M = (K, V, q_0, F, \delta)$ (set of states, alphabet, initial state, set of final states, transition mapping) be a deterministic finite automaton, and construct the P system

$$\begin{aligned} \Pi \ &(\ K \cup V, V, [_1 \ ]_1, q_0, K, R_1), \\ R_1 &= \{(q, out; q'a, in) \mid q, q' \in K, a \in V, \delta(q, a) = q'\} \\ &\cup \{(q, out; q, in) \mid q \in K - F\}. \end{aligned}$$

Each antiport rule $(q, out; q'a, in)$ simulates a transition in M defined by $\delta(q, a) = q'$ (bringing into the system the corresponding symbol a). We start with q_0 in the system. If we reach the end of the input string in a final state, then the computation halts, otherwise the rule $(q, out; q, in)$ can be used forever. Therefore, $L(M) = A_I(\Pi)$ and $REG \subseteq A_I LP_1(sym_0, anti_2)$.

This converse is not true. Indeed, consider the system

$$\begin{aligned} \Pi &= (\{a, b, d, e\}, \{a, b\}, [_1 \ ]_1, dd, \{e\}, R_1), \\ R_1 &= \{(d, out; ae, in), \ (e, out; de, in), \ (e, out; b, in), \ (e, out; e, in)\}. \end{aligned}$$

The reader is asked to check that $A_I(\Pi) \cap a^+b^+ = \{a^n b^n \mid n \geq 1\}$, which is not a regular language. □

As a consequence of Theorem 1 we get:

Theorem 8. $1RE = 1A_I LP_n(sym_r, anti_t)$ *for* $(n, r, t) \in \{(3,2,2), \ (5,2,0), \ (4,3,0)\}$.

The proof is based on the following lemma (the systems used in the proofs of the results in [9] and [6] have the properties from this lemma).

Lemma 3. *Let $Q \in NOP_m(sym_r, anti_t)$ be a set of numbers generated by a P system of the type associated with the family $NOP_m(sym_r, anti_t)$, which counts in the halting configuration the symbols from a set T present in an elementary membrane i_o and such that the only rules associated with membrane i_o and involving elements of T are of the form $(a, in), a \in T$. Then $\{b^j \mid j \in Q\} \in 1A_I LP_{m+2}(sym_{r'}, anti_t)$, where $r' = \max(r, 2)$.*

7 Conclusions

We have investigated analysing P systems with symport/antiport rules of the forms considered in [3], [10], and [5], recognizing strings in the modes proposed in these papers, or with a particular mode of introducing the strings into the system, which we call *initial*. The results from [3], [10] were significantly improved: all recursively enumerable languages can be recognized in the arbitrary mode by systems with only two membranes, while the same result holds true also in the initial mode for languages over the one-letter alphabet. For systems as in [5], the initial mode remains to be investigated.

References

1. F. Bernardini, V. Manca, P systems with boundary rules, in [14].
2. E. Csuhaj-Varju, C. Martin-Vide, V. Mitrana, Multiset automata, *Multiset Processing* (C.S. Calude, Gh. Păun, G. Rozenberg, A. Salomaa, eds), LNCS **2235**, Springer, Berlin, 2001, 69–84.
3. E. Csuhaj-Varju, G. Vaszil, P automata, in [14].
4. R. Freund, M. Oswald, P systems with activated/prohibited membrane channels, in [14].
5. R. Freund, M.Oswald, A short note on analysing P systems, *Bulletin of the EATCS*, 78 (October 2002), 231–236.
6. R. Freund, A. Păun, Membrane systems with symport/antiport: universality results, in [14].
7. R. Freund, Gh. Păun, On the number of non-terminal symbols in graph-controlled, programmed and matrix grammars, *Proc. Conf. Universal Machines and Computations,* Chişinău, 2001 (M. Margenstern, Y. Rogozhin, eds.), *LNCS* 2055, Springer. Berlin, 2001, 214–225.
8. R. Freund, P. Sosik, P systems versus register machines: two universality proofs, in [14].
9. P. Frisco, H.J. Hoogeboom, Simulating counter automata by P systems with symport/antiport, in [14].
10. M. Madhu, On a class of P automata, manuscript, 2002.
11. M.L. Minsky, *Computation: Finite and Infinite Machines*, Prentice Hall, Englewood Cliffs, New Jersey, USA, 1967.
12. A. Păun and Gh. Păun, The power of communication: P systems with symport/antiport, *New Generation Computing*, **20,** 3 (2002), 295–306.
13. Gh. Păun, *Computing with Membranes: An Introduction*, Springer, Berlin, 2002.
14. Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, eds., *Membrane Computing 2002, Lecture Notes in Computer Science*, Springer, Berlin, 2002.