P automata revisited

Gheorghe Păun [a,b], Mario J. Pérez-Jiménez [b]

[a] *Institute of Mathematics of the Romanian Academy, PO Box 1-764, 014700 București, Romania*
[b] *Department of Computer Science and Artificial Intelligence, University of Sevilla, Avda. Reina Mercedes s/n, 41012 Sevilla, Spain*

Keywords:

Membrane computing
P automata
Turing computability
Closure properties

A B S T R A C T

We continue here the investigation of P automata, in their non-extended case, a class of devices which characterize non-universal family of languages. First, a recent conjecture is confirmed: any recursively enumerable language is obtained from a language recognized by a P automaton, to which an initial (arbitrarily large) string is added. Then, we discuss possibilities of extending P automata, following suggestions from string finite automata. For instance, automata with a memory (corresponding to push-down automata) are considered and their power is briefly investigated, as well as some closure properties of the family of languages recognized by P automata. In the context, a brief survey of results about P and dP automata (a distributed version of P automata) is provided, and several further research topics are formulated.

1. Introduction

P automata are usual symport/antiport P systems used in the accepting mode: the sequence of objects which are input from the environment during a halting computation is said to be accepted by the automaton and the set of all such strings forms the language recognized by the automaton. There is an important point here: at the same moment, several objects can be "read" from the environment. In the simplest case, considered in [9] and in a series of subsequent papers, see the bibliography, when several objects are taken at the same time, then all their permutations are introduced in the accepted string. A more general possibility is to proceed as in [5], and consider a function which associates a symbol (or a string) with any multiset, and then the string accepted during a computation is the concatenation of the images of the considered mapping for the sequence of the input multisets. Various functions can give various different results—but here we work in the style of [9] and we leave as a research topic the more general approach of [5].

In the extended case (a terminal alphabet is considered and any object which is not from this alphabet is ignored; thus, no relation there is between the length of the accepted string and the number of objects taken from the environment), P automata characterize the family of recursively enumerable languages. In the non-extended case, because the working space is limited to the number of input symbols, to which one adds the objects initially present in the system, we only recognize context-sensitive languages. This is a very important point, as most of the classes of P systems are computationally universal, and many times in membrane computing the interest was shown for classes of P systems which are not universal. In spite of this interest, not so many papers were devoted so far to non-extended P automata; some results can be found in [8], [14], in general, in the context of studying the recently introduced dP automata, [13]. For the reader convenience, we also introduce here this notion, of a dP automaton.

The present paper aims to contribute to filling this gap, on the one hand, investigating basic properties of P automata and of their languages, such as closure properties, and on the other hand considering extensions of P automata, on the model

of the extensions of string finite automata. Automata with a memory (reminding the push-down automata) and two-way automata are considered.

Many other questions remain to be investigated; we formulate a series of open problems and research topics, but the reader can imagine many other, just following this general idea: extend to P automata ideas investigated in the theory of classic string finite automata.

2. P and dP automata

The reader is assumed to be familiar with elementary facts about membrane computing, e.g., from [12], [16], and of formal language theory, e.g., from [17], [18]. (In what follows, V^* is the free monoid generated by the alphabet V, λ is the empty word, $V^+ = V^* - \{\lambda\}$, and $|x|$ denotes the length of the string $x \in V^*$. *REG*, *LIN*, *CF*, *CS*, *RE* denote the families of regular, linear, context-free, context-sensitive, and recursively enumerable languages, respectively. As usual in membrane computing, the multisets over an alphabet V are represented by strings in V^*.)

For the reader convenience, we start by introducing the more general notion of a *dP automaton*, which is a construct

$$\Delta = (O, E, \Pi_1, \ldots, \Pi_n, R),$$

where (1) O is an alphabet (of objects), (2) $E \subseteq O$ (the objects available in arbitrarily many copies in the environment), (3) $\Pi_i = (O, \mu_i, w_{i,1}, \ldots, w_{i,k_i}, E, R_{i,1}, \ldots, R_{i,k_i})$ is a symport/antiport P system of degree k_i (O is the alphabet of objects, μ_i is a membrane structure of degree k_i, $w_{i,1}, \ldots, w_{i,k_i}$ are the multisets of objects present in the membranes of μ_i in the beginning of the computation, E is the alphabet of objects present—in arbitrarily many copies—in the environment, and $R_{i,1}, \ldots, R_{i,k_i}$ are finite sets of symport/antiport rules associated with the membranes of μ_i; the symport rules are of the form (u, in), (u, out), where $u \in O^*$, and the antiport rules are of the form $(u, out; v, in)$, where $u, v \in O^*$; note that we do not have an output membrane), with the skin membrane labeled with $(i, 1) = s_i$, for all $i = 1, 2, \ldots, n$, and (4) R is a finite set of rules of the form $(s_i, u/v, s_j)$, where $1 \leq i, j \leq n$, $i \neq j$, and $u, v \in O^*$, $uv \neq \lambda$.

The systems $\Pi_1, \ldots, \Pi_n$ are called *components* of Δ and the rules in R are called *communication rules*. For a rule $(u, out; v, in)$ or $(s_i, u/v, s_j)$, $|uv|$ is the *weight* of this rule.

Each component can take an input, work on it, communicate with other components (by means of rules in R), and provide the answer to the problem in the end of a halting computation. Note that the environment is common, hence the components can also communicate, in two steps, through the environment.

A halting computation with respect to Δ accepts the string $x = x_1 x_2 \ldots x_n$ over O if the components $\Pi_1, \ldots, \Pi_n$, starting from their initial configurations, using the symport/antiport rules as well as the inter-components communication rules, in the non-deterministically maximally parallel way, bring from the environment the substrings $x_1, \ldots, x_n$, respectively, and eventually halts. We denote by $L(\Delta)$ the language recognized in this way by Δ. (Remember that we work here with the assumption that, if several objects enter at the same time the system, then any permutation of them is considered as a substring of the accepted string.)

The dP automaton is synchronized, a universal clock exists for all components, marking the time in the same way for the whole dP automaton.

Let us denote by *LdP* the family of all languages recognized by dP automata. A dP automaton of degree 1 (hence with only one component, with no communication rules) is a usual P automaton. We denote by *LP* the family of languages recognized by P automata.

Clearly, a P automaton (with k membranes) is written in the form $\Pi = (O, \mu, w_1, \ldots, w_k, E, R_1, \ldots, R_k)$.

Note that we ignore here the number of components of dP automata, of membranes in P automata, as well as other descriptive measures, such as the weight of rules. If necessary, such parameters can be easily taken into consideration (and many problems appear about them: obtaining results for small systems, finding hierarchies according to such criteria, and so on).

3. A short survey of results

We recall now some of the results reported in the literature of P and dP automata.

We mentioned that when a terminal alphabet T is considered, as a subset of O, and the symbols from $O - T$ are ignored when building the accepted string, then for the obtained family of languages, denoted by *ELP*, we have:

Theorem 3.1. $RE = ELP$.

In turn, for non-extended P automata, the following inclusion was suggested above:

Lemma 3.1. $LP \subseteq CS$.

P automata can recognize non-context-free languages, hence [8]:

Lemma 3.2. $LP - CF \neq \emptyset$.

An example of a non-regular language in *LP* is $L_1 = \{(a^2c)^s(b^2d)^s \mid s \geq 1\}$—and it can be easily extended to non-context-free languages (the equality of three blocks of the form $(\alpha^2\beta)^s$ can be checked in terms of P automata).

Two necessary conditions for a language to be in *LP* were given in [8]:

Lemma 3.3. *For every language $L \subseteq V^*$, $L \in LP$, which is not regular there is a string $w \in L$ which can be written in the form $w = w_1abw_2$, for some $w_1, w_2 \in V^*$ and $a, b \in V$ (not necessarily distinct) such that $w_1baw_2 \in L$.*

This lemma implies, for instance, that the linear language

$$L_2 = \{(ab)^n(ac)^n \mid n \geq 1\}$$

is not in *LP*. Actually, a more general consequence of Lemma 3.3 is drawn in [8]:

Theorem 3.2. *All families of languages which include strictly the family of regular languages and are closed under λ-free morphisms contain languages which are not in LP.*

Another necessary condition for a language to be in *LP* given in [8] is:

Lemma 3.4. *Let V be an alphabet with at least two elements and $f : V^* \longrightarrow V^*$ an injective mapping. The language $L_f = \{wf(w) \mid w \in V^*\}$ is not in the family LP.*

Because this result can be extended to other types of P automata (e.g., to P automata with an internal memory), we recall the idea of the proof: the number of configurations of a P automaton which has brought inside m symbols is bounded by a polynomial in m, but there are more than 2^m different strings of length m over an alphabet with more than two symbols (hence exponentially many); this makes impossible the matching between the two halves of the strings.

As a consequence of the previous lemma, for instance, the context-sensitive language, $L_3 = \{wf(w) \mid w \in \{a, b\}^*\}$ for $f(a) = a', f(b) = b'$, is not in *LP*.

Based on the idea of the previous proof, in [14] it is also shown that the language $L_4 = \{(ww')^s \mid w \in \{a, b\}^+, s \geq 2\}$, where w' is obtained from w by priming the symbols a and b, is not in the family *LdP*.

As expected, P automata can recognize all regular languages ([8]; a simpler proof is given in [15]):

Theorem 3.3. *$REG \subset LP$.*

Also natural is the fact that the distribution increases the power, dP automata are strictly more powerful than P automata:

Theorem 3.4. *$LP \subset LdP \subset CS$.*

Synthesizing these results, we obtain the diagram in Fig. 1, based on a similar diagram from [8]; the languages L_1, L_2, L_3, L_4 are specified above and L_5 is only conjectured: $L_5 = \{x\, mi(x) \mid x \in \{a, b\}^*\}$, where $mi(x)$ is the mirror image of x.

4. A representation theorem for RE languages

We pass now to providing some new results about P automata.

The following theorem is classic in formal language theory—see, e.g., [18]:

Theorem 4.1. *For every language $L \in RE, L \subseteq V^*$, there is a language $L' \in CS$ and two symbols $a, c \notin V$ such that:* (i) *$L' \subseteq L\{c\}a^*$,* (ii) *for each $w \in L$ there is $i \geq 0$ such that $wca^i \in L'$.*

Otherwise stated, the two languages are "the same" up to a tail of arbitrary length added to strings in L.

The following counterpart of Theorem 4.1 was proved in [8]:

Theorem 4.2. *For every language $L \in RE, L \subseteq V^*$, there is a language $L' \in LdP$ and an alphabet U disjoint of V such that:* (i) *$L' \subseteq LU^*$,* (ii) *for each $w \in L$ there is $y \in U^*$ such that $wy \in L'$.*

Moreover, it is conjectured in [8] that a similar result is valid also for languages recognized by P automata, but this time with the "tail" placed in the left hand of the string. We confirm here this conjecture. The result is non-trivial, because *LP* (the same for *LdP*) is strictly included in *CS*.

Theorem 4.3. *For every language $L \in RE, L \subseteq V^*$, there is a language $L' \in LP$, and an alphabet U disjoint of V such that:* (i) *$L' \subseteq U^*L$,* (ii) *for each $w \in L$ there is $y \in U^*$ such that $yw \in L'$.*

Proof. We follow the idea of the proof of Theorem 4.2 from [8], implemented for P automata, but we only give here part of the technical details of the construction.

Consider a language $L \in RE$ over an alphabet $V = \{a_1, a_2, \ldots, a_n\}$ for some $n \geq 1$. Consider the strings in V^+ as numbers in basis $n + 1$, hence written with the digits $a_i = i$, $1 \leq i \leq n$; we omit the digit zero. Let $val_{n+1}(w)$ be the integer which represents the value of "number" $w \in V^*$ in this sense. Clearly, $val_{n+1}(xa_i) = (n + 1)val_{n+1}(x) + i$, for any $x \in V^*$ and $a_i \in V$, $1 \leq i \leq n$. This computation can be done by a register machine $M_i = (H_i, l_{0,i}, l_{h,i}, I_i)$ (set of labels, initial label, halting label, set of instructions) which starts from instruction with label $l_{0,i}$ with $val_{n+1}(x)$ stored in register 1 and ends with the number $val_{n+1}(xa_i)$ in the same register, after reaching the instruction $l_{h,i}$: `halt`.

We extend the mapping val_{n+1} in the natural way to languages, $val_{n+1}(L) = \{val_{n+1}(w) \mid w \in L\}$, for $L \subseteq V^*$. Obviously, $L \in RE$ if and only if the set $val_{n+1}(L)$ is a recursively enumerable one, hence if and only if there is a register machine

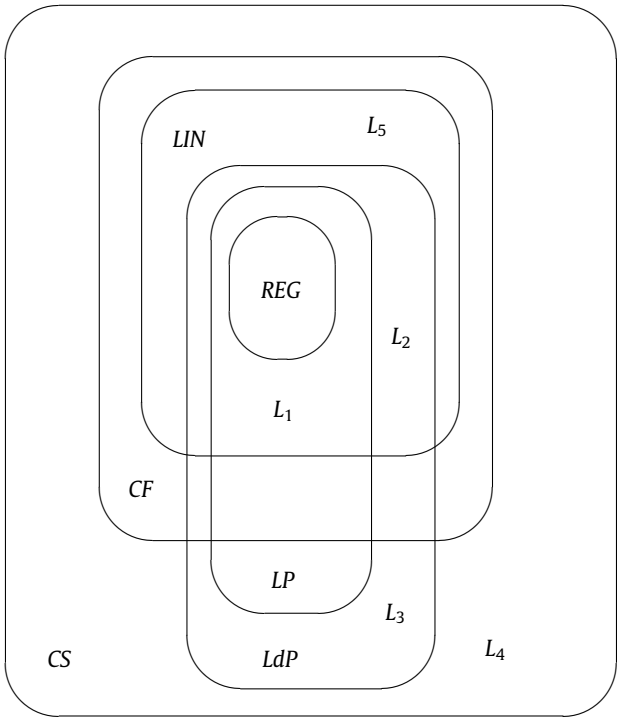

Fig. 1. The place of the families *LP* and *LdP* in Chomsky hierarchy.

$M = (H, l_0, l_h, I)$ which accepts exactly the numbers in $val_{n+1}(L)$ (starting from the initial label l_0, with a number m in its first register, the machine M halts if and only if $m \in val_{n+1}(L)$; halting means reaching the instruction l_h : `halt`).

The general strategy we follow is the following.

We construct a P automaton Π as schematically indicated in Fig. 2, where we have specified only part of the rules associated with membranes; instead, inside membrane 1 we have indicated the two "macro-steps" of the computation done here. These macro-steps are the following: (1) computing $val_{n+1}(xa_i)$, by means of a register machine $M_i = (H_i, l_{0,i}, l_{h,i}, I_i)$, $1 \leq i \leq n$, and (2) checking whether $val_{n+1}(w) \in N(M)$ for a register machine M as above.

Let us denote

$$V' = \{a'_i \mid a_i \in V\},$$
$$V'' = \{l, l', l'', l''', l^{iv} \mid l \in H \cup \cup_{i=1}^{n} H_i\},$$
$$V''_h = \{l_h\} \cup \{l_{h,i} \mid 1 \leq i \leq n\},$$
$$B = \{b_1, b_2, b_3\}.$$

The computation starts by introducing in the skin region arbitrarily many copies of objects from $V'' \cup B$, as well as copies of objects in V', these last ones paired with copies of f. This is done in the presence of object d, present both in the skin region and in the environment (the environment is supposed to contain all objects we need, in sufficiently many copies). Immediately after entering the system, the pairs fa'_i should be moved to membrane 1 (otherwise a'_i will release the trap object # from membrane 2 and the computation never halts, # will oscillate forever across membrane 1).

In this way, we introduce the prefix y from the theorem statement, whose objects are used as a workspace for the computations done in membrane 1 (these computations need an environment, but it is not possible to use the system environment, because the objects we input are considered in the input string, that is why we use the skin region as the environment of membrane 1).

At any moment, a rule $(d, out; a_i, in)$ can be used and we pass now to the second stage of the computation. After introducing any symbol of V inside the system, the numerical value of the string read so far is calculated in membrane 1 and expressed as the number of occurrences of the symbol b_1.

To this aim, a_i enters membrane 1 together with $l_{0,i}$; when the computation is completed, hence $l_{h,i}$ is produced, the triple $a_i a'_i l_{h,i}$ exits membrane 1. The pair $a_i a'_i$ should exit the system, bringing inside another element of V: if a_i enters again membrane 1, a'_i can only release the trap object from membrane 2 (there is no copy of f present in the skin membrane, to help a'_i to enter membrane 1). If the computation of the register machine is not completed because we do not have enough

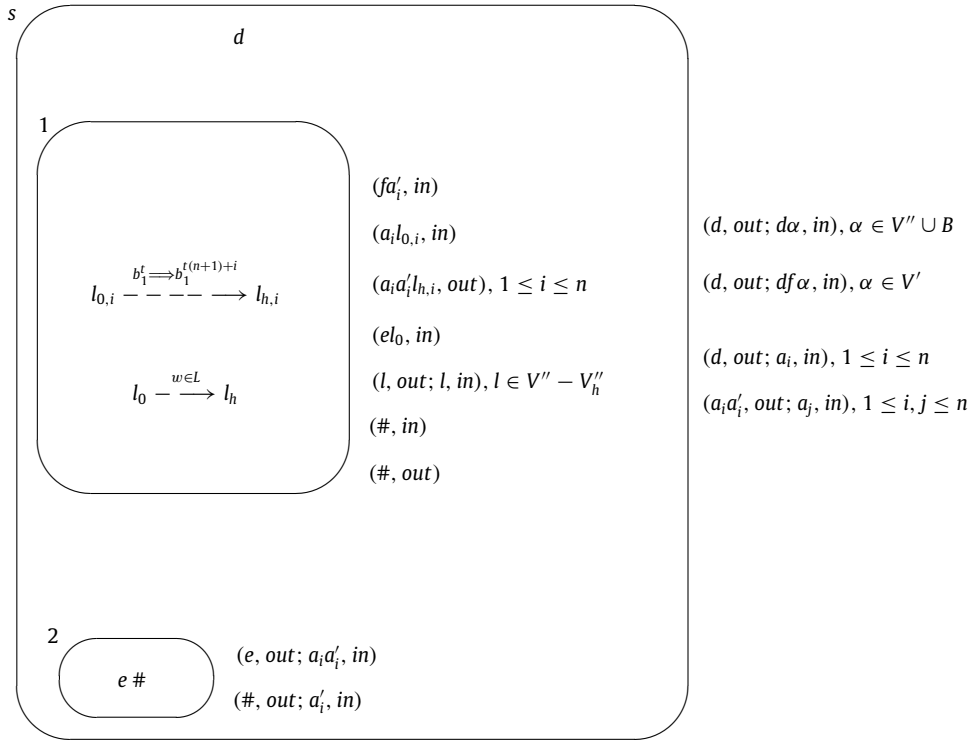

Fig. 2. The P automaton from the proof of Theorem 4.3.

working symbols in the skin region, then the computation never halts, cycling rules of the form $(l, out; l, in)$ are provided for all labels which are not halting labels.

At some moment, non-deterministically, we stop introducing objects from the environment and pass to checking whether the string read up to now belongs to the language L. The pair $a_i a_i'$ enters membrane 2, object e is released, and it enters membrane 1 together with object l_0. This means activating the simulation of the register machine M. If $w \in L$, then the label l_h is reached, if not, the computation continues forever.

Now, what remains to be done is to implement the register machines M_i, $1 \leq i \leq n$, and M, and this can be done exactly as in [8], making use of the objects from $V'' \cup B$ from the skin region. We omit the details, we just mention that, without loss of the generality, we assume that all register machines we use have at most three registers, and that in the end of the computations all registers are empty except the first one.

With the notation $U = V' \cup V'' \cup B \cup \{f\}$, the proof is complete. □

This theorem has a series of consequences:

Corrolary 4.1. (i) *The family LP is not closed under arbitrary morphisms and under left quotients with regular languages.* (ii) *If a family FL of languages is closed under arbitrary morphisms or under left quotients by regular languages, and* $FL \subset RE$, *then* $LP - FL \neq \emptyset$. (iii) *LP is incomparable with all families of languages FL such that* $REG \subset FL \subset RE$ *and FL is closed under arbitrary morphisms.*

Proof. Point (i) is a direct consequence of the previous theorem and of the fact that $LP \subset CS \subset RE$. If $LP \subseteq FL \subset RE$ and FL is closed under the mentioned operations, then, from the theorem above we get $FL = RE$, which is a contradiction, hence point (ii) follows. Finally, combining point (ii) with Theorem 3.2, we obtain the assertion (iii). □

5. A non-closure property

We were not able to find too much about the closure properties of the family LP, in particular, we found no positive closure property. We *conjecture* that LP is neither an AFL nor an anti-AFL. We only give here the following further negative result:

Theorem 5.1. *The family LP is not closed under inverse morphisms.*

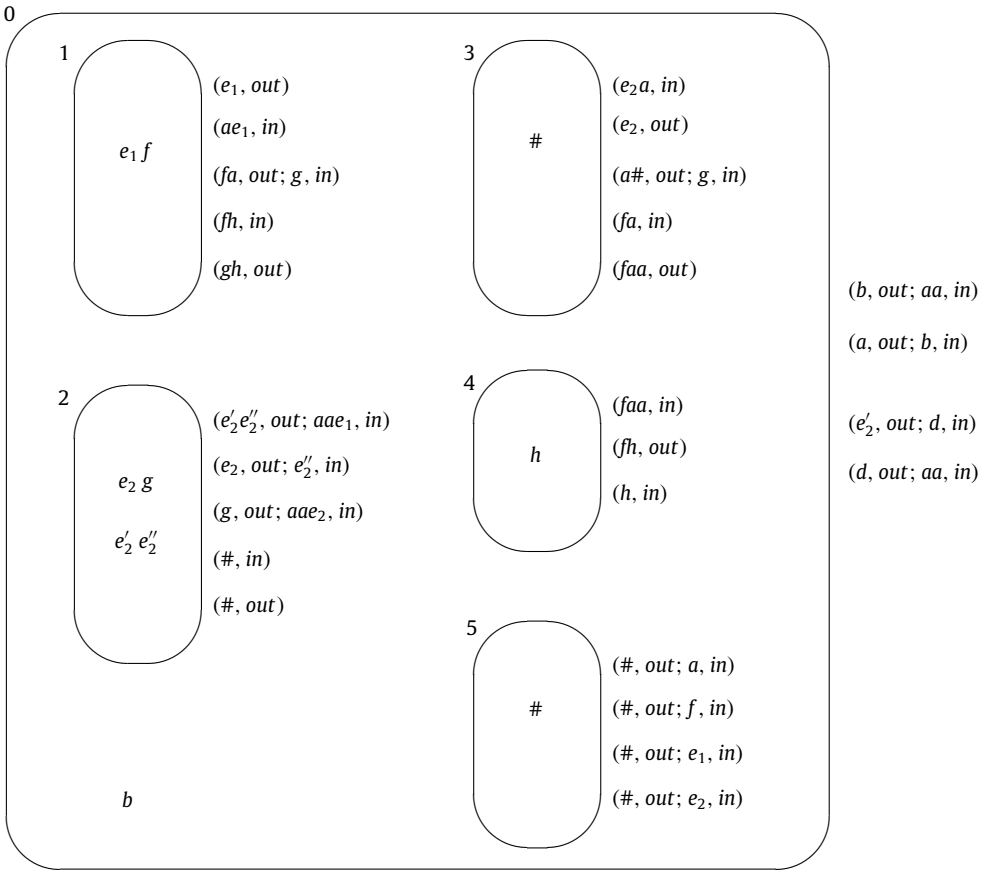

Fig. 3. The P automaton from the proof of Theorem 5.1.

Proof. Let us consider the following two non-regular languages:

$$L = \{(ab)^m ad(ab)^n a \mid m \geq n \geq 0\},$$
$$L' = \{(aab)^m aad(aab)^n a \mid m \geq n \geq 0\}.$$

The first language is not in *LP* because it does not have the property in Lemma 3.3. For the morphism $h : \{a, b, d\}^* \longrightarrow \{a, b, d\}^*$ defined by $h(a) = aa$, $h(b) = b$, $h(d) = d$ we have $L = h^{-1}(L')$. In order to prove the non-closure under inverse morphisms is then enough to show that $L' \in LP$. To this aim, we construct the P automaton Π whose initial configuration is shown in Fig. 3.

The automaton works as follows. In the first step, the rule $(b, out; aa, in)$ brings inside two copies of a, and at the same time object e_1 exist membrane 1. If in the next step both objects a exit in exchange of b, then e_1 will release the trap object # from membrane 5 and the computation never halts (# oscillates forever across membrane 2). Thus, one copy of a should enter membrane 1 together with e_1, while the second copy should go to the environment (another possibility is to get changed with the trap object from membrane 5). This process can be iterated for $m \geq 1$ steps, and then the rule $(e'_2 e''_2, out; aae_1, in)$ can be used. The membrane 1 contains m copies of a, e_1 gets "hidden" in membrane 2, and we pass now to reading objects in the second "half" of the string. First, e'_2, e''_2 are released from membrane 2, e'_2 brings d in the system, while e''_2 is exchanged with e_2 from membrane 2. Now, strings aab are again introduced in the skin membrane, but e_2 moves one a from each pair aa into membrane 3. In this way, $n \geq 1$ copies of a are stored here. This process is stopped by using the rule $(g, out; aae_2, in)$, which releases the object g from membrane 2, to start checking whether or not $m \geq n$.

Object g helps fa to exit membrane 2 and remains there; fa enters membrane 3 and exits with a further a. If this happens once again, then one a remains unused and it releases # from membrane 5, or it can bring one b from the environment, and again the computation never halts, because of the rules $(b, out; aa, in)$, $(a, out; b, in)$, which can be used forever, the number of copies of as present in the skin region will continuously increase. Thus, we have to use immediately the rule (faa, in) from R_4. Object f exits membrane 4 together with h; h cannot return immediately, because f alone can only release the trap object. Therefore, fh enters membrane 1, gh exits, fa is again exchanged for g, now h enters membrane 4, and the process is iterated. One by one, copies of a from membranes 1 and 3 are paired and moved to membrane 4.

If, at some moment, f takes one a from membrane 1 and no a exists in membrane 3, then the computation halts, as f remains blocked in membrane 3. If no copy of a exists in membrane 1, f remains blocked here, gh exits, h returns to

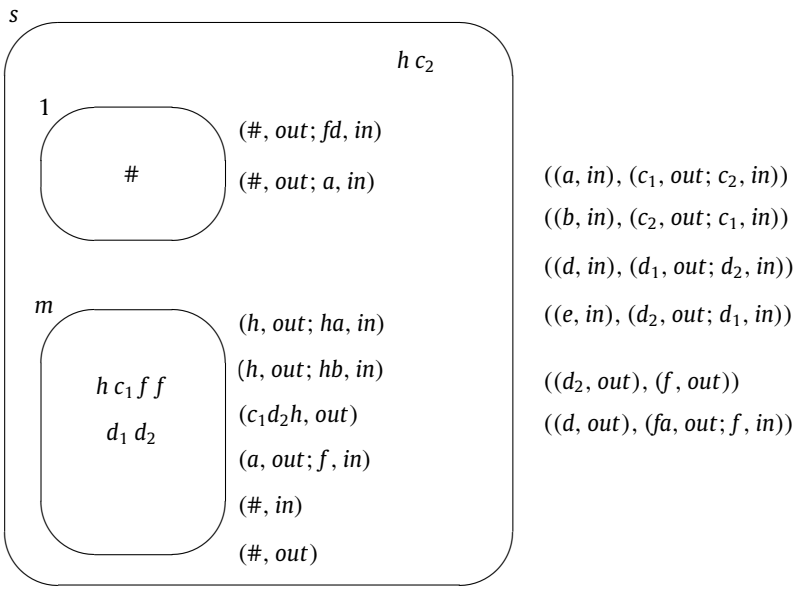

Fig. 4. The P automaton from the proof of Theorem 6.1.

membrane 4, but g checks whether membrane 3 contains any copy of a. In the affirmative case, the rule $(a\#, out; g, in)$ can be used, and the computation never stops. If no a exists here, then the computation halts.

Therefore, the computation halts if and only if the input string is of the form $(aab)^m aad(aab)^n aa$, for some $m \geq n \geq 1$, hence L' is a language in LP. □

6. P automata with an internal memory

Another natural extension of a P automaton, suggested by the difference between a finite automaton and a push-down one (the latter has a memory tape, which controls the movements along the input tape) is to consider an *internal memory* also for P automata, used for controlling the input of multisets from the environment. To have a simple definition, we use here as memory one (and only one) elementary membrane, which has objects inside and rules associated as any usual membrane, but the rules of the skin membrane are paired with rules associated with the memory membrane. More specifically, besides symport or antiport rules associated with the memory membrane m—let us denote their set with R_m, as usual—all rules associated with the skin membrane are pairs of the form (r_1, r_2), where r_1 is a symport/antiport rule which passes objects across the skin membrane and r_2 is a symport/antiport rule which passes objects across the membrane m; there is no relation between such rules r_2 and the set R_m.

The work of such an automaton is as usual. Let us denote by LmP the family of languages recognized by P automata with an internal memory.

Clearly, $LP \subseteq LmP$: to a usual P automaton, a "dummy" memory membrane can be added, with an object f inside and an object f outside, and a rule $(f, out; f, in)$ is coupled with each rule in R_s; the accepted strings remain the same.

As expected, using a memory as above helps, it strictly increases the power of P automata:

Theorem 6.1. $LmP - LP \neq \emptyset$.

Proof. We consider the automaton from Fig. 4, which works as follows.

While c_1 and c_2 oscillate across membrane m, objects a and b are brought into the skin region. Immediately, they are moved to membrane m, assisted by the objects h present both in membrane m and in the skin region. At any moment, when c_2 is outside membrane m, the rule (c_1d_2h, out) from R_m can be used. The reading of objects a, b stops and objects d, e are introduced, with d_1, d_2 playing now the role of c_1, c_2, and oscillating across membrane m. Objects d, e remain in the skin region.

At some moment, when a string $(ab)^n$, $n \geq 0$, was read (and moved into the memory membrane), followed by a string $(de)^m$, $m \geq 0$ (which remains in the skin membrane), we can use the coupled rule $((d_2, out), (f, out))$ of R_s. No further objects can be brought from the environment, we start checking whether $n = m$. The coupled rule $((d, out), (fa, out; f, in))$ is used to this aim: simultaneously, one d is sent to the environment and one a is taken from membrane m. If the multisets of a and d are exhausted at the same time, then the computation stops. If we have more copies of d than of a, then the rule $(\#, out; fd, in)$ will take the trap object # from membrane 1 and the computation never halts, the object # oscillates forever

across membrane m. If this rules is used while still objects a are present, the computation is "lost", because it continues forever.

Similarly, if we finish the objects d and still copies of a are present in membrane m, then the rule $(a, out; f, in)$ should be used, and now the object a brings # outside membrane 1, and the computation never stops. Consequently, the computation halts if and only if $n = m$, therefore $L(\Pi) = \{(ab)^n(de)^n \mid n \geq 0\}$. This language does not have the property from Lemma 3.3, hence it is not in the family LP, that is, $LP \subset LmP$ is a strict inclusion, and this completes the proof. □

It is easy to see that Lemma 3.4 holds true also for the family LmP, with the same proof as for usual P automata, hence languages of the form $\{ww' \mid w \in \{a, b\}^*\}$, with w' being the primed version of w, are not in LmP. Moreover, the workspace of a P automaton with memory is equal with the length of the input string, plus the finite number of objects present in the initial configuration, hence $LmP \subseteq CS$. Thus, the following strict inclusions hold:

Corrolary 6.1. $LP \subset LmP \subset CS$.

7. Further research topics

As pointed out several times above, there are many open problems and topics to be investigated for P automata. We have settled only two closure properties involved in the definition of AFLs: LP is not closed under arbitrary morphisms and under inverse morphisms (the same with the quotient with respect of regular languages). What about other operations? We have said nothing about other properties, such as the decidability of basic questions (except membership, which is decidable because of the inclusion $LP \subset CS$).

Other "classic" issues about automata remains to be considered. For instance, the proof of Theorem 4.3 suggests the idea of a two-way automaton: the prefix appended to the string in the RE language we want to represent is meant to create inside the system a large enough workspace, to be used when introducing the relevant part of the input. If we imagine that the symbols to read are written on a tape, as in the case of usual automata, and we read them by means of symport/antiport rules, then we can imagine that the tape remains always at the disposal of the automaton, hence we can move both to right, as usual, and to left along this tape (just add left-right indications to the symport/antiport rules). Then, we can place in the left hand of the tape one copy of each symbol necessary during the computation; by reading this part of the tape in a repeated way, we input as many copies of these symbols as we need, while reading a fixed amount of the tape. In this way, the representation from Theorem 4.3 can be obtained with a prefix of a bounded length. As a consequence, the non-closure of the family of languages recognized by two-way P automata under left derivatives is obtained. The details are left to the reader—starting with the formalization of two-way P automata and two-way computations. Similar formalizations can be found in [6], where also the idea of multi-head automata is considered.

Another idea is that of considering still more restrictive classes of P automata, thus obtaining "sub-regular" classes of languages, as it is possible for finite automata and regular grammars. Restrictions on the form of rules, number of membranes, number of rules can be useful.

A possible extension is to also associate an output with a computation, thus obtaining a *transducer*. Actually, neither the simple idea of a language generating P system was considered so far. There are cases when the generative and the accepting versions of the same device from classic automata and language theory behave differently, hence the question might be non-trivial also for P automata. The same occurs with the description of transductions which can be obtained in this framework. Find examples and counterexamples, maybe characterizations (as there are for finite states transductions). Note that we work here in the non-extended and in a different set-up than the one from [1].

We give only a simple example of a transducer (it seems that even the identity function requests a P transducer with two membranes). Indeed, consider an alphabet $V = \{a_1, \ldots, a_n\}$ and construct:

$$\begin{aligned}\Pi \;=\; & (V, [\,[\;\;]_2\,]_1, c, a_1a_2\ldots a_n,\\ & \{(a_i, out; a_i, in) \mid 1 \leq i \leq n\},\\ & \{(a_i, out; c, in) \mid 1 \leq i \leq n\} \cup \{(c, out)\}\\ \cup\; & \{(a_i, out; ca_j, in) \mid 1 \leq i, j \leq n\}\\ \cup\; & \{(ca_i, in) \mid 1 \leq i \leq n\}.\end{aligned}$$

The rule $(a_i, out; a_i, in)$ ensures the equality of the input with the output, and the change of the symbol a_i is done with the help of c (and the object present in the inner membrane); the same object c determines the end of the computation.

Some ideas can also come from the membrane computing itself: what about minimal parallelism, sequential or asynchronous P automata, introducing rules for handling also membranes, controls about using the rules (promoters, inhibitors, etc.)?

A wealth of research topics, hence an area of investigation which is worth considering, in spite (or, better, because) of the simple definitions and the "eccentric" computing power.

For further reading

[2–4,7,10,11,19].

Acknowledgements

The work of Gh. Păun is supported by Proyecto de Excelencia con Investigador de Reconocida Valía, de la Junta de Andalucía, grant P08 – TIC 04200.

References

[1] G. Ciobanu, Gh. Păun, Gh. Ştefănescu, P transducers, New Generation Computing 24 (1) (2006) 1–28.
[2] E. Csuhaj-Varjú, P automata, in: G. Mauri, et al. (Eds.), Membrane Computing, International Workshop, WMC5, Milano, Italy, 2004, Selected Papers, in: LNCS, vol. 3365, Springer, Berlin, 2005, pp. 19–35.
[3] E. Csuhaj-Varjú, O.H. Ibarra, G. Vaszil, On the computational complexity of P automata, Natural Computing 5 (2006) 109–126.
[4] E. Csuhaj-Varjú, M. Oswald, G. Vaszil, P automata. Chapter 6 in [16], pp. 144–167.
[5] E. Csuhaj-Varjú, G. Vaszil, P automata or purely communicating accepting P systems, in: Gh. Păun, et al. (Eds.), Membrane Computing. International Workshop, WMC 2002, Curtea de Argeş, Romania, August 2002. Revised Papers, in: LNCS, vol. 2597, Springer, 2003, pp. 219–233.
[6] E. Csuhaj-Varju, G. Vaszil, A connection between finite dP automata and multi-head finite automata. Proc. Twelfth Intern. Conf. on Membrane Computing, CMC12, M. Gheorghe, Gh. Păun, S. Verlan (eds.), Fontainebleau, France, August 2011, pp. 109–126.
[7] J. Dassow, G. Vaszil, P finite automata and regular languages over countably infinite alphabets, in: H.J. Hoogeboom, et al. (Eds.), Membrane Computing, International Workshop, WMC7, Leiden, The Netherlands, 2006, Selected and Invited Papers, in: LNCS, vol. 4361, Springer, 2006, pp. 367–381.
[8] R. Freund, M. Kogler, Gh. Păun, M.J. Pérez-Jiménez, On the power of P and dP automata, in: Annals of Bucharest University, in: Mathematics-Informatics Series, vol. 63, 2009, pp. 5–22.
[9] R. Freund, M. Oswald, A short note on analysing P systems, Bulletin of the EATCS 79 (October) (2002) 231–236.
[10] R. Freund, M. Oswald, L. Staiger, ω-P automata with communication rules, in: C. Martin-Vide, et al. (Eds.), Membrane Computing, International Workshop, WMC 2003, Tarragona, July 2003, Selected Papers, in: LNCS, vol. 2933, Springer, 2004, pp. 203–217.
[11] M. Oswald, P Automata. Ph.D. Thesis, TU Vienna, 2003.
[12] Gh. Păun, Membrane Computing. An Introduction, Springer, Berlin, 2002.
[13] Gh. Păun, M.J. Pérez-Jiménez, Solving problems in a distributed way in membrane computing: dP systems, International Journal of Computers, Communication and Control 5 (2) (2010) 238–252.
[14] Gh. Păun, M.J. Pérez-Jiménez, P and dP automata: a survey, Lecture Notes in Computer Science (in press).
[15] Gh. Păun, M.J. Pérez-Jiménez, An infinite hierarchy of languages defined by dP systems, Theoretical Computer Science, in press (doi:10.1016/j.tcs.2011.12.053).
[16] Gh. Păun, G. Rozenberg, A. Salomaa (Eds.), Handbook of Membrane Computing, Oxford University Press, 2010.
[17] G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages. 3 volumes, Springer, Berlin, 1998.
[18] A. Salomaa, Formal Languages, Academic Press, New York, 1973.
[19] The P Systems Website: http://ppage.psystems.eu.