P Systems with External Input and Learning Strategies

José M. Sempere

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
jsempere@dsic.upv.es

Abstract. This is a preliminary work in which we propose a variant of P systems by adding in every region a dynamic engine that allows the change of the internal rewriting rules along the computation time, obtaining in this way a new family of P systems with adaptation to changing environments. They will be called *adaptative P systems*. Here, the engine that we propose to act inside every membrane is based on learning algorithms under the grammatical inference framework. The behavior of every region changes according to the information received from the external environment and the internal regions.

1 Introduction

P systems [2,13] have been proposed as a computational mechanism with high parallelism inspired by the membrane structure of the cell. In the recent times, several variants of P systems motivated by biological or formal language theory aspects have been proposed. We will refer, among others, to *Generalized P-Systems* [5], *Hybrid P-Systems* [7], *Gemmating P-Systems* [1], *Tissue P-Systems* [8], *P-Systems with carriers* [9], etc. Here, we will propose a new variant of P systems based on the information received outside the external membrane and on the way the system interacts with this information. Our model is inspired by systems that learn, as proposed in the artificial intelligence framework. We refer to [11] for a formal introduction to such systems.

It is a fact that the external information received by the living cell can change its behavior drastically (e.g., this is a common situation in virus attacks or cancer diseases). The system has to adapt itself to the new situation or it could be damaged. We try to explore, under a formal framework, the generative capacity of P systems in such situations. Furthermore, we initiate an exploration beyond recursively enumerable languages by setting this framework to several hierarchies proposed in classical computability theory, for example the arithmetic hierarchy [15].

The structure of this work is as follows: First, we introduce some basic concepts and notation about formal language theory, P systems and inductive inference systems. Then, we introduce the simplest model to work with external input and we prove some equivalence properties with respect to general P systems. We introduce more sophisticated P systems by adding some dynamics in

C. Martín-Vide et al. (Eds.): WMC 2003, LNCS 2933, pp. 341–356, 2004

the internal rules by using learning engines (inductive inference algorithms). We relate some properties of these systems with some language classes defined in oracle computation models given by classical recursion theory. Finally, we will overview some future research guidelines and discussion about this work.

2 Basic Concepts

First, we refer to [6] for basic concepts on formal language theory. Let V be an alphabet and V^* the set of all strings (words) defined over V. The empty string will be denoted by ε and $V^+ = V^* - \{\varepsilon\}$. For any string $x \in V^*$, we denote by $perm(x)$ the set of permutations over the string x. The set of segments of any string x is denoted by $segment(x)$. Given any string x, $|x|_a$ denotes the number of occurrences of the symbol a in x. A language L is any subset of V^* and $segment(L) = \bigcup_{x \in L} segment(x)$. Given any set A, $\mathcal{P}(A)$ will denote the power set of A. A family of languages can be defined by using some characterization results on formal grammars or abstracts machines. We denote the family of recursively enumerable languages by $\mathcal{RE}$.

Now, we introduce some basic concepts about P systems. A general P system of degree m, according to [13], is a construct

$$\Pi = (V, T, C, \mu, w_1, \ldots, w_m, (R_1, \rho_1), \ldots, (R_m, \rho_m), i_0),$$

where:

- V is an alphabet (the *objects*)
- $T \subseteq V$ (the *output alphabet*)
- $C \subseteq V$, $C \cap T = \emptyset$ (the *catalysts*)
- μ is a membrane structure consisting of m membranes
- w_i, $1 \leq i \leq m$, is a string representing a multiset over V associated with the region i
- R_i, $1 \leq i \leq m$, is a finite set of *evolution rules* over V associated with the ith region and ρ_i is a partial order relation over R_i specifying a *priority*. An evolution rule is a pair (u, v) (written $u \to v$) where u is a string over V and $v = v'$ or $v = v'\delta$ where v' is a string over

$$\{a_{here}, a_{out}, a_{in_j} \mid a \in V, 1 \leq j \leq m\}$$

and δ is an special symbol not in V (it defines the *membrane dissolving action*).
- i_0 is a number between 1 and m and it specifies the *output* membrane of Π; it can also be equal to ∞, and in this case the output is read outside the system).

The language generated by Π in external mode ($i_0 = \infty$) is denoted by $L(\Pi)$ and it is defined as the set of strings that can be defined by collecting the objects that leave the system and arranging them in the leaving order (if several objects leave the system at the same time then permutations are allowed). The set of

numbers that represent the objects in the output membrane i_0 will be denote by $N(\Pi)$. Obviously, both sets $L(\Pi)$ and $N(\Pi)$ are defined only for *halting computations*. We suggest to the reader Păun's book [13] to learn more about P systems.

The next ingredient that supports our work is a learning system. This is a classical topic in artificial intelligence. There have been different proposals of learning systems based on different theories. Here, we will fix our attention to *inductive systems*. The motivation to use this approach will be explained later. First, we define an *inductive learning paradigm* through the following items:

1. A *naming* function to design hypotheses and target concepts. Hypotheses and concepts are formal languages defined over a fixed alphabet. Mainly, the naming function can be referred to *generators* (any language is defined by some construct that generates it) or *acceptors* (any language is defined by abstract machines like Turing machines or finite automata).
2. An information protocol to present examples and/or counterexamples of the target concept. The examples are those strings that belong to the target language (they will be represented by a superscript $+$) while the counterexamples are strings out of the language (they will be represented by the superscript $-$).
3. A success criterion for the task. As in optimization problems the main success criterion are *identification* (the learning system must guess the exact target language) or *approximation* (some error in the symmetric difference between the target and the guess is permitted).

When the target languages are represented by formal grammars and/or Turing machines, then the inductive learning paradigm becomes a grammatical inference one [16].

An interesting learning criterion is the *identification in the limit* proposed by E. M. Gold in 1967 [4]. The approach is the following: Let $\mathcal{L}$ be a family of recursively enumerable languages and $L \in \mathcal{L}$ (the target language). An algorithm A identifies L in the limit if A produces a sequence $H_1, H_2, \ldots, H_i, \ldots$ such that for a given integer i we have $H_i = H_j$ whenever $j \geq i$ and $H_i = L$. Here, the hypothesis sequence is produced from an information source that gives to A some examples (and possibly counterexamples) of the language L. We will say that an inference algorithm is *incremental* if every output hypothesis is constructed from a previous one and new input data.

A well known result is that the family $\mathcal{RE}$ is identifiable in the limit from examples and/or counterexamples [4]. The proof follows from an enumeration of Turing machines and the subsequent hypothesis changes according with the information received so far. On the other hand, an algorithmic technique based on Nerode's congruences was proposed by Oncina and García to identify any regular language in the limit in polynomial update time [10].

Finally, it can be argued that any system that learns can be viewed as an abstract machine with an *oracle* [15]. So, different results from computability theory about relativized computation can be applied to learning systems.

2.1 General P Systems with Covering Rules

Now, we will introduce a variant of P systems by defining a new kind of evolution rules that we will name *covering rules*. A covering rule will be in the form $u \to v$ or $u \to v\delta$ where $u = u_1u_2 \ldots u_n$ with $u_i \in \mathcal{P}(V^*)$ and v is a string over $\{\mathcal{P}(V^*)_{here}, \mathcal{P}(V^*)_{out}, \mathcal{P}(V^*)_{in_j}\}$ where the semantics of the last set is similar to the basic definition except that we introduce languages instead of symbols. So, for example, the rule $ab^* \to a^*_{here}$ in region i means that all symbols b together with one symbol a are replaced by symbols a. For instance, using this rule, *abbbb* is substituted by *aaaaa*. Another example could be the rule $\{a^nb^n \mid n \geq 1\} \to c^*_{out}d_{here}(ab)^*_{in_k}$ and it means that in the region where the rule is associated, every pair of symbols a and b is replaced by two symbols c which are sent out the region, one pair of symbols a and b which is sent to the region k and, finally, after making the substitution on all pairs one symbol d is produced in the current region.

The order in the covering rules is important. For example, the rule $ab^*c^* \to c^*_{here}d^*_{here}$ means that set of symbols b and c (if a symbol a is presented) is substituted as follows: the unique symbol a disappears, every symbol b is substituted by a symbol c and every symbol c is substituted by a symbol d.

The term *covering* refers to the situation in which the rule *covers* an undefined number of objects. We will make use of such rules in the next section.

3 AP Systems with Static Rules

Our purpose here is to incorporate external information in P systems in order to obtain dynamic systems that change during the computation. Given that there exist different ways and choices to make so, we will introduce different types of P systems. Anyway, there is a feature common to all of them: the system adapts itself to environment changes. Informally, we say that a P system is an AP system if the set of internal rules of every region changes during the computation (here AP means P systems with adaptation or *adaptative P systems*). Our motivation in this work is the study of the relationship between learning strategies and adaptation in P systems. Other approaches to different AP systems will be discussed in the conclusions section.

First, we consider the case that external information only changes the active rules of every membrane in the system. So, we will provide an external alphabet and a finite set of evolution rules to manage external information. Formally, the P system is defined as

$$\Pi = (V, T, C, E, \mu, w_E, w_1, \ldots, w_m, (R_1, \rho_1, \gamma_1), \ldots, (R_m, \rho_m, \gamma_m), i_0),$$

where all elements are defined according to the general setting, except E (the *external input alphabet*) with $E \cap V = \emptyset$, and γ_i that denotes a finite set of evolution rules of the form $u \to v$ where u is a string over $(E \cup V)^*E^+(E \cup V)^*$ and $v = v'$ or $v = v'\delta$ where v' is a string over $\{a_{here}, a_{out}, a_{in_j} \mid a \in (V \cup E), 1 \leq$

$j \leq m\}$. The string w_E will denote (in a multiset way) the objects that there exist in the environment where the P system acts.

A configuration of the system will be defined by the tuple $(\mu', w_{i_E}, w_{i_1}, \ldots, w_{i_k})$. The computation will be performed in a way similar to the general setting with the following exceptions:

1. At any computation step, an object from w_E can enter in region 1 through the skin membrane. In this case, the rules of γ_1 can start to manipulate those objects. The rest of the rules in the regions where the external objects have entered are inhibited (this means that the rules of ρ_i will not be applied even if they can). If all the objects enter in region 1 at any computation step we will say that the system works with *complete information loading*.
2. If any object from w_E enters in region 1, then it can disappear from w_E (*nonpersistent environment*) or remain in w_E (*persistent environment*).
3. The result of the computation is collected in region i_0 or outside the system. Here, we have to make the following remark: if the system works within a persistent environment, then it will never halt (given that the external objects enter region 1 at any computation step and rules from γ_1 work with them). In such situation the halting criterion is substituted by an inhibition one. We will accept the result of a computation only when no set of rules ρ_i can be applied again

The latter systems, according to the previous definition and remarks, will be called *simple AP systems*. Let us consider an example of a simple AP system working with and without persistent environments.

Example 1 Let Π be the simple AP system shown in Figure 1.

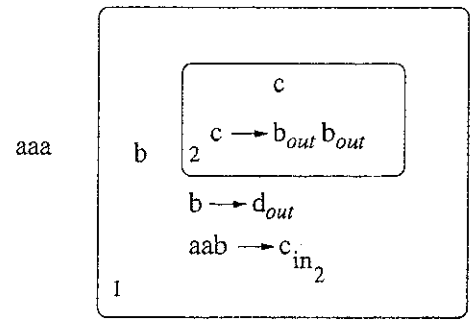

Fig. 1.

Here, the external input alphabet is $\{a\}$, $i_0 = \infty$ and the output alphabet is $\{d\}$. Let us analyze the behavior of the system in different situations.

First, let's consider that Π works in a nonpersistent environment and complete information loading. We will set the state of the system by defining the strings w_i and w_E at every computation step. At step 0 $w_E = aaa$, $w_1 = b$ and $w_2 = c$. At step 1 every symbol from the external environment enters in region

1 (complete information loading) and we have $w_E = \varepsilon$ (nonpersistent environment), the object b in region 1 is transformed in d and exits the system; at the same time the object c in region 2 is transformed in two objects b that go to region 1 so $w_1 = aaabb$ and $w_2 = \varepsilon$. At step 2, in region 1 only the rule that works with external objects can be applied so two objects a and one object b are replaced by one object c that goes to region 2. Observe that the rest of the objects in region 1 are not transformed because the rule that works only with internal objects are inhibited due to the presence of external objects. So, at step 2, $output = d$, $w_E = \varepsilon$, $w_1 = ab$ and $w_2 = c$. At step 3, only the rules from region 2 can be applied, so the object c is transformed in two objects b and leave the region 2 to region 1. In this case, $output = d$, $w_E = \varepsilon$, $w_1 = abbb$ and $w_2 = \varepsilon$. This is a stationary situation, given that no rule can be applied again (the rules from region 1 keep on inhibited due to the presence of object a).

Now, let us consider that Π works in a persistent environment and complete information loading. The sequence of the computation is as follows: At step 0 $w_E = aaa$, $w_1 = b$ and $w_2 = c$. At step 1 every symbol from the external environment enters in region 1 (complete information loading) and we have $w_E = aaa$ (persistent environment), the object b in region 1 is transformed in d and exits the system, and the object c in region 2 is transformed in two objects b that go to region 1, so $w_1 = aaabb$ and $w_2 = \varepsilon$. At step 2, in region 1 only the rule that works with external objects can be applied so two objects a and one object b are replaced by one object c that goes to region 2. Observe that the rest of the objects in region 1 are not transformed because the rules that work only with internal objects are inhibited due to the presence of external objects. Again, every object from the environment goes into region 1 (persistent environment). So, at step 2, $output = d$, $w_E = aaa$, $w_1 = aaaab$ and $w_2 = c$. At step 3, the rules from region 2 can be applied, so the object c is transformed in two objects b which leave the region 2 to region 1. On the other hand, in region 1 two objects a and one object b go to region 2 and again every external object enters in region 1. In this case, $output = d$, $w_1 = aaaabb$ and $w_2 = c$. We can observe that the output of the system is the same as in the previous case but Π does not enter a stationary situation, given that there will be always a sufficient number of objects that can be transferred to region 1 and, later, to region 2.

If the system Π works with non complete information loading and persistent environment, then eventually, the output of the system is $\{d^n \mid n \geq 1\}$, given that whenever there be no object a in region 1 an object b can be output as an object d.

We can state the following results to relate these systems with the general ones.

Lemma 1. *Any simple AP system working in any nonpersistent environment can be simulated by a general P system with covering rules.*

Proof First, we will consider that the system works with complete information loading and we take an arbitrary simple AP system Π defined by the tuple

$$\Pi = (V, T, C, E, \mu, w_E, w_1, \ldots, w_m, (R_1, \rho_1, \gamma_1), \ldots, (R_m, \rho_m, \gamma_m), i_0)$$

We construct a general P system Π' as follows

$$\Pi' = (V \cup E, T, C, \mu', w_0, w_1, \ldots, w_m, (R_0, \rho_0), (R'_1, \rho'_1), \ldots, (R'_m, \rho'_m), i_0),$$

where $\mu' = [_0\mu]_0$, $w_0 = w_E$, $R'_i = R_i \cup \gamma_i \cup CR_i$ for $1 \leq i \leq m$, where $CR_i = \{aa_1^* \ldots a_n^* \to a_{here}(a_1^*)_{here} \ldots (a_n^*)_{here} \mid a \in E, a_i \in V\}$, ρ'_i includes ρ_i and $\gamma_i > CR_i > R_i$, $R_0 = \{a \to a_{in_1} \mid a \in E\} \cup \{a \to a_{out} \mid a \in V\}$ and $\rho_0 = \emptyset$. The general P system Π' defined before is constructed by adding a new skin membrane (membrane 0) to system Π and the evolution rules that send the original external objects to the old primary region through membrane 1.

The behavior of the system Π' is equivalent to the simple AP system Π given that the external objects enter in region 1 and then disappear. The problem with priorities is the following: We must inhibit all the rules that do not work with objects from the external environment but now we have no difference between external and internal objects. We solve this situation as follows: First, we give maximum priority to all the rules from γ_i with respect to rules from CR_i and R_i. So, the rules that manipulate external objects are used first. The covering rules from CR_i are used to lock the rest of internal symbols if a external symbol is present in the region and cannot be manipulated by rules from γ_i. That is why $CR_i > R_i$. So, rules from R_i are only used when no external object are present in the region.

Now, let us suppose that Π works with non complete information loading. In this case we propose a construction for a general Π system with covering rules as in the previous case with the following exception: The set of rules from R_0 described in the previous case is substituted by the set $R_0 = \{x \to in_1(x) \mid x \in segment(perm(w_E)) - \{\varepsilon\}\} \cup \{x \to here(x) \mid x \in segment(perm(w_E)) - \{\varepsilon\}\} \cup \{a \to a_{out} \mid a \in V\}$. The transformation $in_1(x)$ is defined as $in_1(x) = x_{1_{in_1}} x_{2_{in_1}} \ldots x_{n_{in_1}}$ with $x = x_1 x_2 \ldots x_n$ and $here(x)$ is defined as $here(x) = x_{1_{here}} x_{2_{here}} \ldots x_{n_{here}}$ with $x = x_1 x_2 \ldots x_n$

The effect of the new rules of R_0 is just considering all the combinations of symbols that can enter in region 1 from the environment (even the case where no symbol enters this region). □

Now, let us see an example of the constructions that we have proposed in Lemma 1.

Example 2 Let Π be the simple AP system from example 1. The equivalent general P system with covering rules and working in a non persistent environment and complete information loading is showed in the Figure 2.

On the other hand, the equivalent general P system with covering rules and working in a non persistent environment and non complete information loading is showed in the Figure 3.

Now we give a result similar to Lemma 1, but for the case of persistent environments.

Lemma 2. *Any simple AP system working in any persistent environment can be simulated by a general P system with covering rules.*

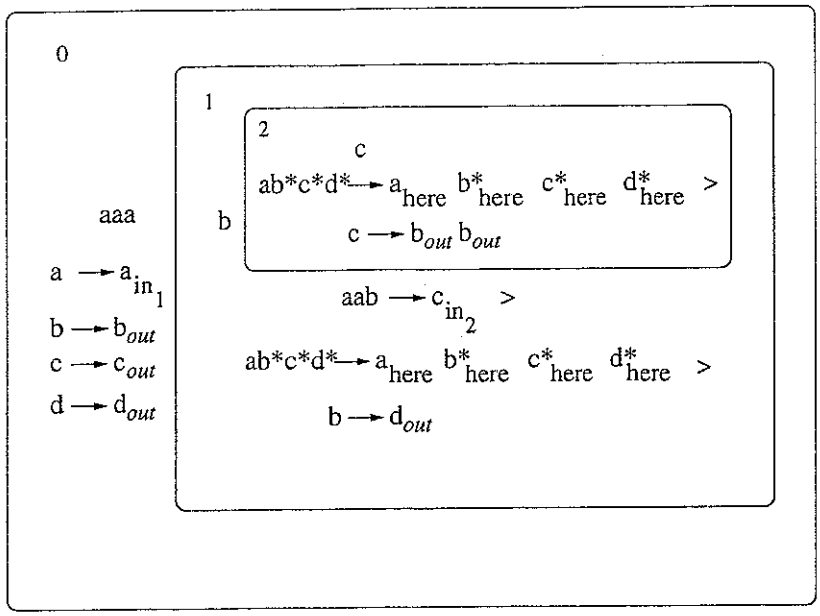

Fig. 2.

Proof. Let us take an arbitrary simple AP system Π defined by the tuple

$$\Pi = (V, T, C, E, \mu, w_E, w_1, \ldots, w_m, (R_1, \rho_1, \gamma_1), \ldots, (R_m, \rho_m, \gamma_m), i_0).$$

We construct a general P system Π' as in the proof of Lemma 1, but the difference is the set of rules R_0. We have two different cases. First, if the system works with complete information loading, then $R_0 = \{a \to aa_{in_1} \mid a \in E\} \cup \{a \to a_{out} \mid a \in V\}$ and $\rho_0 = \emptyset$. Second, if the system works with non complete information loading, then the set R_0 is defined as $R_0 = \{x \to here(x)in_1(x) \mid x \in segment(perm(w_E)) - \{\varepsilon\}\} \cup \{x \to here(x) \mid x \in segment(perm(w_E)) - \{\varepsilon\}\} \cup \{a \to a_{out} \mid a \in V\}$.

The effect of the new rules of R_0 is just considering all the combinations of symbols that can enter in region 1 from the environment (even the case where no symbol enters this region) and keeping the original set of objects of the environment in region 0. □

Now, let us examine an example of constructions as in the proof of Lemma 2.

Example 3 Let Π be the simple AP system from example 1. The equivalent general P system with covering rules and working in a persistent environment and complete information loading is showed in Figure 4.

The corresponding version with non complete information loading is shown in Figure 5.

4 AP Systems: The Learning Approach

Our next step in studying how the external information influences the behavior of a P system is to allow that such information not only activates a finite number

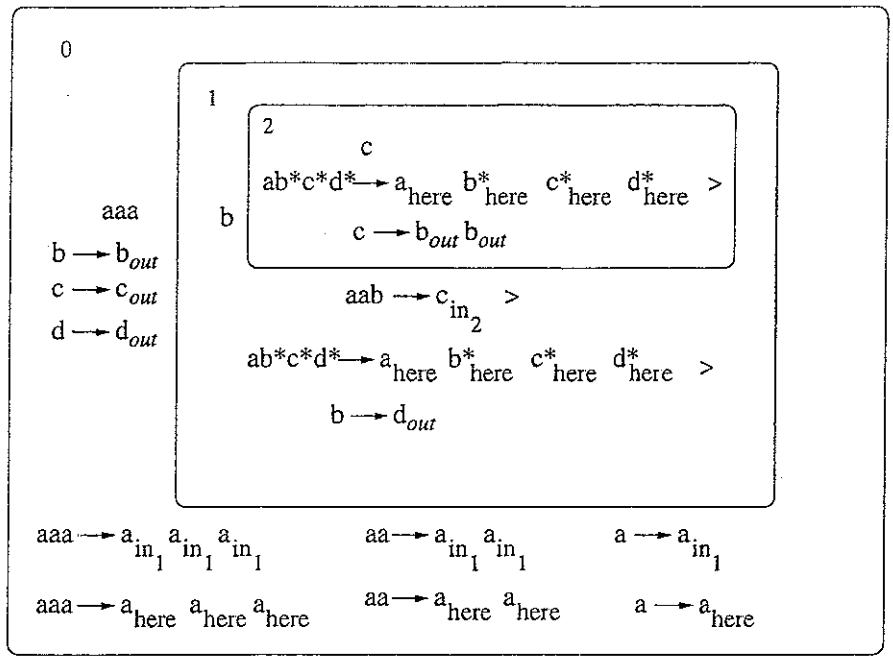

Fig. 3.

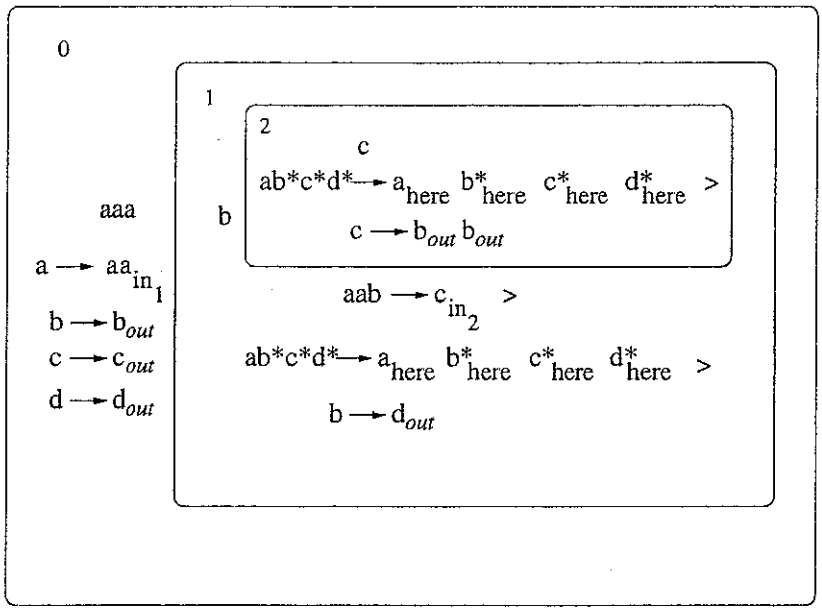

Fig. 4.

of extra rules and inhibit the others, but changes the rules that were working up to that moment. Here, we use a framework similar to those of learning systems.

We define an AP system of degree m based on learning engines as the following tuple:

$$\Pi = (V, T, C, E, \mu, L_E^+, L_E^-, \Im, w_1, \ldots, w_m, \Re_1, \ldots, \Re_m, i_0),$$

where L_E^+ and L_E^- are (possibly non finite) languages over E with $L_E^+ \cap L_E^- = \emptyset$, $\Re_i$ includes R_i and ρ_i which are defined as in the general case, as well as A_i

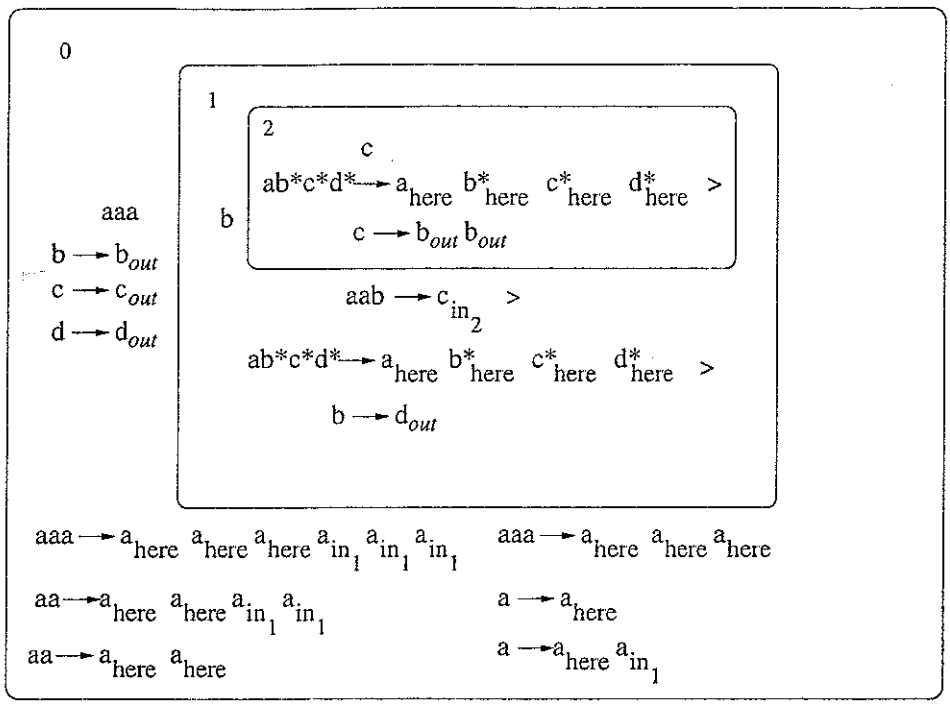

Fig. 5.

which is a deterministic finite automaton. We make the following remarks with respect to the simple case:

1. The words from L_E^+ and L_E^- do not mean multisets in the analysis of A_i but in the evolution rules
2. Eventually, any string from $L_E^+ \cup L_E^-$ can enter into the system and, later, it can move to other regions according to the membrane structure no matter which are the rules working in every region.
3. We say that the AP system is *incremental* if at every computation step only one string from the environment enters into the system. Otherwise, only a finite number of strings do enter.
4. Once any string w_i enters in region j then it is analyzed by the automaton A_j. If $w_i \in L(A_j)$ and $w_i \in L_E^-$ or $w_i \notin L(A_j)$ and $w_i \in L_E^+$, then we say that region j *reacts to the environment*. Observe that this way to analyze the string could be substantially changed by using multiset automata as those described in [3].

The last ingredient that we have added to the system is $\Im$, that means a *learning strategy* (under our approach, a *grammatical inference* algorithm).

We propose two different ways of changing the rules in every region: The static version means that there is a predefined set of rules that only are activated whenever the region reacts to the environment. We denote these systems as AsP systems. The dynamic version means that, in every reacting region, there will be a transformation over the set of evolution rules. We denote these systems as AdP systems. Formally, in an AsP system $\Re_i = (R_i, \rho_i, A_i, \gamma_i)$ where γ_i is a finite set of rules as in the simple AP system case. By the other hand, in an AdP

system $\Re_i = (R_i, \rho_i, A_i, \mathcal{T}_i)$ where $\mathcal{T}_i$ is an algorithm that transform the set of rules R_i and the set of priorities ρ_i into new sets R_i' and ρ_i'.

4.1 Using Regular Learners

As a simple example of the systems described before, we will make $\Im$ to be a grammatical inference algorithm that identifies any regular language in the limit. In the next section, we will use universal learners (inference methods that identify any recursively enumerable language in the limit). The learning of regular languages has been deeply studied along the time. We can mention, among others, the works by Trakhtenbrot and Barzdin [18], and Oncina and García [10], where proposes inference algorithms that learn regular languages and identify them in the limit if a *complete sample*[1] is given as input. The referred algorithms are not incremental. Examples of incremental algorithms that identify any regular language in the limit if all the examples are given in a lexicographic order are the one proposed by Porat and Feldman [12] and Sempere and García [17]. The only request of these algorithms is that a complete presentation of E^* must be given.

Example 4. Let the AsP system Π be defined as follows

$$\Pi = (V, T, C, E, \mu, L_E^+, L_E^-, \Im, w_0, w_1, (R_0, \rho_0, A_0, \gamma_0), (R_1, \rho_1, A_1, \gamma_1), \infty),$$

where $V = \{a, b, e\}$, $T = \{a, b\}$, $C = \emptyset$, $\mu = [_0[_1]_1]_0$, $E = \{c, d\}$, $L_E^+ = c^*$, $L_E^- = \overline{L_E^+}$, $w_0 = ab$, $w_1 = a$, $\rho_0 = \rho_1 = \emptyset$, and A_0 and A_1 are trivial automata accepting the empty language. The rules of γ_0 are $ac \to a_{here}e_{in_1}$ and $bd \to b_{here}$. The only rule of γ_1 is $ae \to a_{here}e_{out}$. R_0 is defined by the rule $abe \to a_{here}b_{here}e_{out}$ and R_1 is empty. Finally $\Im$ is the learning algorithm proposed in [17], where the sequence of hypotheses and words is showed in Figure 6.

The behavior of Π is the following: According to Figure 6, regions 0 and 1 will react when the external input strings are $+c, -d, -dc$ and $-dd$. The rest of strings are accepted by the finite automata that the learning algorithm outputs as showed in Figure 6. So, the last string that makes any region react is $-dd$ and then the system arrives to the correct automata that can accept and reject the rest of strings from L_E^+ and L_E^-. The rules of γ_0 transform every symbol d from a string that makes the region react into a symbol e which is sent to region 1. The symbols c are eliminated. In region 1, every time that a string makes reaction, the symbol e is sent to region 0. So, when the system arrives to a stationary state (i.e., no more reactions are produced), there will be as many symbols e in region 0 as the total number of symbols d that composed the reacting strings. In a simplified manner, the systems behavior is a counter for the symbols d

[1] A complete sample for a target language is a finite set of strings such that if given as input to the learning algorithm, then it outputs the target language in finite time.

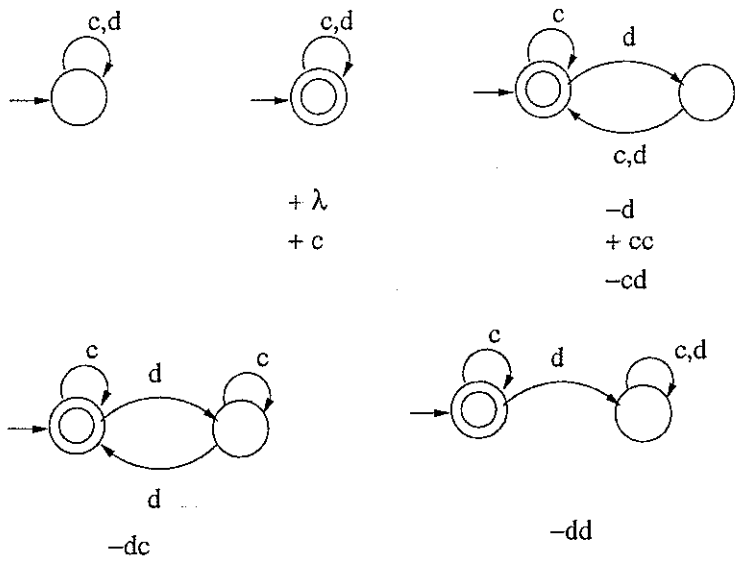

Fig. 6. The hypothesis sequence according with complete ordered information presentation using a learning algorithm for regular languages [17]

needed to identify the regular language in the limit. Observe that if the learning algorithm does not converge to a correct identification, then the system does not output so many symbols.

Example 5 Let the AdP system Π be defined as follows

$$\Pi = (V, T, C, E, \mu, L_E^+, L_E^-, \Im, w_0, w_1, (R_0, \rho_0, A_0, \mathcal{T}_0), (R_1, \rho_1, A_1, \mathcal{T}_1), \infty),$$

where $V = \{a, b\}$, $T = \{a, b\}$, $C = \emptyset$, $\mu = [_0[_1]_1]_0$, $E = \{c, d\}$, $L_E^+ = c^*$, $L_E^- = \overline{L_E^+}$, $w_0 = ab$, $w_1 = ab$, $\rho_0 = \rho_1 = \emptyset$, and A_0 and A_1 are trivial automata accepting the empty language. The algorithms $\mathcal{T}_0$ and $\mathcal{T}_1$ are showed in Figure 7. R_0 is defined by the rule $ab \to a_{here}b_{here}a_{out}$ and R_1 is defined by the rule $ab \to a_{here}b_{here}b_{out}$. Finally $\Im$ is again the learning algorithm proposed in [17].

Observe that, according to algorithm $\mathcal{T}_0$, every time that the region 0 reacts to the environment, the number of symbols a presented in the region and sent to other regions is increased. A similar situation happens in region 2 with symbols b. If region 2 arrives to 5 symbols b, then it is dissolved. So, the number of wrong hypotheses that the learning algorithm outputs is limited in region 2.

4.2 Using Universal Learners: A Characterization of the Arithmetic Hierarchy

In the previous section we have used learning algorithms for the class of regular languages. Now, we introduce more sophisticated learners which are able to identify any recursively enumerable language in the limit (*universal learners*). Most of these algorithms are based in enumeration techniques such as those described in [11] or [4]. The effect of the learning algorithms over AsP or AdP

Input: An evolution rule $u \to v$ over the alphabet $V = \{a, b\}$
Output: An evolution rule $u' \to v'$ over the alphabet V
Method
If $v = w\delta$ then output$(u \to v)$
For every symbol a_{here} in v substitute it by $a_{here}a_{here}$ in v'
For every symbol a_{in_k} in v substitute it by $a_{in_k}a_{in_k}$ in v'
For every symbol a_{out} in v delete it v'
For every symbol b_{here} in v substitute it by b_{here} in v'
For every symbol b_{in_k} in v substitute it by b_{in_k} in v'
For every symbol b_{out} in v delete it v'
Output$(u \to v')$
endMethod

Algorithm $\mathcal{T}_0$

Input: An evolution rule $u \to v$ over the alphabet $V = \{a, b\}$
Output: An evolution rule $u' \to v'$ over the alphabet V
Method
If $v = w\delta$ then output$(u \to v)$
If $|v|_b \geq 5$ then Output$(b \to b\delta)$
For every symbol a_{here} in v substitute it by a_{here} in v'
For every symbol a_{in_k} in v substitute it by a_{in_k} in v'
For every symbol a_{out} in v delete it v'
For every symbol b_{here} in v substitute it by $b_{here}b_{here}$ in v'
For every symbol b_{in_k} in v substitute it by $b_{in_k}b_{in_k}$ in v'
For every symbol b_{out} in v delete it v'
Output$(u \to v')$
endMethod

Algorithm $\mathcal{T}_1$

Fig. 7. Algorithms for the transformation in evolution rules

systems is giving them the power of computing languages beyond $\mathcal{RE}$. In this situation we can use AP systems as oracle machines as in classical computation with Turing machines. If we assume that any recursively enumerable language can be accepted by a general P system according to the literature on membrane computing, then our purpose is allowing the use of oracles in such systems In order to make so, we need to propose new aspects of AP systems:

1. The environment contains strings over any recursively enumerable language.
2. If any region reacts to the environment, then a set of *stand by* rules are activated.
3. The region of the skin membrane can outputs strings with some special marker to make queries.

We will fix our attention to the arithmetic hierarchy [15], and, specifically, to the hierarchy of language classes Σ_i. We propose a method in which relativized

computation can be simulated by using the environment of P systems as oracle machines.

The following definitions come from classical recursion theory and can be consulted in any book such as [15]

Definition 1. *A language L is recursive if there exists a halting Turing machine M such that $L = L(M)$. A language L is A-r.e. if there exists a Turing machine M with a recursive oracle A such that $L(M) = L$*

Definition 2. *(The arithmetic hierarchy) Σ_0 is the class of recursive languages. For each $n \geq 0$ Σ_{n+1} is the class of languages which are A-r.e. for some set $A \in \Sigma_n$. For all n Π_n = co-Σ_n, $\Delta_n = \Sigma_n \cap \Pi_n$.*

We propose an AP system architecture to solve problems from Σ_i. The scheme that we propose is showed in Figure 8.

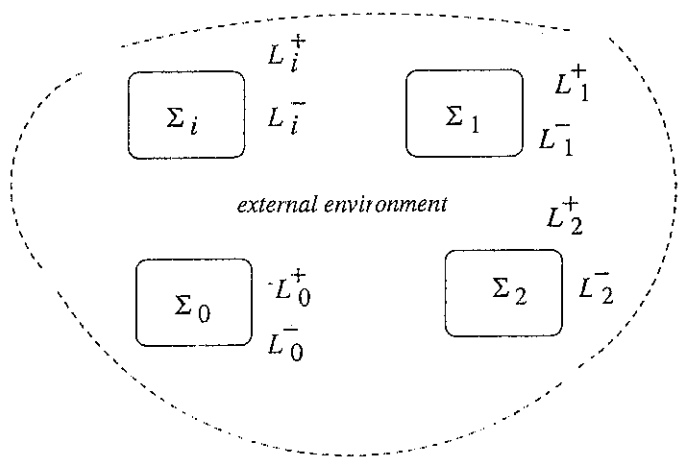

Fig. 8. Using the environment as an oracle channel in the arithmetic hierarchy

This system architecture (which is different from tissue P systems [8]) works as follows. Let us suppose that $L \in \Sigma_2$ and it uses an oracle L' from $\mathcal{RE}$. The AP system that we propose for L uses L' as the environment language and a universal learner. The learner converges to the correct hypothesis, so every query can be directly solved by the learner. If we use this system as an output device, then we can solve languages from Σ_3 and, iteratively, from Σ_i. Given that we can specialize an AP system for every class of the arithmetic hierarchy, they can work together by using a common environment and different output alphabets for every class.

5 Conclusions and Future Research

As we have mentioned at the beginning of this paper, this is a preliminary work about the possibility of introducing the environment and adaptative engines

in P systems. We have proposed new features of P systems. First, the role of covering rules is independent from the environment. The power of P systems with covering rules must be studied apart from the rest of this work. At the first sight, it looks that covering rules help to decrease the description complexity of P systems (we can summarize an undefined set of evolution rules in just one rule). Our purpose is to start a complete study of covering rules and the effect that different language families produce in the generative power of P systems (e.g., what happens if we restrict covering rules to only regular languages, context-free languages, or recursively enumerable languages?).

Second, simple AP systems have been defined and we have proposed different simulation techniques for general P systems

The definition of AP systems suggests the study of different choices that we have proposed. The differences between dynamic and static P systems will be study in future works. The choice about the adaptation triggers should be compared with other choices (here, we have only used finite automata to handle the environment information). Here we have used learning engines to introduce the adaptation of the system to changing environments. We could select other ways to make achieve this (e.g., neural networks with filtering options). We have to study different possibilities of such topic. Nevertheless, the main topic in AdP systems is the definition of transformation algorithms $\mathcal{T}_i$. Here a complete catalog of choices should be exposed. Mainly, we could manage transduction algorithms, learning algorithms, or biologically inspired algorithms (e.g., what happens if splicing operation is applied over the set of evolution rules instead of the set of objects as described in [14]?). We will return to these topics in future works.

References

1. D. Besozzi, G. Mauri, G. Păun, C. Zandron, Gemmating P systems: collapsing hierarchies, *Theoretical Computer Science*, 296 (2003), 253–267
2. C. Calude, G. Păun, *Computing with Cells and Atoms*, Taylor & Francis, 2001.
3. E. Csuhaj-Varjú, C. Martín-Vide, V. Mitrana, Multiset automata, *Multiset Processing* (C. Calude, G. Păun, G. Rozenberg, A. Salomaa, ed.), LNCS 2235, Springer-Verlag, 2001, 65–83
4. E.M. Gold, Language identification in the limit, *Information and Control*, 10 (1967), 447–474.
5. R. Freund, Generalized P-systems, *Proceedings of the 12th International Symposium, FCT'99*, LNCS 1684, Springer-Verlag, 1999, 281–292.
6. J. Hopcroft, J. Ullman, *Introduction to Automata Theory, Languages and Computation*, Addison Wesley Publishing Co., 1979.
7. M. Madhu, K. Krithivasan, A note on hybryd P systems, *Grammars*, 5 (2002), 239–244.
8. C. Martín-Vide, G. Păun, J. Pazos, A. Rodríguez-Patón, Tissue P systems, *Theoretical Computer Science*, 296 (2003), 295–326
9. C. Martín-Vide, G. Păun, G. Rozenberg, Membrane systems with carriers, *Theoretical Computer Science*, 270 (2002), 779–796.
10. J. Oncina, P. García, Inferring regular languages in polynomial update time. *Series in Machine Perception and Artificial Intelligence* (1): *Pattern Recognition and Image Analysis*, World Scientific, 1992

11. D. Osherson, M. Stob, S. Weinstein, *Systems that Learn*, MIT Press, 1986
12. S. Porat, J. Feldman, Learning automata from ordered examples, *Machine Learning*, 7 (1991), 109–138.
13. G. Păun, *Membrane Computing. An Introduction*, Springer-Verlag, 2002.
14. G. Păun, T. Yokomori, Membrane computing based on splicing, *Proceedings of the DIMACS Workshop DNA Based Computers V* (E. Winfree, D. Gifford, eds.), American Mathematical Society DIMACS series Vol. 54 (2000), 217–231.
15. H. Rogers, Jr., *Theory of Recursive Functions and Effective Computability*, MIT Press, 1987
16. Y. Sakakibara, Recent advances of grammatical inference, *Theoretical Computer Science*, 185 (1997), 15–45.
17. J.M. Sempere, P. García, A new regular language learning algorithm from lexicographically ordered complete samples *Colloquium on Grammatical Inference: Theory, Applications and Alternatives* Proceedings edited by S. Lucas as IEE Digest number 1993/092, 1993, pp. 6/1-6/7
18. B. Trakhtenbrot, Y. Barzdin, *Finite Automata: Behavior and Synthesis*, North Holland Publishing Company, 1973.