Covering reductions and degrees in P systems [*] (Extended Abstract)

José M. SEMPERE

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camino de Vera s/n 46020 Valencia, Spain
E-mail: jsempere@dsic.upv.es

1 Introduction

Membrane Computing and P systems [4] can be considered as a substantial part of *natural computing* inspired by some aspects of the biology of the cell and how these aspects can be adapted to propose universal computational models that show high parallelism, distributed and cooperative computation and formal language (or number sets) acceptance or generation.

The number of works related to P systems has been rapidly increased in the recent years. One can refer to [5, 3] for studying several variants of P systems that take into account different biological or computational aspects, for example, *proton pumping, promotors/inhibitors, symport/antiport, membrane creation or deletion*, etc.

A P system consists of a hierarchical finite set of regions where there are an undefined number of objects that react according to a previously defined set of rules. The reactions take part in every region in a parallel nondeterministic manner and the result of the reactions can be communicated to other regions by allowing the pass of objects from one region to a closest one through the membranes. Here, the reactions in every region are defined by a finite set of rules which, usually, change a finite set of predefined objects into another (possibly different) set of objects. The *covering rules* are defined in a different way: The set of objects to be changed are not defined explicitly but according to a pattern set (i.e. a language). The use of covering rules has been explored in previous works. So, in [8], we introduced some aspects about the influence of the external environment over the behavior of a P system and we used covering rules to manage an undefined number of objects coming from the outside environment every time unit. In [9], we showed some applications of covering rules in P systems related to thermodynamic equilibria, description complexity, the control for predominance of objects, etc. Finally, in a recent work [10], we have explored new aspects of covering rules to analyze the description and computational complexity of P systems.

[*]Work partially supported by the Ministerio de Ciencia y Tecnología under project TIC2003-09319-C03-02

In this work we use covering rules to study new computational aspects of P systems. Mainly, we will use covering rules to define *oracle P systems* and we will explore this kind of machines in order to define reductions and degrees as in classical recursion theory.

2 Basic Definition and Notation

Here, we will introduce some basic concepts from formal language theory according to [1, 7], and from membrane computing according to [4].

An alphabet Σ is a finite nonempty set of elements named symbols. A string defined over Σ is a finite ordered sequence of symbols from Σ. The infinite set of all the strings defined over Σ will be denoted by Σ^*. The empty string will be denoted by λ and Σ^+ will denote $\Sigma^* - \{\lambda\}$. A language L defined over Σ is a set of strings from Σ. L can be empty, finite or infinite. The number of strings that belong to a language L is its cardinality.

Now, we will introduce some basic concepts about P systems. A general *P system* of degree m, according to [4], is a construct

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

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) (or $u \to v$) 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*). From now on, we will denote the set $\{here, out\} \cup \{in_k \mid 1 \leq k \leq m\}$ by tar.
- i_0 is a number between 1 and m and it specifies the *output* membrane of Π (in the case that it equals to ∞ 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 by 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 multiplicity of 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 [4] to learn more about P systems.

3 Covering Rules

Now, we will introduce a variant of P systems by defining *covering rules*. Observe that in general P systems, as described in the previous section, different rules can handle identical objects (e.g., $a \to bc$ and $a \to de$ will transform a objects into objects b, c, d, and e). Here, the application of a covering rule can handle the objects in an *exclusive* manner. The term *covering* refers to the situation in which the rule *covers* an undefined number of objects.

In addition, we can see that general P systems are systems with covering rules in which the languages used in the right and left parts of the evolution rules have the cardinality equal to 1.

We provide the formal definition of covering rules, as follows.

Definition 3.1 *Let Π be a P system. We say that r is a covering rule if $r : L_u \to L_v$ or $r : L_u \to L_v\delta$, where $L_u \subseteq V^*$ and $L_v \subseteq (V \times tar)^*$ and δ implies membrane dissolving.*

Now, we will show how covering rules handle the objects of the region.

Example 3.1 *Let $ab^* \to (c_{here})^*$ be a covering rule and abab be the set of objects of its region. Then, after applying the rule, we will obtain the set of objects accc.*

In the previous example we have handled the objects in a *conservative* manner. That is, the number of objects after applying the rule does not decrease. The *non conservative* choice implies that the result of the rule application could be a, ac, acc, or $accc$, given that the object a or the objects b could be substituted by λ (which belongs to c^*), so they disappear.

Example 3.2 *Let the following covering rules be in the same region: $r_1 : ab^+ \to (c_{here})^*$ and $r_2 : ab^+ \to (d_{here})^*$. Let us suppose that the objects in the region before applying the rules are aabbb. If we use the rules in the exclusive mode, then the result will be acccc or adddd (both in conservative mode). That is, the selected rule r_1 or r_2 covers all the objects b.*

If we apply the rules in non exclusive manner, then the combinatorics increase the number of results: we can obtain cccdd or ccddd or acccc or adddd depending on the number of objects b that every rule covers.

We can combine different ways of application of every rule (*exclusive* vs. *non exclusive* together with *conservative* vs. *non-conservative*). Furthermore, in the case that the rules are applied in a non-conservative manner we can arrive to an *extremely non-conservative* mode. So, in example 3.1 the rule can be applied in a non-conservative manner by eliminating some objects, as explained before, or it can increase the number of objects so an undefined number of objects are presented at a given computation step.

All the mentioned ways of application of the covering rules imply that a *second degree of nondeterminism* appears in P systems. Obviously, general P systems are nondeterministic in the first sight. That is, whenever two or more rules can be applied at a given computation step, then the election of the rules to work is made in a non-deterministic manner, so all the combinatorics must be taken into account in order to study the different computation sequences. Here, the covering rules introduce a second degree of non-determinism given that, first a rule is non-deterministically selected and then, if it is a covering rule, then the result of its application is again non-deterministically produced. Let us illustrate this situation in the following example.

Example 3.3 *Consider the rules $r_1 : ab \rightarrow cd$ (non-covering rule) and $r_2 : ab^+ \rightarrow e^+f^+g^+$. Let us suppose that the objects in the region are aabbb. Then, if rule r_1 is selected twice, the result is ccddb, if rule r_2 is selected and it works in the exclusive conservative manner, then the result can be aeefg or aeffg or aefgg. If rule r_2 works in non-exclusive manner, then the rule r_1 could be applied together with the covering rule. In the case that rule r_2 is applied in extremely non-conservative exclusive manner, then an infinite number of results can be obtained.*

We can summarize all the application modes by means of the following definition.

Definition 3.2 *Let Π be a P system, and $r : \alpha \rightarrow \beta$ a covering rule of the system. We will say that P works in*

(a) **conservative mode** *if the application of r will never decrease the number of selected objects in the system,*

(b) **non-conservative mode** *if the application of r can decrease the number of selected objects in the system,*

(c) **exclusive mode***: if rule r is selected and applied, then it covers all the objects according to expression α and no object that belong to α remains free,*

(d) **extremely non-conservative mode** *if the result of applying the rule r is any string that belongs to β and the number of selected objects can be increased.*

Limiting the non-determinism: Indexed covering rules

As mentioned before, the introduction of covering rules in *P* systems increases the non-determinism of the system. Now, we will introduce a variant of covering rules that attempts to reduce this non-determinism. For example, let us take the rule

$ab^+c^+ \rightarrow (c_{here})^+(d_{here})^+(e_{here})^+$. There is no explicit correspondence between symbols of left-hand side and right-hand side. So, the objects *abbcc* could be transformed in *cddee* or *cccde* or *cdeee*, etc. (always in the case that the conservative mode be applied). That is, there is not knowledge to make correspondences between every pair of symbols from left and right sides. In order to control this situation we will introduce indices to make this correspondence explicit.

Definition 3.3 *Let Π be a P system. We will say that r is an indexed covering rule if $r : L_u \rightarrow L_v$ or $r : L_u \rightarrow L_v\delta$, where $L_u \subseteq (V \times \mathbb{N})^*$ and $L_v \subseteq (V \times tar \times \mathbb{N})^*$, δ implies membrane dissolving and $dom_{\mathbb{N}}(L_u) = dom_{\mathbb{N}}(L_v)$* [1].

Example 3.4 *Let $a_1b_2^+c_3^+ \rightarrow (c_{here})_1^+(d_{here})_2^+(e_{here})_3^+$ be an indexed covering rule. The meaning of the rule is that every object a is substituted by at least one object c, every object b is substituted by at least one object d and every object c is substituted by at least one object e.*

Obviously, different objects can collapse to a single one: the rule $a_1b_1^+ \rightarrow (c_{here})_1$ means that one single object a together with an undefined positive number of objects b are replaced by the object c. Observe that the previous rule always works in the non-conservative mode.

4 Oracle P Systems

Once we have introduced the basic definitions of P systems and covering rules notions, we will fix our attention to the arithmetic hierarchy. The following definitions come from classical recursion theory and can be consulted in any book such as [6]

Definition 4.1 *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 $L(M) = L$.*

The notion of *oracle* is quite simple. No matter what computational model we use, the oracle always answer in one time unit a question about formal languages. We can use different oracle machines such as the following ones:

- *membership* oracles

  Given any string x, the oracle L answer 'yes' (usually defined by the output '1') if $x \in L$ and 'not' (usually defined by the output '0') otherwise.

- *superset/subset* oracles

  In superset (resp. subset) oracles, given any language L_1, the oracle L answer 'yes' if $L_1 \supseteq L$ (resp. $L_1 \subseteq L$) and 'not' otherwise.

[1]Given $L \subseteq (V \times \mathbb{N})^*$ or $L \subseteq (V \times tar \times \mathbb{N})^*$ we will denote by $dom_{\mathbb{N}}(L)$ the set of positive integers that appear in the description of L.

- *equivalence* oracles

  Given any description M of a language (usually M is an abstract machine or grammar), the oracle L answer 'yes' if $L(M) = L$ and 'not' otherwise.

Obviously, there have been other proposals for oracle types along the time. Usually they have been used in computational learning theory or complexity theory, among other areas. Now, we will use only membership oracles such as the one defined before. Furthermore, in Definition 4.1 A is a membership oracle.

The definition of oracles in computability theory has been commonly used to explore new aspects such as relativized complexities or families of non recursively enumerable languages. The last approach will be formally used in the next definition

Definition 4.2 *(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 $\Pi_n =$ co-Σ_n, $\Delta_n = \Sigma_n \cap \Pi_n$.*

Constructing an oracle region

An oracle region with language L_{or} will consists of an inner *query* region where the oracle makes its performance together with an *answer* region where the question is answered. Formally, the structure of the region will be $[[\ ]_{query}[\ ]_{answer}]_{oracle}$ where the region *query* will include the covering rules $r_1 : L_{or} \rightarrow 1_{in_{answer}}$ and $r_2 : \Sigma^* \rightarrow 0_{in_{answer}}$ with the priority $r_1 > r_2$. The language L_{or} is the language of the oracle and the covering rule r_1 works as follows: if the *query* string belongs to L_{or}, then an object 1 is transmitted to the region *answer*, otherwise the rule r_2 is applied and then an object 0 is transmitted to the region *answer*. In the region *answer* the only rules are $0 \rightarrow 0_{out}$ and $1 \rightarrow 1_{out}$. Finally the oracle region consists of the following rules $1\# \rightarrow 1_{out}$ and $0\# \rightarrow 0_{out}$ and the rules $a \rightarrow a_{in_{query}}$ for every object a except $\#$.

The P system with an oracle region will work as follow:

1. The *query* string is sent to the oracle region.
2. The oracle region sends the query string to the *query* region.
3. The *query* region sends the answer (0 or 1) to the *answer* region.
4. The answer region sends the answer to the oracle region.
5. The oracle region sends the answer to the outside whenever the answer is requested.

Observe that the system sends first the query string and later it sends the *answer requesting* symbol #. In addition, the oracle computation time is always constant given that it consists only of four computation steps. Here, the answer region is used to ensure some synchronization of queries and answers. Furthermore, the queries could be sent at an initial stage and all the answers could be received together.

Obviously, we can use the last construction in order to characterize the languages from Σ_i, Π_i and Δ_i. First, for all those languages in Σ_i we can use P systems with an oracle region where the query region uses a rule $r_1 : L_{or} \rightarrow 1_{in_{answer}}$ with $L_{or} \in \Sigma^{i-1}$. We can use P systems for Σ_i and inverting the output in order to accept languages in Π_i or making intersections in order to accept languages in Δ_i.

5 Covering reductions and degrees

The last construction induces a new way of reduction technique inspired from Turing reduction between languages [6]. Here we can use oracle P systems in order to make reductions between languages. It will define degrees between language classes. Now we will give some basic definitions related to this topic

Definition 5.1 (**covering reduction**) *Let L_1 and L_2 be two languages. We will say that $L_1 \leq_{Pc} L_2$ if there exists a P system Π such that*

1. *Π uses an oracle region with language L_2, and*
2. *Π accepts the language L_1.*

The last definition is quite similar to Turing reduction between languages. Obviously if any language can be Turing-reduced to another language, then it can be reduced with a covering reduction.

Definition 5.2 *Let L_1 and L_2 be two languages. We say that $L_1 \equiv_{Pc} L_2$ if $L_1 \leq_{Pc} L_2$ and $L_2 \leq_{Pc} L_1$.*

Property 5.1 *The relation $\equiv_{Pc}$ between languages is a binary equivalence relation (i.e., it is reflexive, symmetric and transitive)*

Definition 5.3 *Let L be a language. Then $\equiv_{Pc} [L] = \{L' \mid L \equiv_{Pc} L'\}$. We will say that $\equiv_{Pc} [L]$ is a covering degree.*

Finally, the last definition induces, for any language L, a language class $\mathcal{L}^L_{\equiv_{Pc}}$ which consists of all the languages in $\equiv_{Pc} [L]$. The relation between language classes $\mathcal{L}^L_{\equiv_{Pc}}$ will be studied in the near future.

References

[1] J. Hopcroft, J. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison Wesley Publishing Co., 1979.

[2] M. Li, P. Vitànyi. *An Introduction to Kolmogorov Complexity and its Applications.* Springer-Verlag. 1993.

[3] C. Martín-Vide, G. Mauri, G. Păun, G. Rozenberg, A. Salomaa (Eds.). *Membrane Computing. International Workshop WMC-2003. LNCS*, Vol. 2933. Springer, 2004.

[4] G. Păun. *Membrane Computing. An Introduction.* Springer, 2002.

[5] G. Păun, G. Rozenberg, A. Salomaa, and C. Zandron (Eds.). *Membrane Computing. International Workshop WMC-CdeA 2002. LNCS*, Vol. 2597. Springer, 2003.

[6] H. Rogers, Jr. *Theory of Recursive Functions and Effective Computability.* MIT Press, 1987.

[7] G. Rozenberg, A. Salomaa (Eds.). *Handbook of Formal Languages* Vol. 1. Springer, 1997.

[8] J.M. Sempere. P systems with external input and learning strategies. *Proceedings of the Workshop on Membrane Computing WMC03. LNCS* Vol. 2933, pp 341–356. Springer, 2004.

[9] J.M. Sempere. Covering Rules in P systems: some preliminary ideas. Work volume of the *Second Brainstorming Week on Membrane Computing.* TR 01/2004 of the Department of Computer Science and Artificial Intelligence, pp 449-456. University of Seville. 2004.

[10] J.M. Sempere. Complexity applications of covering rules in P systems. (*submitted*)