On P Systems with Membrane Creation

Carlos MARTÍN-VIDE
Research Group on Mathematical Linguistics
Rovira i Virgili University
Pl. Imperial Tàrraco 1, 43005 Tarragona, Spain
E-mail: cmv@astor.urv.es

Gheorghe PĂUN[1]
Institute of Mathematics of the Romanian Academy
PO Box 1-764, 70700 Bucureşti, Romania
E-mail: gpaun@imar.ro

Alfonso RODRÍGUEZ-PATÓN
Department of Artificial Intelligence, Faculty of Computer Science
Polytechnical University of Madrid
Campus de Montegancedo, Boadilla del Monte 28660, Madrid, Spain
E-mail: arpaton@fi.upm.es

Abstract

We first give a general result about P systems with symbol-objects, which says that systems with membrane creation with only one initial membrane can simulate usual systems without using additional membranes (and this implies as a direct consequence a recent result of Mutyam and Krithivasan), then we extend the membrane creation feature to P systems with string-objects and we prove the computational universality of such systems.

Keywords: Molecular computing, membrane computing, recursively enumerable sets, matrix grammars

1 Introduction

P systems [6] are computability models which abstract from the way the living cells process chemicals in the compartmental structure defined by their membranes. Several variants were proved to be computationally complete (see details and references in [7]), many variants were considered which are able to solve NP-complete problems in polynomial (even linear) time (see, e.g., [3], [5], [7]). One variant having both these features (when some other ingredients are present, such as the possibility of controlling the membrane permeability, hence the communication among compartments) is that which has the possibility of producing new membranes: recently, in [4], a way to solve the Hamiltonian Path Problem was given, and a characterization of Turing computable numbers was found, by P systems with only one initial membrane and using in total only four membranes.

[1]Work supported by a grant of NATO Science Committee, Spain, 2000–2001, and by Facultad de Informatica, Universidad Politecnica de Madrid

We give here a general result, which shows that systems with symbol-objects without membrane creation, using at most n membranes, can be simulated by systems with membrane creation which start from one unique membrane and use in total at most n membranes. In this way, the universality result from [4] can be obtained by combining our result and the result from [1]. Because such a general result cannot be obtained also for systems with string-objects (unless we also use string replication), we settle this case in a direct way, by proving a universality result with a reduced number of membranes. (This result corresponds to similar results from [8], [1], but instead of controlling the membrane permeability we use here the membrane creation.)

2 P Systems: A Short Introduction

We informally introduce here the basic ideas of computing with membranes. For further details the reader is referred to the survey paper [7]. An up-to-date bibliography of the area as well as several downloadable papers can be found at the web address `http://bioinformatics.bio. disco.unimib.it/psystems`.

A P system consists of a *membrane structure* (usually, represented graphically by an Euler-Venn diagram and mathematically by a string of labeled parentheses, indicating the membranes and their relative position), in the *regions* of which we have *objects*; the objects can be symbols from a given alphabet or strings over a given alphabet. The objects evolve according to *evolution rules*, which are also associated with regions. The objects can also pass through membranes (this operation is called a *communication*), the membranes can be dissolved or they can get thicker; in the former case, the objects of the dissolved membrane are left free in the membrane surrounding it, in the latter case the communication through the membrane is not allowed (but the membrane can become permeable at a later step). The evolution rules are used in the maximally parallel manner, chosing nondeterministically the rules and the objects to which they are applied. In this way, we get transitions from a configuration of the system to the next configuration; a sequence of transitions forms a computation and with a halting computation we can associate an output, by taking into consideration the objects (irrespective whether they are symbols or strings) which leave the system during the computation. Several further ingredients can be considered, for instance, concerning the control of communication, the way of handling membranes, etc, but we refer to the literature for details.

3 Membrane Creation for P Systems with Symbol-Objects

We start by a more formal definition of a symbol-object P system. Such a device (of degree $m \geq 1$) is a construct

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

where V is the alphabet of the system, $T \subseteq V$ is the *terminal* alphabet, μ is a membrane structure with m membranes, injectively labeled by $1, 2, \ldots, m$, and represented by a

string of matching parentheses, $w_1, \ldots, w_m$ are strings over V, representing the multisets of objects initially present in the regions $1, 2, \ldots, m$ of the system, and $R_1, \ldots, R_m$ are finite sets of evolution rules over V associated with the regions $1, 2, \ldots, m$ of μ.

The general form of rules is $a \to v$, where $a \in V$ and v is a string of symbols of the forms $(b, here), (b, out), (b, in)$, with $b \in V$; the meaning is that after replacing a with the objects specified by v, these latter objects remain in the same region, go out of the region, or go to any of the immediately inner membranes (if any exists, and if they are permeable at that moment, but we do not enter here into such details), depending on the associated indication *here, out, in*, respectively. By using these rules in the maximally parallel manner (all objects which can evolve should evolve), we get transitions between system configurations, hence computations. Because we deal with multisets of objects, the result of a computation is a number (or a vector of natural numbers). We denote by $N(\Pi)$ the set of all numbers computed by Π.

Besides rules as above, we can also consider rules with *catalysts*, that is, of the form $ca \to cv$, where $c \in V$ is a catalyst (it is never changed by using a rule), or for *membrane creation*. Such a rule is of the form $a \to [_i v]_i$, with $a \in V, v \in V^*$, and i a number from a given list; the idea is that the object a creates a new membrane, having inside the objects indicated by v, and with the label i. The label is one from a given list, so that we know which are the evolution rules associated with the new membrane.

For a usual system, the important parameter about the membrane structure is the number of membranes (the degree of the system), while for systems with membrane creation we have to consider three parameters: the number of initial membranes (n_1), the maximal number of membranes simultaneously present in the system (n_2) during any successful (i.e., halting) computation, and the number of all possible types of membranes (n_3). We say that (n_1, n_2, n_3) is the *profile* of the system.

The following result, although intuitively easy to prove, has interesting consequences for the study of P systems with membrane creation, as it makes non-necessary the direct study of the power of such systems.

Theorem 1. *Given a P system of degree m, without membrane creation, one can construct an equivalent P system with membrane creation, with the profile $(1, m, m)$.*

Proof. The idea is that given a system $\Pi = (V, T, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m)$, with the membrane structure μ of *height* h (there are h levels of membranes, counting both the external membrane and the elementary membranes), we can construct an equivalent system Π', with only one initial membrane, which in the first phase of a computation constructs the membrane structure μ and the multisets $w_1, \ldots, w_m$, by making use of the membrane creation. This phase takes $2h$ steps. After completing it, the obtained system is exactly Π, hence the equivalence is obvious.

Here are a few details of this construction. The initial configuration of Π' is $[_1 d_1^{(0)}]_1$ (note that 1 is assumed the label of the external membrane of Π and that the membranes of Π are assumed labeled in a one-to-one manner). To each set $R_i, 1 \leq i \leq m$, we add the following rules:

$$d_i^{(j)} \to c_{k_1}^{(j+1)} \ldots c_{k_i}^{(j+1)} e_i^{(j+1)}, \ 0 \leq j \leq 2h-1,$$
$$e_i^{(j)} \to e_i^{(j+1)}, \ 1 \leq j \leq 2h-1,$$

$$e_i^{(2h-1)} \to w_i,$$

where $k_1, \ldots, k_i$ are the labels of membranes placed in membrane with label i (if there is no such a membrane, then the rule becomes $d_i^{(j)} \to e_i^{(j+1)}$). Moreover, if $up(i)$ is the membrane placed immediately above membrane i, then in $R_{up(i)}$ we also add the rules

$$c_i^{(j)} \to [_i d_i^{(j+1)}]_i, \ 1 \leq j \leq 2h-1.$$

The objects d introduce the membrane creating objects c and the counters e; all these symbols have superscripts indicating the step of the computation, so that the process lasts exactly as needed, $2h$ steps, while both the branching and the creation of membranes is accomplished – in the last step one also introduces the multisets $w_i, \ldots, w_m$ in the corresponding membranes.

The reader can easily see that the obtained system is equivalent to the initial one, and this is true independently of any other ingredients used by Π, as the initial sets of rules $R_1, \ldots, R_m$ were just augmented with the above mentioned rules. □

By combining this result with one of the results from [1], we obtain the fact that P systems with membrane creation using catalysts and the membrane permeability control, with one initial membrane and at most 4 membranes (more precisely, of profile (1, 4, 4)) can generate all Turing computable sets of vectors of natural numbers, a result which is proven directly in [4]; actually, also a slight improvement of the result from [4] is obtained, as weaker target indications are used in communication (only *here, out, in*, without commands in_j, as used in [4], which also specify the label of the lower membrane where the objects are sent).

4 The Case of String-Objects

So far, the membrane creation feature was not considered for P systems with string-objects, but it can be introduced in a natural way: consider rules of the form $a \to [_i v]_i$; when rewriting a string $w_1 a w_2$ by such a rule we get $[_i w_1 v w_2]_i$, that is, a new membrane with label i containing the string $w_1 v w_2$.

In such a framework, the proof of Theorem 1 cannot be extended to the case of string-objects, because we cannot enforce the creation of several membranes at the same level (unless if we use more powerful rules than of the form $a \to [_i v]_i$, for instance, for creating two or more membranes at the same time, or for replicating strings). Because we want to remain here in the "classic" framework of rewriting P systems, we leave as an *open problem* the question of obtaining a general result as that in Theorem 1 also for this case, and we will directly prove a universality result for rewriting P systems with membrane creation.

The systems we consider are of the form $\Pi = (V, T, \mu, L_1, \ldots, L_n, R_1, \ldots, R_m)$, where V is the alphabet, $T \subseteq V$ is the terminal alphabet, μ is the initial membrane structure, of degree n, $L_1, \ldots, L_n$ are finite languages over V (strings present in the initial configuration of the system), and $R_1, \ldots, R_m$ are finite sets of context-free rewriting rules over V (the rules associated with the possible membranes, of types $1, 2, \ldots, m$; each membrane is

identified with its label, which in this way also identifies the corresponding rules). The rules are of two basic forms, string evolution rules, and membrane creation rules: $a \to v(tar)$ and $a \to [_i v]_i$, respectively, where $a \in V, v \in V^*$, $1 \leq i \leq m$, and $tar \in \{here, out, in\}$. The meaning is that if a string $w_1 a w_2$ is rewritten, then the string $w_1 v w_2$ will be placed into the membrane indicated by tar, or to the new membrane $[_i \,]_i$, respectively.

Each string which can be rewritten must be rewritten, but this is done by only one rule (if there are several strings in a membrane, then they are processed in parallel (simultaneously), but each of them is rewritten sequentially, in a context-free manner). The strings which are sent out of the system during a computation form the language $L(\Pi)$ generated by Π.

In what follows we also allow rules for dissolving a membrane, and they are of the form $a \to v\delta$: after using this rule in a membrane i, the membrane is dissolved, all its strings (and inner membranes) remain free in the enclosing membrane; the external membrane of the system is never dissolved.

We denote by $CRP(n_1, n_2, n_3)$ the family of languages generated by rewriting P systems with membrane creation, using the communication commands *here, out, in* (the indication *here* is always omitted), the membrane dissolving action, and of profiles componentwise smaller than (n_1, n_2, n_3). By RE we denote the family of recursively enumerable languages.

Theorem 2. $RE = CRP(1, 2, 4)$.

Proof. We only have to prove the inclusion $\subseteq$, the other one can be obtained by a straightforward (but long) construction.

We start from the fact that each language in RE can be generated by a matrix grammar with appearance checking in the strong binary normal form [2]. Such a grammar is a construct $G = (N, T, S, M, F)$, where N, T are disjoint alphabets (the nonterminal and the terminal alphabet, respectively), $N = N_1 \cup N_2 \cup \{S, \#\}$, with these three sets mutually disjoint (S is the axiom of the grammar), F is a set of occurrences of rules from M, and M is a finite set of sequences (they are called *matrices*) of the following forms:

1. $(S \to XA)$, with $X \in N_1, A \in N_2$,
2. $(X \to Y, A \to x)$, with $X, Y \in N_1, A \in N_2, x \in (N_2 \cup T)^*$,
3. $(X \to Y, A \to \#)$, with $X, Y \in N_1, A \in N_2$,
4. $(X \to \lambda, A \to x)$, with $X \in N_1, A \in N_2$, and $x \in T^*$.

Moreover, there is only one matrix of type 1 and F consists exactly of all rules $A \to \#$ appearing in matrices of type 3; $\#$ is a trap-symbol, because once introduced, it is never removed. A matrix of type 4 is used only once, in the last step of a derivation.

The derivation starts from S and proceeds by using the matrices from M; when using a two-rule matrix, both the two rules should be used, possibly skipping the rules from F if their left hand symbols are not present in the current string (that is why one says that they are used in the appearance checking mode).

As proved in [2], each recursively enumerable language can be generated by a matrix grammar as above, with only two symbols A used in rules of the form $A \to \#$.

Take such a grammar G, let $B^{(1)}, B^{(2)}$ be the two symbols in N_2 for which we have rules of the form $B^{(j)} \to \#$, let us assume that we have k matrices of the form $m_i : (X \to \alpha,$

$A \to x), X \in N_1, \alpha \in N_1 \cup \{\lambda\}, A \in N_2$, and $x \in (N_2 \cup T)^*$, $1 \leq i \leq k$ (that is, without rules to be used in the appearance checking manner). The matrices of the form $(X \to Y, B^{(j)} \to \#), X, Y \in N_1$ (that is, with rules used in the appearance checking manner), are labeled by m_i, with $i \in lab_j$, for $j = 1, 2$, such that lab_1, lab_2 and $lab_0 = \{1, 2, \ldots, k\}$ are mutually disjoint sets.

We construct the system $\Pi = (V, T, [_1\ ]_1, L_1, R_1, R_2, R_3, R_4)$, where

$$\begin{aligned} V &= N_1 \cup N_2 \cup T \cup \{X_{i,j} \mid X \in N_1, 1 \leq i \leq k, 0 \leq j \leq k\} \\ &\cup \{A_{i,j} \mid A \in N_2, 1 \leq i, j \leq k\} \cup \{C, f, Z\}, \\ L_1 &= \{XA, C\}, \text{ for } (S \to XA) \text{ being the initial matrix of } G, \end{aligned}$$

and with the following sets of rules:

R_1: $X \to [_2 X_{i,i}]_2$, for $m_i : (X \to \alpha, A \to x) \in M, 1 \leq i \leq k$,
$C \to C(in)$,
$A_{i,j} \to Z$, for $1 \leq i, j \leq k$,
$A_{i,j} \to A_{i,j-1}(in)$, for $1 \leq i \leq k, 2 \leq j \leq k$,
$A_{i,1} \to [_4 x]_4$, for $m_i : (X \to \alpha, A \to x) \in M, 1 \leq i \leq k$,
$f \to \lambda(out)$,
$X \to [_{2+j} Y]_{2+j}$, for $m_i : (X \to Y, B^{(j)} \to \#) \in M, i \in lab_j, j = 1, 2$;

R_2: $A \to [_3 A_{i,i}]_3$, for $1 \leq i \leq k$,
$C \to C\delta$;

R_3: $X_{i,j} \to X_{i,j-1}(out)$, for $1 \leq i \leq k, 2 \leq j \leq k$,
$X_{i,1} \to X_{i,0}\delta$, for $1 \leq i \leq k$,
$B^{(1)} \to Z$,
$C \to C$,
$C \to C\delta$;

R_4: $B^{(2)} \to Z$,
$C \to C\delta$,
$X_{i,0} \to \alpha$, for $m_i : (X \to \alpha, A \to x) \in M, 1 \leq i \leq k$,
$X_{i,j} \to Z$, for $1 \leq i, j \leq k$.

This system works as follows. Initially, we have two strings, XA and C, in the unique initial membrane, with label 1. If we start by using a rule $X \to [_2 X_{i,i}]_2$, then we will simulate a matrix $m_i : (X \to \alpha, A \to x) \in M, 1 \leq i \leq k$, in the following way. The membrane with label 2 is created; simultaneously, the rule $C \to C(in)$ sends the symbol C into this membrane. At the next step, in membrane 2 we create a membrane with label 3, containing a string of the form $X_{i,i} w_1 A_{j,j} w_2$, and the rule $C \to C\delta$ dissolves membrane 2. The symbol C returns to the external membrane.

In membrane 3 we decrease the second component of the subscript of X and the string is sent to membrane 1; simultaneously, C is sent to the inner membrane. In membrane 3 we can use the rule $C \to C$ an arbitrary number of times, or the rule $C \to C\delta$, which dissolves this membrane. If we dissolve membrane 3 while the string from membrane 1 contains a symbol $A_{j,k}$ with $k \geq 2$, then the only rule which can be applied here is

$A_{j,k} \to Z$, and the trap-symbol Z is introduced, which can never be removed, hence no terminal string is obtained. Therefore, we have not to use the rule $C \to C\delta$, or to use it at the right time, as specified bellow.

Assume that membrane 3 exists (we use the rule $C \to C$ inside it), hence in membrane 1 we can use the rule $A_{j,k} \to A_{j,k-1}(in)$. The string is sent to membrane 3, where again the second component of the subscript of X is decreased by one, and the string is sent out. This process is iterated, thus alternately decreasing the second components of the subscripts of X and A.

We distinguish three cases:

Case 1: If $i < j$, hence the rule $X_{i,1} \to X_{i,0}\delta$ is used in membrane 3 while the string contains a symbol $A_{j,k}$ with $k \geq 2$, then the rule $A_{j,k} \to Z$ must be used in membrane 1, hence no terminal string is obtained.

Case 2: If $i > j$, hence we use the rule $A_{j,1} \to [_4 x]_4$ in membrane 1, having a string of the form $X_{i,k}w_1xw_2$ with $k \geq 1$ in membrane 4, then the only rule from this membrane which can be applied to this string is $X_{j,k} \to Z$, and again no terminal string will be obtained.

Case 3: If $i = j$, then when dissolving membrane 3, in membrane 1 we have a string of the form $X_{i,0}w_1A_{i,1}w_2$. We use the rule $A_{i,1} \to [_4 x]_4$, and the string $X_{i,0}w_1xw_2$ is introduced in membrane 4. At the same time, the string-symbol C arrives in membrane 4. In the next step we use the rules $X_{i,0} \to \alpha$ and $C \to C\delta$, hence we return to membrane 1 the strings $\alpha w_1 x w_2$ and C, the first one being the string obtained by a correct simulation of the matrix $m_i : (X \to \alpha, A \to x) \in M, 1 \leq i \leq k$.

If $\alpha \in N_1$, then the processs can be continued, if $\alpha = f$, then the rule $f \to \lambda(out)$ can be used and the string is sent out of the system. If the string is terminal, then it is accepted in $L(\Pi)$, otherwise it is "lost". (The auxiliary string C remains in the system and cannot be rewritten, because no inner membrane exists.)

Assume now that we start in the unique membrane 1 by using a rule $X \to [_{2+j} Y]_{2+j}$, for some $j = 1, 2$ and $m_i : (X \to Y, B^{(j)} \to \#), i \in lab_j$. This produces a new membrane, containing a string Yw; at the same time, the string C is sent to the newly generated membrane. In membrane $2 + j$ we check whether $B^{(j)}$ is present in the string (in the positive case the trap-symbol Z is introduced). If no occurrence of $B^{(j)}$ appears in the string Yw, then it cannot be rewritten, and waits unchanged. In membrane 3 we can use for a while the rule $C \to C$, but eventually the rule $C \to C\delta$ must be used, otherwise we get no output. In membrane 4 we have to use the rule $C \to C\delta$ immediately. In both cases, the membranes are dissolved, hence the string Yw is returned to membrane 1. In this way, the correct use of the matrix m_i was simulated, with its second rule used in the appearance checking mode.

The process can continue. It is clear that in this way all derivations in G can be simulated in Π, and that if a terminal string is sent out of the system Π, then it can also be generated by the grammar G. That is, $L(G) = L(\Pi)$. Because we start from a unique initial membrane, at each step (of a correct computation) we have at most two membranes in the system, out of four possible membranes, the profile of Π is $(1, 2, 4)$, hence the proof is complete. □

Note that in the previous result we have not used the membrane thickness control (as done in [8] and [1]); it remains as an *open problem* to see whether by using such a feature one can decrease the profile of the used system (we conjecture that this is not possible). It also remains to be investigated the case of P systems with string-objects processed by other operations than rewriting, for instance, by DNA-like splicing (splicing P systems can be found in several papers).

We conclude with the remarks that the possibility of creating membranes is not only biochemically motivated, but also interesting from mathematical points of view; also, it is worth remembering its computational usefulness in solving hard problems in a tractable time, as shown in [4].

References

[1] R. Freund, C. Martin-Vide, Gh. Păun, Computing with membranes: Three more collapsing hierarchies, submitted, 2000.

[2] R. Freund, Gh. Păun, On the number of non-terminals in graph-controlled, programmed, and matrix grammars, *Proc. MCU Conf.*, Chişinău, 2001 (M. Margenstern, Y. Rogozhin, eds.), *Lect. Notes in Computer Sci.*, **2055**, Springer-Verlag, 2001.

[3] S. N. Krishna, R. Rama, A variant of P systems with active membranes: Solving NP-complete problems, *Romanian J. of Information Science and Technology*, 2, 4 (1999), 357–367.

[4] M. Mutyam, K. Krithivasan, P systems with membrane creation: Universality and efficiency, *Proc. MCU Conf.*, Chişinău, 2001 (M. Margenstern, Y. Rogozhin, eds.), *Lect. Notes in Computer Sci.*, **2055**, Springer-Verlag, 2001, 276–287.

[5] A. Obtulowicz, Membrane computing and one-way functions, *Intern. J. Found. Computer Science*, 2001, in press.

[6] Gh. Păun, Computing with membranes, *Journal of Computer and System Sciences*, 61, 1 (2000), 108–143.

[7] Gh. Păun, G. Rozenberg, A guide to membrane computing, *Theoretical Computer Science*, 2001, in press.

[8] Cl. Zandron, G. Mauri, Cl. Ferretti, Universality and normal forms on membrane systems, *Proc. Intern. Workshop Grammar Systems 2000* (R. Freund, A. Kelemenova, eds.), Bad Ischl, Austria, July 2000, 61–74.