Membrane Systems with External Control

Robert Brijder [1], Matteo Cavaliere [2,3],
Agustín Riscos-Núñez [3], Grzegorz Rozenberg [1], Dragoş Sburlan [3,4]

(1) Leiden Inst. of Advanced Computer Science (LIACS),
Leiden University, Leiden, The Netherlands,
rbrijder@liacs.nl, rozenber@liacs.nl
(2) Microsoft Research - University of Trento,
Centre for Computational and Systems Biology, Trento, Italy,
(3) Dept. of Computer Science and Artificial Intelligence,
University of Seville, Seville, Spain,
matteo.cavaliere@msr-unitn.unitn.it, ariscosn@us.es,
dsburlan@univ-ovidius.ro
(4) Faculty of Mathematics and Informatics,
Ovidius University, Constantza, Romania.

Abstract. We consider the idea of controlling the evolution of a membrane system. In particular, we investigate a model of membrane systems using promoted rules, where a string of promoters (called the control string) "travels" through the regions, activating the rules of the system. This control string is present in the skin region at the beginning of the computation – one can interpret that it has been inserted in the system before starting the computation – and it is "consumed", symbol by symbol, while traveling through the system. In this way, the inserted string drives the computation of the membrane system by controlling the activation of evolution rules. When the control string is entirely consumed and no rule can be applied anymore, then the system halts – this corresponds to a successful computation. The number of objects present in the output region is the result of such a computation. In this way, using a set of control strings (a control program), one generates a set of numbers. We also consider a more restrictive definition of a successful computation, and then study the corresponding model.

In this paper we investigate the influence of the structure of control programs on the generative power. We demonstrate that different structures yield generative powers ranging from finite to recursively enumerable number sets.

In determining the way that the control string moves through the regions, we consider two possible "strategies of traveling", and prove that they are similar as far as the generative power is concerned.

1 Introduction

Membrane systems (also referred to as P systems) were introduced in 1998 by Gh. Păun as computing devices inspired by the structure and functioning of living cells. Since their introduction, several models of P systems have been

investigated, many of them being proved to be computationally complete. The reader is referred to the monograph [6], and to an up-to-date bibliography of this research area available at the P systems web-page, [11].

In nature, the behavior of cells can be influenced by the signals (controls) that they receive from the "outside". Thus, it may be possible to drive the evolution of a living cell by providing the cell with a specific control.

With this motivation in mind, we introduce and investigate a model of P systems, called *string-controlled P systems* (in short, SC P systems). This model is based (with some modifications) on membrane systems with promoters, introduced in [1]. There, the presence of promoters is used to activate, during the computation, certain rules of the system. The biological motivation is the fact that chemical reactions in living cells can be promoted (or inhibited) by the presence of various enzymes.

A string of promoters (called the *control string*), "produced" by the environment, is present in the skin region of the system at the beginning of a computation. This string (that acts like an external control) travels through the regions of the system, possibly promoting (with its leftmost symbol) the rules of the region where it currently resides. Each time the string moves from one region to another, its leftmost symbol (used as a promoter) gets consumed. When the whole string is consumed, and no rule can be applied in any region, then the system halts, completing a successful computation. The output of such computation is the number of objects present in the output region when the system halts.

We shall also consider another sort of successful computation, which additionally has to satisfy a "clean ending condition" (which requires that an a priori specified "undesirable" object is not present in any region upon the completion of the computation).

In this way, an SC P system generates the set of numbers composed by the outputs of all its computations. Also, a membrane system with a collection of control strings (called the *control program*) generates a set of numbers, which is defined as the union of the sets generated for each single string.

In this paper we pay special attention to SC P systems where all evolution rules of the system are promoted – hence, only the rules defined in the region where the control string currently resides, and whose promoter matches the leftmost symbol of the control string, may be active. In particular, we investigate how the structure of the control program influences the generative power of such systems, which are called *fully-promoted SC P systems*.

We show that if the control program is finite, then the generative power corresponds exactly to the family of finite sets of numbers. On the other hand, if the family of recursively enumerable languages is used as the control program, then, not surprisingly, the resulting generative power corresponds to the family of Turing computable sets of numbers. Several intermediate results are obtained by balancing the structure of the control program and the power of the evolution rules used by the system.

We consider two different ways (operating modes) for a control string to travel through the regions of the system: either the string must move at each step (mode (1)), or it is allowed to remain in the same region for several consecutive steps until it decides (nondeterministically) to move again (mode (2)). We prove that, under some natural conditions on the control program, these two modes are similar as far as the generative power is concerned.

The paper is organized as follows. Section 2 recalls some basic notions of formal languages theory used throughout the paper. A formal definition of SC P systems is presented in Section 3. In Section 4 we show that the generative power of classes of fully-promoted SC P systems with a natural condition on the control program family are "almost" independent on the chosen operating mode of the movement of the control string. In Section 5 we consider structures of control that yield a generative power strictly weaker than RE, and in Section 6 structures that yield the computational completeness.

We conclude the paper by suggesting a number of open problems and research directions.

2   Preliminaries

Let us briefly recall some notions and results of formal languages to the extent needed in this paper – in this way we establish the basic notation and terminology needed later on. For more details the reader can consult standard books, such as [10], [2], and the handbook [9].

An *alphabet* V is a finite set of symbols. By V^* we denote the set of all strings over V, the empty string is denoted by λ, and $V^+ = V^* - \{\lambda\}$.

The *length* of a string $w \in V^*$ is denoted by $|w|$, while the number of occurrences of $a \in V$ in w is denoted by $|w|_a$. For a language $L \subseteq V^*$, the set $length(L) = \{|w| \mid w \in L\}$ is called the *length set* of L.

If FL is a family of languages then NFL is the family of length sets of languages in FL.

We denote by FIN, REG, CF, CS and RE the families of finite, regular, context-free, context-sensitive and recursively enumerable languages, respectively. Accordingly, for instance, the family of length sets of languages in RE is denoted by NRE (this is the family of all recursively enumerable sets of natural numbers).

A multiset over V is a mapping $M : V \longrightarrow \mathbb{N}_0$; assigning to each $a \in V$ a multiplicity $M(a)$. Commonly, multisets are represented by strings of symbols. In this representation the order of symbols does not matter, because the number of copies of an object in a multiset is given by the number of occurrences of the corresponding symbol in the string. Hence, e.g., $a^4 b^3 d$ denotes the multiset consisting of 4 occurrences of a, 3 occurrences of b, and one occurrence of d; the same multiset is also represented by, e.g., $da^2 ba^2 b^2$.

An ET0L system is a construct $G = (\Sigma, T, H, w)$, where Σ is the (total) alphabet, $T \subseteq \Sigma$ is the terminal alphabet, $H = \{h_1, h_2, \ldots, h_k\}$ is a finite set of finite substitutions (tables) over Σ, and $w \in \Sigma^*$ is the axiom; each $h_i \in H$,

$1 \leq i \leq k$, can be represented by a list of context-free productions $A \to x$, such that $A \in \Sigma$ and $x \in \Sigma^*$ (moreover, for each symbol A of Σ and each table h_i, $1 \leq i \leq k$, there is a production in h_i with A as the left hand side). Then G defines, for each $1 \leq i \leq k$, a derivation relation $\Rightarrow_{h_i}$ by $x \Rightarrow_{h_i} y$ iff $y \in h_i(x)$. We write $x \Rightarrow y$ if $x \Rightarrow_{h_i} y$ for some $1 \leq i \leq k$. As usual, $x \Longrightarrow^* y$ denotes the reflexive and transitive closure.

The language generated by G is $L(G) = \{z \in T^* \mid w \Longrightarrow^* z\}$. We denote by $ET0L$ the family of languages generated by ET0L systems, and by $T0L$ the family of languages generated by ET0L systems such that $\Sigma = T$.

A regularly (context-free, respectively) controlled ET0L system, in short E(rc)T0L system (E(cfc)T0L system, respectively), is a pair $\Omega = (G, L)$ where $G = (\Sigma, T, H, w)$ is an ET0L system and L is a regular (context-free, respectively) language over H.

The language generated by Ω is

$$L(\Omega) = \{z \in T^* \mid w = w_0 \Rightarrow_{h_{i_1}} w_1 \Rightarrow_{h_{i_2}} \ldots \Rightarrow_{h_{i_m}} w_m = z, \ h_{i_1} \cdots h_{i_m} \in L\}.$$

We denote by $E(rc)T0L$ the family of languages generated by E(rc)T0L systems, and by $E(cfc)T0L$ the family of languages generated by E(cfc)T0L systems.

The following known inclusions between families of languages will be used in this paper (see, e.g., [10]):

$$FIN \subset CF \subset ET0L \subset CS \subset RE.$$

From [4] we recall the following result.

$$ET0L = E(rc)T0L.$$

Moreover, it is known that for each $L \in ET0L$ there exists an ET0L system G, with only 2 tables, such that $L = L(G)$ (see, e.g., [8]).

A *regularly controlled grammar with appearance checking* is a tuple $G = (N, T, S, P, K, F)$ where $N, T, S,$ and P are the set of nonterminals, the set of terminals, the starting symbol and a finite set of context-free productions, respectively. Each production in P has a uniquely associated label, and the set of all these labels is denoted by $lab(P)$. K is a regular language over $lab(P)$ and $F \subseteq lab(P)$. Let $V = N \cup T$. We say that $x \in V^+$ *derives* $y \in V^*$ in the appearance checking mode by application of $A \to w$ with label p (written as $x \Rightarrow_p^{ac} y$) if either $x = x_1 A x_2$ and $y = x_1 w x_2$, or A does not appear in x, $p \in F$, and $x = y$.

The language $L(G)$, generated by G, consists of all strings $w \in T^*$ such that there is a derivation $S \Rightarrow_{p_{i_1}}^{ac} w_1 \Rightarrow_{p_{i_2}}^{ac} w_2 \Rightarrow_{p_{i_3}}^{ac} \ldots \Rightarrow_{p_{i_n}}^{ac} w_n = w$, for some $n \geq 1$ and $p_{i_1} p_{i_2} \cdots p_{i_n} \in K$.

By rC_{ac} we denote the family of languages generated by regularly controlled grammars with appearance checking and erasing productions, and by rC we denote the family of languages generated by regularly controlled grammars with erasing productions and without appearance checking (the set F is empty).

The following lemma holds (see, [2]):

Lemma 1. $rC_{ac} = RE$.

In what follows we assume that the reader is familiar with the membrane computing area, in particular with the class of P systems with rewriting rules and symbol-objects, and with the notions of P systems using promoters/inhibitors; for instance as presented in [1, 5, 7] or in Chapter 3 of [6].

3   String-Controlled P Systems

A string-controlled P system, as informally described in Introduction, is defined as follows.

Definition 1. *A string-controlled P system (in short, SC P system) is a construct*

$$\Pi = (V, C, P, L, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, i_0),$$

where:

- *V is the alphabet of Π; its elements are called* objects*;*
- *$C \subseteq V$ is the set of* catalysts*;*
- *P is the set of* promoters*; $P \cap V = \emptyset$;*
- *$L \subseteq P^*$ is the* control program *(each string in L is a* control string*);*
- *μ is a* membrane structure *consisting of m membranes labeled $1, \ldots, m$;*
- *w_i, $1 \leq i \leq m$, are strings that represent the multisets over V initially associated with the regions $1, 2, \ldots, m$ of μ;*
- *R_i, $1 \leq i \leq m$, are finite sets of* evolution rules *associated with the regions $1, 2, \ldots, m$ of μ. Each evolution rule is either of the form $u \to v$ or of the form $u \to v|_p$, where $u \in V^+$, $p \in P$, and $v \in V_{tar}^*$ with $V_{tar} = V \times TAR$, for $TAR = \{here, out\} \cup \{in_j \mid 1 \leq j \leq m\}$;*
- *$i_0 \in \{1, \ldots, m\}$ specifies the output region of Π.*

As usual, the *membrane structure* is a hierarchical arrangement of membranes, embedded in a *skin membrane*, which separates the system from the environment. A membrane without any membrane inside is called *elementary*. Each membrane defines a *region*. For an elementary membrane this is the space enclosed by it, while for a non-elementary membrane, is the space in-between the membrane and the membranes directly included in it. As usual, labels $1, \ldots, m$ identify both membranes and their corresponding regions.

Evolution rules of the form $u \to v|_p$ are called *promoted*, and evolution rules of the form $u \to v$ are called *non-promoted*. An evolution rule is called *non-cooperative* if $u \in V$. Also, an evolution rule is called *catalytic* if it is either of the form $ca \to cv$ or of the form $ca \to cv|_p$, where $a \in (V - C)$, $c \in C$, $p \in P$, and $v \in ((V - C) \times TAR)^*$. The elements of TAR are called *targets*. It is convenient to denote $(a, t) \in V_{tar}$ by a if $t = here$, and by a_t otherwise.

A *configuration* of Π is a description of the membrane structure and of the contents of all the regions. An *initial configuration* of Π consists of the membrane structure μ, the objects initially present in the regions of the system, as described

by $w_1, \ldots, w_m$, and by one string from L, present in the skin region (this string is called *control string*). Notice that Π has a set of initial configurations, one for each element of L.

As standard, we suppose the existence of a global clock that marks the steps of the system.

At each step, the control string moves, in a nondeterministic way, across the regions of Π. We distinguish *two possible modes* of operation for Π: (1) at each step the string moves passing from one region to an adjacent one; (2) at each step the string may move to an adjacent region or remain in the same region. In both cases the control string cannot move to the environment, and when it moves from a region to another one, it loses its leftmost symbol. The leftmost symbol of the control string is called the *head*.

At each step the *head* of the current control string is used as a *promoter* for the rules present in the region where the string resides. A promoted rule is *active* if its promoter is present. The rules that are not promoted are always active.

A *transition* between two configurations of Π is obtained by applying in one step the active rules in each region of Π in a maximally parallel nondeterministic manner. More precisely, if a rule $u \to v \in R_i$ or $u \to v|_p \in R_i$ is active and the multiset u is present in region i, then the *application* of this rule means removing u from region i and adding the objects specified by v in the regions indicated by the corresponding target commands.

A sequence of transitions, starting from an initial configuration of Π, is called *computation*. A computation *halts* when there is no applicable rule in any region of Π *and* the control string is entirely consumed (Π has reached a *halting configuration*).

We shall consider two definitions of *successful* computation for Π:

- in the standard case, we say that all halting computations of Π are successful,
- in the # case, we consider that a halting computation of Π is successful if and only if a special *a priori* designated symbol $\# \in V$ is not present in the halting configuration in any region of Π.

The *result* of a successful computation ω is the number of objects present in the output region i_0 in the halting configuration of ω. Depending on the definition of *successful* computation that is considered, we shall say that the system collects the result in the standard way, or in the # way.

We use the notation $P_m(\alpha, FL)$, where $\alpha \in \{ncoo, coo\} \cup \{cat_k \mid k \geq 1\}$ and FL is a family of languages, to denote the class of SC P systems which use at most m membranes, use only non-cooperative (*ncoo*), cooperative (*coo*), or catalytic with at most k catalysts (cat_k) evolution rules (promoted or not), and use a control program in FL. We call FL the *control program family* of the class. In the *coo* case, there is no restriction on the form of the evolution rules. The prefix (*pro*) is added if *only* promoted rules are used (such systems are called *fully-promoted* SC P systems).

We denote by $N^{(i)}(\Pi), i \in \{1, 2\}$, the set of results of all successful computations of Π starting from any possible initial configuration, operating in mode (i),

and collecting the result in the standard way. Similarly, we denote by $N_{\#}^{(i)}(\Pi), i \in \{1, 2\}$, the set of results of all successful computations of Π operating in mode (i) and collecting the result in the $\#$ way. Moreover, $N^{(i)}P_m(\alpha, FL) = \{N^{(i)}(\Pi) \mid \Pi \in P_m(\alpha, FL), i \in \{1, 2\}\}$ denotes the family of sets of natural numbers generated by SC P systems from $P_m(\alpha, FL)$ operating in mode (i), $i = 1, 2$, and collecting the result in the standard way. The family $N_{\#}^{(i)}P_m(\alpha, FL)$ is similarly defined.

The following inclusions follow directly from the definitions.

Lemma 2.

$$(pro)N^{(i)}P_m(\alpha, FL) \subseteq (pro)N_{\#}^{(i)}P_m(\alpha, FL),$$

$$(pro)N_{\#}^{(i)}P_m(\alpha, FL_1) \subseteq (pro)N_{\#}^{(i)}P_m(\alpha, FL_2), \ if \ FL_1 \subseteq FL_2,$$

$$(pro)N_{\#}^{(i)}P_m(ncoo, FL) \subseteq (pro)N_{\#}^{(i)}P_m(cat_j, FL)$$
$$\subseteq (pro)N_{\#}^{(i)}P_m(cat_{j+1}, FL) \subseteq (pro)N_{\#}^{(i)}P_m(coo, FL),$$

for $j \geq 1$, $i \in \{1, 2\}$, $\alpha \in \{ncoo, coo\} \cup \{cat_k \mid k \geq 1\}$, and FL, FL_1, FL_2 families of languages.

4   Fully-Promoted SC P Systems

In this section we start the investigation of fully-promoted SC P systems. Notice that for such systems in each time step there is activity in at most one region (the region where the control string currently resides). First we give an example that illustrates the functioning of an SC P system. Then we prove the equivalence (as far as the generative power is concerned) between modes (1) and (2).

The following example shows that a given SC P system Π can produce different results according to its functioning mode.

Example 1. Let Π be the SC P system:

$$\Pi = (V, C, P, L, \mu, w_1, w_2, R_1, R_2, i_0),$$

where:

- $V = \{A\}$,
- $C = \emptyset$,
- $P = \{a, b\}$,
- $L = \{ab\}$,
- $\mu = [_1 \ [_2 \ ]_2 \ ]_1$,
- $w_1 = \lambda; w_2 = A$,
- $R_1 = \emptyset$,
- $R_2 = \{A \rightarrow AA|_b\}$,
- $i_0 = 2$.

The system collects the result in the standard way.

When Π operates in mode (1), the unique control string of L is initially present in the skin region and moves, in the next step, to region 2, losing its head a. Therefore, now the rule $A \to AA|_b$ is activated. In the following step the control string exits region 2, entering region 1, and then its last symbol, b, is consumed. Therefore, there is only one successful computation and we have $N^{(1)}(\Pi) = \{2\}$.

If Π operates in mode (2), then the unique control string of L is initially present in the skin region and it may remain there for a certain number of steps; meanwhile nothing is produced in region 2. At a certain step the string moves into region 2, losing its head a. Then, the rule $A \to AA|_b$ is activated in region 2 and it will double the number of objects A at each step, until the string b moves back to region 1. When this happens the computation halts and the number of objects produced in region 2 is a power of two, that is, $N^{(2)}(\Pi) = \{2^n \mid n \geq 1\}$.

Example 1 illustrates that for a given fully-promoted SC P system the generated sets under operating modes (1) and (2) may differ (even drastically). However, the family of sets of numbers generated by a *class* of fully-promoted SC P systems with a control program family that is closed under non-erasing regular substitution is "almost" independent on the chosen operating mode. In fact, we show that any fully-promoted SC P system operating in mode (2) [(1), respectively] can be simulated (in a weak sense) by a fully-promoted SC P system operating in mode (1) [(2), respectively] using the same type of rules, the same type of control program, and using a double number of membranes.

Theorem 1. *Let $\Pi \in (pro)P_m(\alpha, FL)$, where $m \geq 1$, $\alpha \in \{ncoo, coo\} \cup \{cat_k \mid k \geq 1\}$, and FL is closed under non-erasing regular substitution. There exists $\Pi' \in (pro)P_{2m}(\alpha, FL)$, such that*

$$N_\#^{(1)}(\Pi') = \{x + 1 \mid x \in N_\#^{(2)}(\Pi)\}.$$

Proof. Let $\Pi = (V, C, P, L, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, i_0) \in (pro)P_m(\alpha, FL)$, and let us construct $\Pi' = (V', C, P', L', \mu', w_1', \ldots, w_{2m}', R_1', \ldots, R_{2m}', i_0) \in (pro)P_{2m}(\alpha, FL)$ as follows.

Let $V' = V \cup \{Z\}$, with $Z \notin V$ and $P' = P \cup \{d\}$, with $d \notin P$. We consider the regular substitution ϕ defined by $\phi(p) = p(dp)^*$ for each $p \in P$; we define $L' = \phi(L)$ (notice that the substitution is non-erasing and so every family of languages in $\{REG, CF, CS, RE\}$ is closed under this operation). The structure μ' has $2m$ membranes and is obtained from μ by adding, in each region $i, 1 \leq i \leq m$, of μ an (elementary) membrane with label $m + i$. Furthermore we define $w_i' = w_i Z$, for $1 \leq i \leq m$, and $w_i' = Z$, for $m + 1 \leq i \leq 2m$.

We define $R_i' = R_i \cup \{Z \to \#|_d\}$, for $1 \leq i \leq m$, and $R_i' = \{Z \to \#|_p \mid p \in P\}$, for $m + 1 \leq i \leq 2m$.

We shall now show that for every successful computation $\mathcal{C}$ of Π with result x operating in mode (2) there exists a successful computation $\mathcal{C}'$ of Π' with result $x + 1$ operating in mode (1).

Consider an arbitrary computation of Π and consider one of its configurations. Now, suppose that in such configuration the current control string w is in region i of Π and has p as its head. Then, there exists a computation in Π', starting with an "appropriate" control string from L' in the skin, that reaches a configuration having the control string $w' = p(dp)^n x$ present in region i of Π'.

Suppose now that w does not move in Π (Π operates in mode (2)) but remains in the same region for several consecutive steps. This is simulated in Π' by moving w' back and forth between region i and the adjacent dummy region $m+i$, consuming for each movement a symbol p and a dummy promoter d. In this way, an arbitrary computation in Π can be simulated in Π' by a computation starting with an appropriate control string from L'.

On the other hand, Π' does not have other successful computations except those simulating successful computations of Π as described above. In fact, since there is a rule $Z \rightarrow \#|_d$ in every set R'_i, for $1 \leq i \leq m$, which guarantees that the dummy symbol d cannot be used to move the control string into non-dummy regions, otherwise the computation would not be successful. Moreover, if the promoter present immediately to the right of the head of the current control string is non-dummy (i.e., the string is of type pqx, with $p, q \in P$, and $x \in P^*$), then the string must move in a non-dummy region, because otherwise the rules $R'_i = \{Z \rightarrow \#|_p \mid p \in P\}$, for $m + 1 \leq i \leq 2m$, would make the computation unsuccessful, if applied. From the above discussion it should be clear that the theorem holds.

Conversely, a fully-promoted SC P system operating in mode (1) can be simulated (in a weak sense) by a fully-promoted SC P system operating in mode (2), using a structure having a double number of membranes.

Theorem 2. *Let $\Pi \in (pro)P_m(\alpha, FL)$, where $m \geq 1$, $\alpha \in \{ncoo, coo\} \cup \{cat_k \mid k \geq 1\}$, and FL is closed under non-erasing morphism. There exists $\Pi' \in (pro)P_{2m}(\alpha, FL)$, such that*

$$N_\#^{(2)}(\Pi') = \{x + 2 \mid x \in N_\#^{(1)}(\Pi)\}.$$

Proof. Given $\Pi = (V, C, P, L, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, i_0)$ we construct $\Pi' = (V', C, P', L', \mu', w'_1, \ldots, w'_{2m}, R'_1, \ldots, R'_{2m}, i_0)$ as follows.

Let $V' = V \cup \{c, c', Z\}$ and $P' = P \cup \{d, d'\}$, with $c, c', Z \notin V$, and $d, d' \notin P$. We consider the non-erasing morphism ϕ defined by $\phi(p) = pdd'$, for each $p \in P$ – then we set $L' = \phi(L)$ (notice that every family of languages in $\{FIN, REG, CF, CS, RE\}$ is closed under non-erasing morphisms). The membrane structure μ' has $2m$ membranes and is obtained from μ in the following way. In each region $i, 1 \leq i \leq m$, of μ an (elementary) membrane with label $m + i$ is added.

The initial multisets of Π' are $w'_i = cZw_i$, for $1 \leq i \leq m$, and $w'_i = Z$, for $m + 1 \leq i \leq 2m$.

Finally, the evolution rules of Π' are defined in the following way: $R'_i = R_i \cup \{c' \rightarrow c|_{d'}, Z \rightarrow \#|_d\} \cup \{c' \rightarrow \#|_p, c \rightarrow c'|_p \mid p \in P\}$, for $1 \leq i \leq m$. $R'_i = \{Z \rightarrow \#|_p \mid p \in P\}$, for $m + 1 \leq i \leq 2m$.

We will prove now that for every computation of Π operating in mode (1) and producing x, there exists a computation of Π' operating in mode (2) producing $x + 2$.

Consider an arbitrary computation of Π and suppose that, after a certain step k during that computation, the control string $p_{i_1} p_{i_2} \cdots p_{i_j}$, with $p_{i_1}, p_{i_2}, \ldots, p_{i_j} \in P$, is present in region i of Π.

Then, there is a computation of Π' (starting with an "appropriate" control string from L') such that the control string $p_{i_1} dd' p_{i_2} dd' \cdots p_{i_j} dd'$ is present in region i of Π' after a given step k'.

In Π, at step $k + 1$, the string must exit region i (Π operates in mode (1)), entering one of the adjacent regions, chosen nondeterministically, losing the promoter p_{i_1} and getting the promoter p_{i_2} as its new head.

This single step of Π is simulated by Π' in the following consecutive steps. The rules activated by promoter p_{i_1} present in region i of Π' are executed at step k', together with the rule $c \to c'$ present in every region of Π' and activated by any promoter of P. Therefore, at step $k' + 1$ the control string must exit region i, as otherwise in the next step the rule $c' \to \#|_{p_{i_1}}$ would be applied and the entire computation would not be successful.

The only region of Π' where the control string can go to is the dummy region $m + i$ present inside region i (otherwise the promoter d that follows p_{i_1} would activate the rule $Z \to \#|_d$ present in any of the non-dummy adjacent regions of region i and the computation would not be successful). Therefore, suppose the control string goes to region $m + i$, losing in this way the promoter p_{i_1}; the control string may remain in region $m + i$ for an unbounded number of steps (no rule can be applied there). At a certain step k'' the control string comes back to region i, losing the promoter d and having now the promoter d' as its head; therefore, in the step $k'' + 1$ the rule $c' \to c|_{d'}$ is applied. The control string having now d' as head may remain in region i for an unbounded number of steps (no rule can be applied). Eventually, the control string exits region i moving to an adjacent region, losing the promoter d', and having the promoter p_{i_2} (the next non-dummy promoter) as its new head.

Thus, all possible movements of the control string in Π (i.e., all possible computations) are correctly captured by the functioning of Π'; consequently, every successful computation of Π can be simulated by Π'.

Notice that, in Π', if the promoter adjacent to the head of the control string is non-dummy (i.e., it belongs to the set P), then the control string must move in a non-dummy region; otherwise a rule from $R_i' = \{Z \to \#|_p \mid p \in P\}$, $m + 1 \le i \le 2m$, is applied and that would make the computation unsuccessful.

Therefore there are no other successful computations of Π' except those that simulate, in the above described way, successful computations of Π. Thus, the theorem holds.

5   The Influence of the Control Program

Now we analyze in more detail the class of fully-promoted SC P systems operating in mode (1). We show how the structure of the control program and the type of evolution rules influence the generative power of the constructed membrane system. A series of results, ranging from finite power to computational universality, is obtained.

It is worth to remark that one can easily obtain the length set of any language L as output of an SC P system using non-cooperative rules and having L as the control program. Hence, the structure of the control program influences the generative power of SC P systems as the following theorem states.

Theorem 3. $NFL \subseteq (pro)N^{(1)}P_2(ncoo, FL)$.

Proof. Given an arbitrary language L over the alphabet $\Sigma = \{a_1, \ldots, a_n\}$, let us consider a symbol $* \notin \Sigma$, and let $L' = h(L)$ where h is the morphism defined by $h(a) = *a$, for every $a \in \Sigma$.

Now let us construct an SC P system that generates $length(L)$ as follows:

$$\Pi = (V, C, P, L', \mu, w_1, w_2, R_1, R_2, i_0),$$

where:

– $V = \{a'_1, \ldots, a'_n\}$,
– $C = \emptyset$,
– $P = \Sigma$,
– $\mu = [_1 [_2]_2]_1$,
– $w_1 = \lambda; w_2 = a'$,
– $R_1 = \emptyset$,
– $R_2 = \{a' \to a'_{out}a'|_a \mid a \in \Sigma\}$,
– $i_0 = 1$.

At the beginning of the computation one of the strings from L', nondeterministically chosen, is present in the skin region of Π (i.e., region 1). The string moves back and forth between region 1 and region 2 of the system, losing alternatively the symbol $*$ (when passing from region 1 to region 2) and a symbol $a \in \Sigma$ (when moving in the opposite direction). When the string is in region 2, its head $a \in \Sigma$ activates exactly the rule that produces and sends out the symbol a'. Therefore, the number of symbols contained in the output region when the computation halts (the string is entirely consumed) is equal to the number of symbols from Σ that occurred in the inserted control string. Thus Π generates exactly the $length(L)$.

Now, from Corollary 2, Theorem 3 and the Turing-Church thesis, we have that the class of fully-promoted SC P systems using arbitrary RE languages as control program is universal, even when only non-cooperative rules are used. Hence, the following theorem holds.

Theorem 4. $(pro)N^{(1)}P_2(ncoo, RE) = (pro)N^{(1)}_\#P_2(ncoo, RE) = NRE$.

It is now natural to ask what happens if we increase the "power" of the evolution rules used by the P system and we decrease the "power" of the control program.

First we consider SC P systems that use cooperative evolution rules and finite control programs.

Theorem 5. $(pro)N_{\#}^{(1)}P_*(coo, FIN) = (pro)N^{(1)}P_*(coo, FIN) = NFIN.$

Proof. Given an SC P system Π, it is sufficient to notice that the number of distinct nondeterministic computations using only a finite number of steps is bounded by a constant that only depends on Π. Therefore, if Π has a finite control program, then the set of numbers produced is finite. The other inclusion follows from Theorem 3.

Let us prove next that the class of fully-promoted SC P systems using arbitrary context-free (regular, respectively) languages as control program generates exactly the family $NE(cfc)T0L$ (or the family $NE(rc)T0L$, respectively), even with non-cooperative rules.

Theorem 6.

$$(pro)N_{\#}^{(1)}P_2(ncoo, REG) \supseteq NE(rc)T0L = NET0L.$$

$$(pro)N_{\#}^{(1)}P_2(ncoo, CF) \supseteq NE(cfc)T0L.$$

Proof. Given $\Omega = (G, L)$ an arbitrary E(rc)T0L system (or E(cfc)T0L system, respectively) we construct a SC P system Π in $(pro)P_2(ncoo, REG)$ (in $(pro)P_2(ncoo, CF)$, respectively) such that $N_{\#}^{(1)}(\Pi) = length(L(\Omega))$ as follows.
Let $G = (\Sigma, T, H, w)$ with $H = \{h_1, \ldots, h_k\}$. Let

$$\Pi = (V, C, P, L, \mu, w_1, w_2, R_1, R_2, i_0),$$

where:

- $V = \Sigma$,
- $C = \emptyset$,
- $P = \{t_1, \ldots, t_k, d, p\}$, with $d, p \notin \{t_1, \ldots, t_k\}$,
- $L' = \phi(L)dp$ with the morphism ϕ defined by $\phi(t_i) = dt_i, 1 \leq i \leq k$,
- $\mu = [_1 \ [_2 \ ]_2 \ ]_1$,
- $w_1 = \lambda; w_2 = w$,
- $R_1 = \emptyset$,
- $R_2 = \{X \to \alpha|_{t_i} \mid X \to \alpha \in h_i, 1 \leq i \leq k\} \cup \{N \to \#|_p \mid N \in \Sigma - T\}$,
- $i_0 = 2$.

Now Π simulates in region 2 the productions of G, applying the tables according to the strings in L', in such a way that each table h_i has an associated promoter t_i, for every $1 \leq i \leq k$.

The dummy promoter d is only used to be consumed while the control string moves from region 1 to region 2. In this way, the new head of the control string

is a symbol t_i, for some $1 \leq i \leq k$. The final promoter p added as last symbol of any string in L' is used to check whether or not there are still nonterminals in region 2 in the last step of the computation. If this is the case, then the special object $\#$ is produced and the computation is not successful. Consequently, the theorem follows.

We continue now to prove that the reverse inclusions also hold.

Theorem 7.

$$(pro)N_{\#}^{(1)}P_*(ncoo, REG) \subseteq NE(rc)T0L = NET0L.$$

$$(pro)N_{\#}^{(1)}P_*(ncoo, CF) \subseteq NE(cfc)T0L.$$

Proof. Consider a fully-promoted SC P system Π of the form

$$\Pi = (V, C, P, L, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, i_0),$$

such that $C = \emptyset$ and L is a regular (context-free, respectively) language over $P = \{p_1, p_2, \ldots, p_k\}$.

We consider the morphisms φ_i, $1 \leq i \leq m$, defined by $\varphi_i(X) = (X, i)$, for all $X \in V$, $1 \leq i \leq m$. By using these morphisms, we associate with each occurrence of any object X the index of the region where the occurrence resides.

We also use the morphisms φ_i^t, $1 \leq i \leq m$, defined by

$$\varphi_i^t(X_{tar}) = \begin{cases} (X, i) & \text{if } tar = here, \\ (X, j) & \text{if } tar = out, \\ (X, k) & \text{if } tar = in_k, \end{cases}$$

for all $X \in V$, where j is the label of the surrounding region of i.

We construct now an E(rc)T0L system (or an E(cfc)T0L system, respectively) $\Omega = (G, L')$ simulating the computations of Π.

First we construct G. Let $G = (\Sigma, T, H, w')$, where $\Sigma = \{(X, i) \mid X \in V, 1 \leq i \leq m\}$, $T = \Sigma - \{(\#, i) \mid 1 \leq i \leq m\}$ and $w' = \varphi_1(w_1) \cdots \varphi_m(w_m)$.

Each table $h_{i, p_j} \in H$, $1 \leq i \leq m$, $1 \leq j \leq k$, is constructed in the following way:

- for each $X \in V$, if $X \rightarrow \alpha|_{p_j} \in R_i$, for some $p_j \in P$, then the rule $(X, i) \rightarrow \varphi_i^t(\alpha)$ is added to the table h_i. Otherwise, if X is not present as the left hand side of any rule in R_i, then the rule $(X, i) \rightarrow (X, i)$ is added to the table h_i;
- for each $X \in V$ and $1 \leq l \leq m$, $l \neq i$, the rule $(X, l) \rightarrow (X, l)$ is added to the table h_i.

Notice that H has mk tables and each one of them is complete.

Finally we construct L'. To this aim we define the finite substitution φ' by $\varphi'(p_j) = \{t_{(i, p_j)} \mid 1 \leq i \leq m\}$ for each $1 \leq j \leq k$. We also define the nondeterministic finite state automaton $A = (Q, V_A, s_0, F, \delta)$, where $Q = \{0, 1, \ldots, m\}$, $V_A = \{t_{(i, p_j)} \mid 1 \leq i \leq m, 1 \leq j \leq k\}$, $s_0 = 0$, $F = Q$ and δ is defined by $\delta(0, t_{(1, p_j)}) = 1$, $\delta(i_1, t_{(i_1, p_j)}) = \{i_2 \mid 1 \leq i_2 \leq m$, and region i_1 is adjacent to

region i_2 in μ} for every $1 \le j \le k$ and $1 \le i_1 \le m$. Without loss of generality, we assume 1 to be the label of the skin membrane of Π.

Now, $L' = \varphi'(L) \cap L(A)$ is regular (context-free, respectively) since regular (context-free, respectively) languages are closed under intersection with regular languages, see e.g. [10].

The underlying idea of the proof is the following.

Each table $t_{(i,p_j)}$ of G with $1 \le i \le m, 1 \le j \le k$, simulates the rewriting in parallel of the objects present in region i of Π, by using rules activated by the promoter p_j. All the objects present in the same region that cannot be rewritten by any active rule, as well as those present in the other regions of the system, are left unchanged by the application of the table.

The language $\varphi'(L)$ is used to pass from one table to another, in the way described by the strings of promoters present in the control program L. More specifically, if the string $w = p_{j_1} \cdots p_{j_l}$ is present in L, then $\varphi'(L)$ contains all the strings of the set $S_w = \{t_{(i_1,p_{j_1})}, \ldots, t_{(i_l,p_{j_l})} \mid i_1, \ldots, i_l \in \{1, \ldots, m\}\}$. In this way, each computation of Π starting with the control string $w = p_{j_1} \cdots p_{j_l}$ can be simulated in G by applying the tables following the order of an appropriate string in S_w. On the other hand, not every string in the set S_w simulates a correct computation in Π starting with the control string $p_{j_1} \cdots p_{j_l}$. In fact, the control string in Π can only move through adjacent regions – this has to be "encoded" in the way that the passage from one table of G to another one is done. For this reason the appropriate regular (context-free, respectively) language L' that controls G is obtained by intersecting the language $\varphi'(L)$ with the regular language $L(A)$.

From the above explanation it follows that each string in $L(\Omega)$ contains pairs $(object, region)$ corresponding to the objects present in the halting configurations of successful computations of Π. In order to get the exact contents of the output region of Π, we apply to $L(\Omega)$ the morphism φ_o, defined by:

$$\varphi_o((X,i)) = \begin{cases} X & \text{if } i = i_0, \\ \lambda & \text{otherwise.} \end{cases}$$

Since the family $E(rc)T0L$ (or the family $E(cfc)T0L$, respectively) is clearly closed under arbitrary morphisms, it follows that $N_{\#}^{(1)}(\Pi)$ belongs to the family $NE(rc)T0L$ (or to the family $NE(cfc)T0L$, respectively). Thus the theorem holds.

From Theorems 6 and 7 we obtain

Corollary 7.a.

$$(pro)N_{\#}^{(1)}P_*(ncoo, REG) = NE(rc)T0L = NET0L.$$
$$(pro)N_{\#}^{(1)}P_*(ncoo, CF) = NE(cfc)T0L.$$

On the other hand, if SC P systems collect the result in the standard way, then one gets the following results.

Theorem 8.

$$(pro)N^{(1)}P_2(ncoo, REG) \supseteq N(rc)T0L = NET0L.$$

$$(pro)N^{(1)}P_2(ncoo, CF) \supseteq N(cfc)T0L.$$

Proof. In the proof of Theorem 6 the special symbol $\#$ is only used to check if any nonterminal of G is still present when the computation of Π halts. Therefore this checking can be avoided during the simulation of a (rc)T0L system (or a (cfc)T0L system, respectively). Hence the theorem holds.

Analogously, note that in Theorem 7 the set of nonterminals used by the ET0L system constructed in the proof contains only the special object $\#$ included in the alphabet of the corresponding SC P system Π. Therefore if Π collects the output in the standard mode (i.e., it does not use $\#$), then one gets the following results.

Theorem 9.

$$(pro)N^{(1)}P_*(ncoo, REG) \subseteq N(rc)T0L = NET0L.$$

$$(pro)N^{(1)}P_*(ncoo, CF) \subseteq N(cfc)T0L.$$

Theorems 8 and 9 yield the following corollary.

Corollary 9.b.

$$N(rc)T0L = (pro)N^{(1)}P_*(ncoo, REG) = NET0L.$$
$$N(cfc)T0L = (pro)N^{(1)}P_*(ncoo, CF).$$

6   Fully-Promoted SC P Systems: Universality

If SC P systems use arbitrary regular control programs, and only one catalyst, then they generate the family of recursively enumerable sets of natural numbers.

In [3], P systems using two catalysts and two membranes have been proved to be universal. This proof can also be applied for non fully-promoted SC P systems to obtain the following universality result.

Corollary 9.c. $N_\#^{(1)}P_2(cat_2, \{\{\lambda\}\}) = NRE.$

In case of fully-promoted SC P systems, the computational universality can be obtained using arbitrary regular control programs and catalytic rules with only one catalyst.

Theorem 10. $(pro)N_\#^{(1)}P_2(cat_1, REG) = NRE.$

Proof. The inclusion in NRE follows from Church-Turing thesis. The opposite inclusion can be proved by simulating regularly controlled grammars with appearance checking, as follows.

Given a regularly controlled grammar $G = (N, T, S, P, K, F)$ with appearance checking , we construct $\Pi \in (pro)P_2^{(1)}(cat_1, REG)$, collecting the output in the # way, that simulates G. Let

$$\Pi = (V, C, P', L, \mu, w_1, w_2, R_1, R_2, i_0),$$

where:

- $V = N \cup T \cup \{c, Z\}$,
- $C = \{c\}$,
- $P' = lab(P) \cup \{d, d'\}, d, d' \notin lab(P)$,
- $L = \phi(K)dd'$ with non-erasing morphism ϕ defined by $\phi(p) = dp$ for each $p \in lab(P)$,
- $\mu = [_1 \ [_2 \ ]_2 \ ]_1$,
- $w_1 = \lambda; w_2 = SZc$,
- $R_1 = \emptyset$,
- $R_2 = \{cA \rightarrow c\alpha|_p \mid p : A \rightarrow \alpha \in P\} \cup \{cZ \rightarrow c\#|_p \mid p \notin F\}$ $\cup \{Z \rightarrow Z_{out}|_{d'}\}$,
- $i_0 = 2$.

We show that Π simulates the derivations of G. Note that, by definition, for every $p_{i_1} \cdots p_{i_k} \in K$, we have $dp_{i_1} \cdots dp_{i_k} dd' \in L$. The promoters d are dummies, they are only used to let the control string to enter and exit region 2, passing in this way from a promoter p_{i_j} as the current head to the promotor $p_{i_{j+1}}$, for $1 \leq j \leq k - 1$. The derivations of G are simulated by the execution of rules from R_2.

Notice that, because of the catalyst c that inhibits the parallelism, at most one rule is executed in region 2 when the control string resides in that region. If a rule cannot be applied and the label of the corresponding production is not in F, then the computation is unsuccessful (# is produced by applying the rule $cZ \rightarrow c\#$ that is activated by any promoter $p \in (lab(P) - F)$) and this is correct since the simulated derivation in G cannot be continued. On the other hand, if a rule cannot be applied and the label of the corresponding production is in F (so the production has to be used in the appearance checking mode), then no rule is applied in region 2, the control string leaves the region and the computation continues. The last promoter d' present for any control string in L is used to move, at the end of the computation, the symbol Z into region 1. It should be clear from the above description that $N_\#^{(1)}(\Pi)$ is exactly the length set of $L(G)$. Thus the theorem holds.

We conclude this section by presenting some preliminary results concerning the class of non fully-promoted SC P systems.

By definition, it is clear that

Lemma 3.

$$(pro)N_\#^{(i)}P_m(\alpha, FL) \subseteq N_\#^{(i)}P_m(\alpha, FL),$$

$$(pro)N^{(i)}P_m(\alpha, FL) \subseteq N^{(i)}P_m(\alpha, FL),$$

for $\alpha \in \{ncoo, coo\} \cup \{cat_k \mid k \geq 1\}$, FL a family of languages, and $i \in \{1, 2\}$.

	RE	**CF**	**REG**	**FIN**
ncoo	NRE	$NE(cfc)T0L$	$NET0L$	$NFIN$
$cat_i, i \geq 1$	NRE	NRE	NRE	$NFIN$
coo	NRE	NRE	NRE	$NFIN$

Table 1. Computational power of fully-promoted SC P systems operating in mode (1) and collecting the result in the # way. Rows specify the types of evolution rules, and the columns specify the types of control programs.

It is easy to notice that systems from $P_1(ncoo, FIN)$ can generate infinite sets of numbers when operating in mode (1) and collecting the result in the standard way. This observation and Theorem 5 yield the following result.

Theorem 11.

$$(pro)N_{\#}^{(1)}P_*(\alpha, FIN) = (pro)N^{(1)}P_*(\alpha, FIN) \subset N^{(1)}P_*(\alpha, FIN),$$

for $\alpha \in \{ncoo, coo\} \cup \{cat_k \mid k \geq 1\}$.

On the other hand, if one can use any RE language as the control program, then both classes of SC P systems have the same computational power. In particular, from Theorem 4 and Corollary 3, one gets the following result.

Theorem 12.

$$(pro)N^{(1)}P_m(\alpha, RE) = N^{(1)}P_m(\alpha, RE) = NRE,$$

for $\alpha \in \{ncoo, coo\} \cup \{cat_k \mid k \geq 1\}$.

7   Concluding Remarks and Open Problems

We have introduced and investigated SC P systems where the computations are driven by control strings (present in their skin region at the beginning of computations). We have mainly investigated fully-promoted SC P systems, where all the rules are promoted (hence controlled by the control strings). Most of the results proved in this paper concern systems operating in mode (1), although this is just a matter of convenience, because we have proved the equivalence between both operating modes (under some conditions).

Table 1 gives an overview of the results obtained for fully-promoted SC P systems operating in mode (1) and collecting the result in the # way.

The results obtained for fully-promoted SC P systems operating in mode (1) and collecting the result in the standard way are summarized in Table 2.

Several problems, mainly concerning non fully-promoted systems, remain open. Are non-fully promoted SC P systems more powerful than fully-promoted SC P systems? The answer is positive for SC P systems operating in mode (1) and having a finite control program (Theorem 11). We conjecture that the strict inclusion also holds when the control program is regular and the result is collected in the standard way.

	RE	**CF**	**REG**	**FIN**
ncoo	NRE	$N(cfc)T0L$	$N(rc)T0L$	$NFIN$
$cat_i, i \geq 1$	NRE	$\supseteq N(cfc)T0L$	$\supseteq N(rc)T0L$	$NFIN$
coo	NRE	$\supseteq N(cfc)T0L$	$\supseteq N(rc)T0L$	$NFIN$

Table 2. Computational power for fully-promoted SC P systems operating in mode (1) and collecting the result in the standard way. Again, rows specify the types of evolution rules, and the columns specify the types of control programs.

Another open problem is to find a non-trivial upper bound for the generative power of fully-promoted SC P systems operating in mode (1), collecting the result in the standard way, and using cooperative or catalytic rules (see Table 2). We only know that these classes of systems can generate at least the family of length sets of languages from $(rc)T0L$ (if the control program is regular) and from $(cfc)T0L$ (if the control program is context-free). We doubt that these two classes are universal – as a matter of fact they may be incomparable with the classical Chomsky classes.

Finally, another interesting issue to be investigated is having the control programs produced by another bio-inspired generative device (as for instance, another membrane system, or a DNA-based system).

Acknowledgments

References

1. P. Bottoni, C. Martín-Vide, Gh. Păun, G. Rozenberg, Membrane Systems with Promoters/Inhibitors. *Acta Informatica*, 38, 10 (2002), 695–720.
2. J. Dassow, Gh. Păun, *Regulated Rewriting in Formal Language Theory*. Springer-Verlag, Berlin, 1989.
3. R. Freund, L. Kari, M. Oswald, P. Sosik, Computationally Universal P Systems without Priorities: Two Catalysts are Sufficient. *Theoretical Computer Science*, 330, 2 (2005), 251–266.
4. S. Ginsburg, G. Rozenberg, T0L Schemes and Control Sets. *Information and Control*, 27 (1974), 109–125.
5. M. Ionescu, D. Sburlan, On P Systems with Promoters/Inhibitors. *Journal of Universal Computer Science*, 10, 5 (2004), 581–599.
6. Gh. Păun, *Membrane Computing. An Introduction*. Springer-Verlag, Berlin, 2002.
7. Gh. Păun, G. Rozenberg, A Guide to Membrane Computing. *Theoretical Computer Science*, 287, 1 (2002), 73–100.
8. G. Rozenberg, A. Salomaa, *The Mathematical Theory of L Systems*. Academic Press, Inc. Orlando, FL, USA, 1980.

9. G. Rozenberg, A. Salomaa (eds.), *Handbook of Formal Languages.* Springer-Verlag, Berlin, 1997.

10. A. Salomaa, *Formal Languages.* Academic Press, New York, 1973.

11. P systems web page: `http://psystems.disco.unimib.it/`