On P Systems with Global Rules

Andrei Păun
Department of Computer Science, University of Western Ontario
London, Ontario, Canada N6A 5B7
E-mail: `apaun@csd.uwo.ca`

Abstract. We prove here that the P systems with the same rules associated to all membranes (we say that we have global rules) are computationally universal. This is proved for two types of P systems: splicing P systems (in the form considered in [9]) and rewriting P systems (defined in [7]). For the splicing P systems we also try to minimise the "diameter" of the used system, while in the case of rewriting P systems we show that two membranes are enough for universality. This improves the result from [7] where three membranes were used.

1 Introduction

P systems are distributed computing devices inspired from the functioning of alive cells: in the regions defined by a membrane structure one places objects (in the present paper they are considered strings over a given alphabet), and rules for processing these objects. In the basic model, as introduced in [7], with each region one associates a (possibly) different set of rules. A natural variant is to consider the case when the rules are not localized, but a unique set of rules exists (a "global" set), which are used in all regions of the system.

We investigate here the cases when the string-objects are processed by rewriting (by means of context-free rules) and by splicing (a model of the recombination of DNA molecules, [4]).

It is known (see [7], [9]) that in the case when each region has the set of rules, rewriting P systems (with three membranes [7]) and splicing P systems (with two membranes, [9], [3]) are computationally universal, they generate all recursively enumerable languages. At the first sight, working with a unique set of rules is a strong restriction, but as we will prove below, we still get universality. This is true for both rewriting and splicing P systems. Again, in both cases two membranes are sufficient. For splicing P systems we also minimise the diameter of the used rules (the length of strings involved in the splicing rules).

We want to emphasize here a common feature of splicing and rewriting P systems (which is considered here for the first time for rewriting P systems): because we use a terminal alphabet for selecting among the strings sent out of the system those which belong to the language generated, one does not need to account only halting computations and to introduce a (terminal) string in the language only if the computation halts. (The use of halting computation is essential in the case of symbol-objects, when we count the objects leaving the system, and a stop calculation is necessary, but not in the case of string-objects.). This slightly changes the proof techniques, because instead of introducing trap rules of the form $Z \to Z$, for running a "wrong" computation forever, we have to make sure that a nonterminal symbol is introduced in a sentential form so that it will never lead to a terminal string.

2 Basic Definitions

We will first remind the splicing operation introduced in [4] as a formal model of the DNA recombination under the influence of restriction enzymes and ligases.

A *splicing rule* (over an alphabet V) is a string $r = u_1\#u_2\$u_3\#u_4$, where $u_1, u_2, u_3, u_4 \in V^*$ and $\#, \$$ are two special symbols not in V. (V^* is the free monoid generated by the alphabet V under the operation of catenation; the empty string is denoted by λ; the length of $x \in V^*$ is denoted by $|x|$.)

For $x, y, w, z \in V^*$ and r as above we write

$$\begin{aligned}(x, y) \vdash_r (w, z) \quad \text{iff} \quad & x = x_1u_1u_2x_2, \ y = y_1u_3u_4y_2,\\ & w = x_1u_1u_4y_2, \ z = y_1u_3u_2x_2,\\ & \text{for some } x_1, x_2, y_1, y_2 \in V^*.\end{aligned}$$

We say that we splice x, y at the *sites* u_1u_2, u_3u_4. These sites encode the patterns recognized by restrictions enzymes able to cut the DNA sequences between u_1, u_2, respectively between u_3, u_4.

Let us now pass to splicing P systems, the object of our investigation.

We identify a membrane structure with a string of correctly matching parentheses, placed in a unique pair of matching parentheses; each pair of matching parentheses corresponds to a membrane.

A *P system* is a membrane structure with multisets of *objects* placed in its regions and provided with *evolution rules* for these objects. We define here the splicing P systems, in the variant we investigate in this paper (and later we will see also the rewriting P systems).

A *splicing P system with global rules* (of degree $m, m \geq 1$) is a construct

$$\Pi = (V, T, \mu, L_1, \ldots, L_m, R),$$

where:

(i) V is an alphabet; its elements are called *objects*;

(ii) $T \subseteq V$ (the *output* alphabet);

(iv) μ is a membrane structure consisting of m membranes (labeled with $1, 2, \ldots, m$);

(v) $L_i, 1 \leq i \leq m$, are languages over V associated with the regions $1, 2, \ldots, m$ of μ;

(vi) R is a finite set of *evolution rules* given in the following form: $(r = u_1\#u_2\$u_3\#u_4; tar_1, tar_2)$, where $r = u_1\#u_2\$u_3\#u_4$ is a usual splicing rule over V and $tar_1, tar_2 \in \{here, out\} \cup \{in_j \mid 1 \leq j \leq m\}$.

Note that, as usual in H systems, when a string is present in a region of our system, it is assumed to appear in arbitrarily many copies (any number of copies of a DNA molecule can be obtained by amplification).

Any m-tuple $(M_1, \ldots, M_m)$ of languages over V is called a *configuration* of Π. For two configurations $(M_1, \ldots, M_m), (M'_1, \ldots, M'_m)$ of Π we write $(M_1, \ldots, M_m) \Longrightarrow (M'_1, \ldots, M'_m)$ if we can pass from $(M_1, \ldots, M_m)$ to $(M'_1, \ldots, M'_m)$ by applying the splicing rules from R, in parallel, to all possible strings from the corresponding regions, and following the target indications associated with the rules. More specifically, if $x, y \in M_i$ and $(r = u_1\#u_2\$u_3\#u_4, tar_1, tar_2) \in R$ such that we can have $(x, y) \vdash_r (w, z)$, then w and z will go to the regions indicated by tar_1, tar_2, respectively. If $tar_j = here$, then the string remains in M_i, if $tar_j = out$, then the string is moved to the region immediately outside the membrane i (maybe, in this way the string leaves the system), if $tar_j = in_k$, then the string is moved to the region k, providing that this is immediately below; if not, then the rule cannot be applied. Note that the strings x, y are still available in region M_i, because we have supposed that they appear in arbitrarily many copies (an arbitrarily large number of them were spliced, arbitrarily many remain), but if a string w, z is sent out of region i, then no copy of it remains here.

A sequence of transitions between configurations of a given P system Π, starting from the initial configuration $(L_1, \dots, L_m)$, is called a *computation* with respect to Π. The result of a computation consists of all strings over T which are sent out of the system at any time during the computation. We denote by $L(\Pi)$ the language of all strings of this type. We say that $L(\Pi)$ is *generated* by Π.

Note two important facts: if a string leaves the system but it is not terminal, then it is ignored; if a string remains in the system, even if it is terminal, then it does not contribute to the language $L(\Pi)$. It is also worth mentioning that we do not consider here halting computations. We leave the process to continue forever and we just observe it from outside and collect the terminal strings leaving it.

We denote by $SPLG(tar, m, p)$ the family of languages $L(\Pi)$ generated by splicing P systems as above, of degree at most $m, m \geq 1$, and of depth at most $p, p \geq 1$, and with global rules. (The depth of a P system is equal to the height of the tree describing its membrane structure.) If all target indications tar_1, tar_2 in the evolution rules of a P system are of the form *here, out, in*, then we say that Π is of the *i/o type*; the strings produced by splicing and having associated the indication *in* are moved into any lower region immediately below the region where the rule is used. The family of languages generated by P systems with this weaker target indication and of degree at most m and depth at most p is denoted by $SPLG(i/o, m, p)$.

We define the *diameter* of a splicing P system $\Pi = (V, T, \mu, L_1, \dots, L_m, R)$, in a similar way as in the case of extended H systems ([5]), by $dia(\Pi) = (n_1, n_2, n_3, n_4)$, where

$$n_i = \max\{|u_i| \mid u_1\#u_2\$u_3\#u_4 \in R\},\ 1 \leq i \leq 4.$$

We denote the family of languages generated by P systems with the weaker target indication *here, out, in*, of degree at most m and depth at most p, with global rules and with diameter (n_1, n_2, n_3, n_4) by $SPLG(i/o, m, p, (n_1, n_2, n_3, n_4))$.

The following auxiliary result is easy to be proved.

Lemma 2.1 *$SPLG(i/o, m, p, (n_1, n_2, n_3, n_4)) = SPLG(i/o, m, p, (n_3, n_4, n_1, n_2))$, for all $m, p \geq 1$ and $n_i \geq 0$, $1 \leq i \leq 4$.*

Let us now introduce the second model of P systems discussed here:

A rewriting P system is a construct:

$$\Pi = (V, T, \mu, L_1, \dots, L_n, (R_1, \rho_1), \dots, (R_n, \rho_n)),$$

where V is an alphabet, $T \subseteq V$ is the terminal alphabet, μ is a membrane structure, $L_1, \dots, L_n$ are finite languages over V, $R_1, \dots, R_n$ are finite sets of context-free evolution rules, and $\rho_1, \dots, \rho_n$ are partial order relations over $R_1, \dots, R_n$, respectively.

The rules are provided with indications on the target membrane of the produced string, and always we use only context-free rules. Thus, the rules are of the form

$$X \to v(tar),$$

where $tar \in \{here,\ out \cup \{in_j \mid 1 \leq j \leq n\}$, with the obvious meaning: the string produced by using this rule will go to the membrane indicated by *tar*.

The language generated by a system Π is denoted by $L(\Pi)$ and it is defined as follows: we start from an initial configuration of the system and proceed iteratively, by transition steps done by using the rules in parallel, to all strings which can be rewritten, obeying the priority relations relative to the membranes, and collecting the terminal strings sent out of the system during the computation.

Note that each string is processed by one rule only, the parallelism refers here to processing simultaneously all available strings by all applicable rules. So, even if we can apply more than one rule to a string, only one of the possible rules is applied. If we have priorities, then the high priority rule "forbids" the application of a low priority rule.

For examples and properties of rewriting P systems we refer the reader to [7].

We say that such a system has global rules iff $R_1 = R_2 = \ldots = R_n = R$ and a partial order relation ρ over R is given; then we write the system in the following form:

$$\Gamma = (V, \mu, L_1, \ldots, L_n, R, \rho).$$

We denote by $RPL_n(Pri)$ the family of languages generated by rewriting P systems of degree at most $n, n \geq 1$, using priorities; and with $RPLG_n(Pri)$ the family of languages generated by rewriting P systems of degree at most n, with priorities and having global rules.

3 Splicing P Systems

We prove now that the splicing P systems with global rules and only two components are computationally universal:

Theorem 3.1 *$SPLG(i/o, 2, 2) = RE$.*

Proof: Let $G = (N, T, S, P)$ be a type-0 Chomsky grammar and let B be a new symbol. Assume that $N \cup T \cup \{B\} = \{\alpha_1, \ldots, \alpha_n\}$ and that P contains m rules, $u_i \to v_i, 1 \leq i \leq m$. Consider also the rules $u_{m+j} \to v_{m+j}, 1 \leq j \leq n$, for $u_{m+j} = v_{m+j} = \alpha_j$.

We construct the splicing P system (of degree 2):

$$\begin{aligned}
&\Pi{=}(V, T, \mu, L_1, L_2, R = R' \cup R''),\\
&V{=}N \cup T \cup \{B, X, X', Y, Y', Z_1, Z_2, Z_1'\} \cup \{X_i \mid 0 \leq i \leq n+m\} \cup \{Y_i \mid 0 \leq i \leq n+m\},\\
&\mu{=}[_1[_2]_2]_1,\\
&L_1{=}\{Z_1', X'Z_1, Z_1Y'\} \cup \{Z_1Y_i \mid 0 \leq i \leq n+m\} \cup \{X_iv_iZ_1 \mid 1 \leq i \leq n+m\},\\
&L_2{=}\{XSBY, XZ_2, Z_2Y\} \cup \{Z_2Y_i \mid 1 \leq i \leq m+n\} \cup \{X_iZ_2 \mid 0 \leq i \leq m+n-1\},\\
&R'{=}\{(X_iv_i\#Z_1\$X\#;\ in,\ out),\ (\alpha\#Y_i\$Z_1\#Y_{i-1};\ in,\ out) \mid 1 \leq i \leq m+n,\\
&\qquad \alpha \in N \cup T \cup \{B\}\}
\end{aligned}$$

$$\begin{aligned}
&\cup\{(\#Y_0\$Z_1\#Y';\ in,\ out),\ (X_0\#\$X'\#Z_1;\ out,\ here) \mid \alpha \in N \cup T \cup \{B\}\}\\
&\cup\{(\#BY\$Z_1'\#;\ here,\ out),\ (X\#\$\#Z_1';\ out,\ out)\\
R''=&\{(\alpha\#u_iY\$Z_2\#Y_i; out, here), (X_i\#\alpha\$X_{i-1}\#Z_2; here, out) \mid 1 \leq i \leq m+n,\\
&\quad \alpha \in N \cup T \cup \{B\}\}\\
&\cup\{(X'\#\$X\#Z_2;\ here,\ here),\ (\#Y'\$Z_2\#Y;\ out,\ here)\}.
\end{aligned}$$

The idea of this proof is the "rotate-and-simulate" procedure, as used in many proofs in the H systems theory. Here both the simulation of the rules in G and the circular permutation of strings are performed in Π in the same way: a suffix u of the current string is removed and the corresponding string, v, is added in the left end of the string. For a rule $u \to v$ from P, we simulate a derivation step in G; while for a symbol in $N \cup T \cup \{B\}$ we have one symbol "rotation" of the current string ($u = v$).

In the end of a computation we want to have the word in the right permutation, so we have to mark the beginning of the word. For this purpose we use the new letter B which marks the beginning of the sentential form of G. For instance, if Xw_1Bw_2Y is produced in Π, this means that in G we have the word w_2w_1.

Let us see in more detail the work of the system.

The "main" axiom is $XSBY$; we will always process a string of the form Xw_1Bw_2Y (the axiom is of that form). We replace a suffix u_iY of this word with Y_i (in region 2) and the prefix X with X_jv_j (in region 1). Then we will decrease repeatedly the subscripts of Y_i and X_j by one (each time the string is sent to the other membrane). In the end we will replace Y_0 with Y and X_0 with X (this means that $i = j$; so we simulated the production $u_i \to v_i$). In this way we can simulate the productions from P (using the splicings that model $u_i \to v_i,\ 1 \leq i \leq m$) and rotate the string (using the splicings that model $u_i \to v_i,\ m+1 \leq i \leq m+n$).

In the end, in membrane 1, we delete the markers B and Y together (to be sure that we have the right permutation of the word) and finally we delete the marker X and send the string out. Thus, we get $L(G) \subseteq L(\Pi)$.

Now we will prove the converse inclusion. We will start observing that the rules from the set R' can only be applied in membrane 1 and the rules from R'' can only be applied in membrane 2: To prove this, first observe that the rules with a target *in* can only be applied in the outer membrane (membrane 1). Because of this and because the "by products" of a splicing in membrane 1 are expelled from the system, the special characters Z_1, and Z_1' can never reach membrane 2. Because the rest of the rules from R' have always either Z_1 or Z_1' in their pattern, we obtain that the rules from the group R' cannot be applied in membrane 2.

On the other hand, the symbol Z_2 can reach membrane 1 if we have two rules $u_i \to v_i,\ u_j \to v_j$ in G such that $u_i = xu_j,\ x \in \{N \cup T\}^+$. In this case, using the rule $(\alpha\#u_jY\$Z_2\#Y_i;\ out,\ here)$, the string Z_2xY_i will go into membrane 1, but it cannot enter any more splicings there.

One can also notice that we send out many strings from the first membrane, but these strings have at least a marker $Z_1,\ X,\ Y$ (or variants of them with subscripts and/or primed). The only exception is the following splicing rule: $(X\#\$\#Z_1';\ out,\ out)$. Of course, the first string produced by this splicing rule contains the markers X and Z_1', but the second string can be a terminal one. So, only this rule can produce a string in the language of the system.

We prove now that nothing outside of $L(G)$ is produced.

Suppose that we start in the first membrane. The rule $(\alpha\#Y_i\$Z_1\#Y_{i-1}; in, out)$ cannot be applied because we don't have an axiom that has αY_i a subword ($\alpha \in N \cup T$), and later we will have only the words X_jwY_i that satisfy this condition; so this rule will be applied only to these strings. The same discussion applies to the rule $(\alpha\#Y_0\$Z_1\#Y'; in, out)$. The rule ($X_0\#\$X'\#Z$; *out*, *here*) cannot be applied (we don't have X_0 now in the first membrane), and later X_0 can enter this membrane only in the word X_0wY_i, so also this rule cannot produce anything wrong.

If we apply the rule ($\#BY\$Z_1'\#$; *here*, *out*) to the strings $XSBY$, Z_1', then we will produce the string XS in membrane 1. This string (XS) can enter a splicing using the rule ($X_iv_i\#Z_1\$X\#$; *in*, *out*) and then the string X_iv_iS gets in the second membrane. There the subscript of X is decreased by one (using $X_i\#\alpha\$X_{i-1}\#Z_2$; *here*, *out*)) and then the string $X_{i-1}S$ gets in the first membrane. Here this string cannot enter any more splicings. If we apply first the rule $(X\#\$\#Z_1'; out, out)$, then the strings are sent out, and because they contain special symbols, they are not in the language of the system.

Thus, we have to use first a rule ($X_iv_i\#Z\$X\#$; *in*, *out*).

If we start in the second membrane, then the last two rules cannot be applied (we don't have X' or Y' here yet), while the output of the first rule are the same strings that have entered the splicing, or strings which cannot enter any new splicing. The rule $(X_i\#\alpha\$X_{i-1}\#Z_2; here, out)$ cannot be applied to other strings but X_iwY and X_iwY_j. If it is applied to X_iwY, then the string $X_{i-1}wY$ reaches membrane 1 and there cannot enter any more splicings.

Consequently, we have to replace X by X_iv_i in membrane 1 and then to cut u_jY and replace it with Y_j in membrane 2. The string $X_iv_iwY_j$ gets in membrane 1, where the only possibility is to apply the rule ($\alpha\#Y_j\$Z_1\#Y_{j-1}$; *in*, *out*). The string $X_iv_iwY_{j-1}$ gets in membrane 2, where the only possibility is to apply the rule ($X_i\#\alpha\$X_{i-1}\#Z_2$; *here*, *out*), so the string $X_{i-1}v_iwY_{j-1}$ gets in membrane 1. We iterate the process until at least one of the subscripts of X or Y is 0.

If we got X_0, then we decreased the subscript of X in membrane 2 and sent the string $X_0v_iY_{j-i}$ in membrane 1. Here we have two possibilities: $j \neq i$ or $j = i$.

If $j \neq i$ then $j - i \neq 0$, so we can decrease the subscript of Y and send the string in membrane 2. But here the string $X_0v_iwY_{j-i-1}$ can enter no further splicings. Before decreasing the subscript of Y, in membrane 1 we can also apply the splicing rule ($X_0\#\$X'\#Z_1$; *out*, *here*); the string $X'v_iwY_{j-i}$ is sent to membrane 1 and we continue as before. In this case in membrane 2 we can replace X' by X using the rule ($X'\#\$X\#Z_2$; *here*, *here*) and the string Xv_iY_{j-i-1} cannot enter any other splicings so it remains in membrane 2.

If $j = i$, then the only productions from region 1 that can be applied are ($\alpha\#Y_0\$Z_1\#Y'$; *in*, *out*) and ($X_0\#\$X'\#Z_1$; *out*, *here*). If we apply the first one, then the string X_0v_iwY' is sent to membrane 2, here we can only apply the rule that replaces Y' with Y, so the string X_0v_iwY gets in membrane 1. This string will never lead to a terminal string because we cannot delete the left marker (we can replace X_0 with X' but the string remains in this membrane and that marker cannot be deleted).

If we start with the rule ($X_0\#\$X'\#Z_1$; *out*, *here*), then we get the string $X'v_iwY_0$. The only possibility to continue is to apply ($\alpha\#Y_0\$Z_1\#Y'$; *in*, *out*) and the string $X'v_iwY'$ gets in membrane 2. If we don't replace here X' with X, then again the string that gets into membrane 1 cannot lead to a terminal string. So first we replace X' with X (using the rule $X'\#\$X\#Z_2$) and then we replace Y' by Y by using the rule ($\#Y'\$Z_2\#Y$; *out*, *here*). In this way we send the string Xv_iwY in membrane 1 and we can perform another step of rotating the word or simulating the rules from P.

Therefore, the computations in Π correctly simulate rules in G or circularly permute the string. In the end we remove all markers from a string using $(\#BY\$Z_1'\#;\ here,\ out)$ and $(X\#\$\#Z_1';\ out,\ out)$.

In this way we get that $L(\Pi) \subseteq L(G)$. $\square$

In the following we will try to minimise the diameter of the used splicing P systems with global rules (to this aim, one further membrane will be necessary):

Theorem 3.2 $SPLG(i/o, 3, 3, (1, 2, 1, 0)) = SPLG(i/o, 3, 3, (1, 0, 1, 2)) = RE.$

Proof: We will only prove that $SPL(i/o, 3, 3, (1, 2, 1, 0)) = RE$, the other equality follows from Lemma 2.1.

Let $G = (N, T, S, P)$ be a type-0 Chomsky grammar in the Kuroda normal form, (that is P consists of context-free rules of the form $A \to x,\ A \in N,\ x \in (N \cup T)^*,\ |x| \leq 2$, and non-context-free rules of the form $AB \to CD,\ A, B, C, D \in N$) and let B be a new symbol. Assume that $N \cup T \cup \{B\} = \{\alpha_1, \dots, \alpha_n\}$ and that P contains m rules, $u_i \to v_i, 1 \leq i \leq m$. Consider also the rules $u_{m+j} \to v_{m+j}, 1 \leq j \leq n$, for $u_{m+j} = v_{m+j} = \alpha_j$.

We denote by P_1 the set of context-free rules considered above, and with P_2 the rest of the rules. One can see that the rules $u_{m+j} \to v_{m+j}, 1 \leq j \leq n$ are in P_1.

We construct the splicing P system (of degree 3):

$$\begin{aligned}
\Pi &= (V, T, \mu, L_1, L_2, L_3, R = R' \cup R'' \cup R'''),\\
V &= N \cup T \cup \{B, X, X', Y, Z_X, Z_{X'}, Z_Y, Z_\lambda, Z_\lambda'\} \cup \{Y_i, Z_{Y_i} \mid 0 \leq i \leq n+m\}\\
&\cup\ \{X_i, Z_{X_i}, Z_i, Y_i', Z_{Y_i'} \mid 1 \leq i \leq m+n\},\\
\mu &= [_1[_2[_3]_3]_2]_1,\\
L_1 &= \{XSBY, Z_\lambda, Z_\lambda'\} \cup \{Z_{Y_i}Y_i \mid 0 \leq i \leq n+m\} \cup \{Z_{Y_i'}Y_i' \mid 1 \leq i \leq n+m\},\\
L_2 &= \{XZ_X, X'Z_{X'}\} \cup \{X_iv_iZ_i \mid 1 \leq i \leq m+n\} \cup \{X_iZ_{X_i} \mid 1 \leq i \leq m+n-1\},\\
L_3 &= \{Z_YY\},\\
R' &= \{(\#u_iY\$Z_{Y_i}\#; in, out), \mid u_i \to v_i \in P_1\}\\
&\cup\ \{(C\#DY\$Z_{Y_i'}\#; here, out), (\#CY_i'\$Z_{Y_i}\#; in, out) \mid u_i = CD \to v_i \in P_2\}\\
&\cup\ \{(\alpha\#Y_i\$Z_{Y_{i-1}}\#; in, out) \mid 1 \leq i \leq n+m,\ \alpha \in N \cup T \cup \{B\}\}\\
&\cup\ \{(\alpha\#BY\$Z_\lambda\#; here, out),\ (\#Z_\lambda'\$X\#; out, out) \mid \alpha \in T\},\\
R'' &= \{(\alpha\#Z_i\$X\#; out, in) \mid 1 \leq i \leq n+m,\ \alpha \in N \cup T\}\\
&\cup\ \{(X_{i-1}\#Z_{X_{i-1}}\$X_i\#; out, in) \mid 2 \leq i \leq n+m\}\\
&\cup\ \{(X'\#Z_{X'}\$X_1\#; in, in),\ (X\#Z_X\$X'\#; out, in)\},\\
R''' &= \{(\alpha\#Y_0\$Z_Y\#; out, out) \mid \alpha \in N \cup T \cup \{B\}\}.
\end{aligned}$$

This proof closely follows the proof of Theorem 1 from [9], with a special attention paid to the diameter of the splicing rules and also changing the rules to be global.

We will first prove that the rules from the group R' can only be applied in membrane 1, the rules from R'' in membrane 2 and the rules from R''' in membrane 3. We understand by this that if a rule is applied not according to this, then the resulting strings can never "evolve" into terminal strings.

First, one can observe that the symbols Z_{Y_i}, $Z_{Y_i'}$, Z_λ, Z_λ' can only be found in membrane 1 because the axioms that contain them are all in membrane 1, and the splicing rules that can be applied to these axioms are from the set R'. If we take a closer look to the splicing rules from R' we can see that the "garbage" produced by a splicing rule is sent out of membrane 1, while "the good part" is kept in the system, so always, the byproducts containing these special symbols are expelled from the system and can never enter any more splicings. In this way we showed that the rules from R' can only be applied in membrane 1.

Let's look now to the rules from the group R'': it is easy to see that these rules work at one end of a string that enters splicing (because the special symbols Z_i, X, X_i, Z_{X_i}, $Z_{X'}$, X_1, X', Z_X) appear only to one end of a string. With this observation we can conclude that the "by product" strings resulted from splicing in membrane 2 using rules from R'' are sent to membrane 3: XZ_i, $X_iZ_{X_{i-1}}$, $X_1Z_{X'}$, $X'Z_X$. But because the special form of the rules in the group R'' one can see that no rule from this group can be applied in membrane 3 (the only possibility would be a rule applied to the byproducts that were sent in membrane 3 from membrane 2, but it is easy to see that the rules in R'' always check the presence of a symbol on the first position (before the hash symbol), so in this way the product of a rule cannot enter another splicing using the same rule). In the end we conclude that the splicing rules from R'' can only be applied in the membrane 2.

The only rule that is contained in R''' is $(\alpha\#Y_0\$Z_Y\#; out,\ out)$. We will show that this rule cannot lead to terminal strings if it is applied in membranes 1 and 2. In the beginning the symbol Z_Y is only in membrane 3 (contained in the axiom Z_YY), so only here the rule can be applied now. After the rule was used between $w\alpha Y_0$ and Z_YY the strings produced: $w\alpha Y$ and Z_YY_0 are sent to membrane 2. One can notice that the second string can only enter splicings using this rule that produced it. In membrane 2 the rule can be applied between $w\alpha Y_0$ and Z_YY_0, so the produced strings (which are the same as the inputs) are sent to membrane 1. But in membrane 1 the string $w\alpha Y_0$ cannot enter any more splicings (because the string starts with a variant of X and finishes with Y_0), so nothing "bad" is produced.

If the rule (from R''') is applied in membrane 1, then the produced strings will be sent out from the system, and the nonterminal Y_0 will be present in both strings, so nothing is produced.

¿From now on, the proof follows the proof of Lemma 2 from [6]. The sentential forms generated by G are simulated in Π in a circular permutation: Xw_1Bw_2Y, maybe with variants of X, Y, will be present in a region of Π if and only if w_2w_1 is a sentential form of G. Note that we can remove the nonterminal symbol Y only together with B from a string of the form $XwBY$. In this way, we ensure that the string is in the right permutation.

The simulation of rules in P and the rotation are done in the same way. Assume that some string Xwu_iY is present in region 1. We have now two cases: if $u_i \to v_i \in P_1$, then we simulate right away the rule $u_i \to v_i$ with a splicing rule of the form $(\#u_iY\$Z_{Y_i}, in, here)$. If $u_i = CD \to v_i \in P_2$, then the simulation requires two steps: first we replace DY with Y_i' and then we replace CY_i' with Y_i.

So in both cases we replace the suffix u_iY with Y_i, $1 \le i \le m+n$: initially we have here the string $XSBY$. We can perform

$$(Xw|u_iY, Z_{Y_i}|Y_i) \vdash (XwY_i, Z_{Y_i}u_iY) \text{ for } u_i \to v_i \in P_1,$$

or else,

$$(XwC|DY, Z_{Y_i'}|Y_i') \vdash (XwCY_i', Z_{Y_i'}DY) \text{ and } (Xw|CY_i', Z_{Y_i}|Y_i) \vdash (XwY_i, Z_{Y_i}CY_i').$$

The string XwY_i is sent to region 2, the "by products" are sent out and don't enter the language generated by Π because they contain at least a nonterminal. In region 2 we can only perform a splicing of the form $(X|wY_i, X_jv_j|Z_j) \vdash (XZ_j, X_jv_jwY_i)$, for some $1 \leq j \leq n+m$. The string $X_jv_jwY_i$ is sent back to region 1, XZ_j is sent to membrane 1 and cannot enter other splicings. Now, in region 1 the only splicing which can be applied to the string $X_jv_jwY_i$ is $(X_jv_jw|Y_i, Z_{Y_{i-1}}|Y_{i-1}) \vdash (X_jv_jwY_{i-1}, Z_{Y_{i-1}}Y_i)$. The string $X_jv_jwY_{i-1}$ is sent to region 2, while $Z_{Y_{i-1}}Y_{i-1}$ is sent out and don't enter the generated language. In region 2 we now decrease by one the subscript of X_j, using the rule $(X_j|v_jwY_{i-1}, X_{j-1}|Z_{X_{j-1}}) \vdash (X_jZ, X_{j-1}v_jwY_{i-1})$. We iterate this process of decreasing the subscripts until either the subscript of X reaches 1 or the subscript of Y becomes 0.

If at some moment we reach X_1, hence in region 2 we have a string $X_1v_jwY_k$, then we perform $(X_1|v_jwY_k, X'|Z_{X'}) \vdash (X_1Z_{X'}, X'v_jwY_k)$ and $X'v_jwY_k$ is sent to membrane 3. If $k \neq 0$, then nothing can be done, the string is "lost". Otherwise, Y_0 is replaced with Y and the string $X'v_jwY$ is sent to region 2; X' is replaced here by X and the string Xv_jwY is sent to the skin membrane.

If at some moment in region 2 we get a string $X_kv_jwY_0$, for $k \geq 2$, this string cannot be processed in the skin membrane, hence it is "lost". Thus, we can correctly continue only when $i = j$, hence we have passed from Xwu_iY to Xv_iwY; in this way we have either correctly simulated a rule from P or we have circularly permuted the string with one symbol. This is true because we cannot have "illegal" splicings: "by product" strings generated in membrane 1 are sent out, the ones produced in membrane 2 are sent to membrane 3 (none of them containing either Y_0 or Z_Y, hence cannot enter in splicings in membrane 3) and the "garbage" Z_YY_0 from membrane 3 is sent to membrane 2, where it cannot enter any splicing. Because of this fact, we know that when we have Z with a subscript in a word entering a splicing, then that word is a axiom (and one can see that there are not two different axioms containing Z with the same subscript).

The process of simulating a rule or of rotating the string with one symbol can be iterated. Therefore, all derivations in G can be simulated in Π and, conversely, all correct computations in Π correspond to correct derivations in G. Because we collect only terminal strings which leave the system, we have the equality $L(G)) = L(\Pi)$. It is easy to see that the diameter of the P system is $(0, 2, 1, 0)$. $\square$

4 Rewriting P Systems With Global Rules

We now pass to the second model of P systems we discuss here, rewriting P systems. First we improve the main result from [7] about this variant, that is, we prove that two membranes are sufficient for universality. First we consider systems with local rules.

To this aim, we need the definition of matrix grammars with appearance checking in the binary normal form.

A matrix grammar with appearance checking, $G = (N, T, S, M, F)$ is in the binary normal form (Lemma 1.3.7 in [1]) if $N = N_1 \cup N_2 \cup \{S, \dagger\}$, with these three sets disjoint, and the matrices in M are of one 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. $(X \to Y, A \to \dagger)$, with $X, Y \in N_1, A \in N_2$,

4. $(X \to x_1, A \to x_2)$, with $X \in N_1, A \in N_2$, and $x_1, x_2 \in T^*$.

Moreover, there is only one matrix of type 1 and F consists exactly of all rules $A \to \dagger$ appearing in matrices of type 3.

The symbols in N_1 are mainly used to control the use of rules of the form $A \to x$ with $A \in N_2$, while $\dagger$ is a trap symbol; once introduced, it is never removed. A matrix of type 4 is used only once, at the last step of a derivation.

Theorem 4.1 $RPL_2(Pri) = RE$

Proof: Let $G = (N, T, S, M, F)$ be a matrix grammar with appearance checking in the binary normal form. For each matrix of type 4 $(X \to x_1, A \to x_2)$, with $x_1, x_2 \in T^*$, we also introduce the matrix $(X \to X'x_1,\ A \to x_2)$, which is considered of type $4'$; we also add the matrices $(X' \to \lambda)$; X' is a new symbol associated with X. Clearly, the generated language is not changed. We assume the matrices of the types 2, 3, $4'$ labeled in a one-to-one manner with $m_1, \ldots, m_k$.

We construct the following rewriting P system:

$$\begin{aligned}
\Pi &= (V, T, \mu, L_1, L_2, (R_1, \rho_1), (R_2, \rho_2)),\\
V &= N_1 \cup N_2 \cup \{E, Z, \dagger\} \cup T \cup \{X_i, X_i' \mid X \in N_1, 1 \le i \le k\},\\
\mu &= [_1 [_2\ ]_2]_1,\\
L_1 &= \{XAE\}, \text{ where } (S \to SA) \text{ is the initial matrix of } G,\\
L_2 &= \emptyset,\\
R_1 &= \{r_\alpha : \alpha \to \alpha \mid \alpha \in V - T, \alpha \ne E\}\\
&\quad \cup \{r_0 : E \to \lambda(out)\}\\
&\quad \cup \{t_i : X \to Y_i(in) \mid m_i : (X \to Y, A \to x) \text{ is a matrix of type 2}\}\\
&\quad \cup \{t_i : X \to Y_i(in) \mid m_i : (X \to Y, A \to \dagger) \text{ is a matrix of type 3}\}\\
&\quad \cup \{t_i : X \to X_i'x_1(in),\ X_i' \to \lambda \mid m_i : (X \to X'x_1, A \to x_2)\\
&\qquad \text{is a matrix of type } 4'\}\\
&\quad \cup \{Y_i \to Y,\ Y_i' \to Y \mid Y \in N_1, 1 \le i \le k\},\\
\rho_1 &= \{r_\alpha > r_0 \mid \alpha \in V - T, \alpha \ne E\},\\
R_2 &= \{r_i : Y_i \to Y_i,\ r_i' : A \to x(out) \mid m_i : (X \to Y, A \to x)\\
&\qquad \text{is a matrix of types 2 or 4}\}\\
&\quad \cup \{r_i : X_i' \to X_i',\ r_i' : A \to x_2(out) \mid m_i : (X \to X'x_1, A \to x_2)\\
&\qquad \text{is a matrix of type } 4'\}\\
&\quad \cup \{p_i : Y_i \to Y_i',\ p_i' : Y_i' \to Y_i,\ p_i'' : A \to \dagger(out) \mid m_i : (X \to Y, A \to \dagger)\\
&\qquad \text{is a matrix of type 3}\}\\
&\quad \cup \{p_0 : E \to E(out)\},\\
\rho_2 &= \{r_i > r_j',\ r_i > p_j'',\ p_i > r_j',\ p_i' > r_j' \mid i \ne j, \text{ for all possible } i, j\}\\
&\quad \cup \{p_i'' > p_i,\ p_i > p_0,\ r_i > p_0 \mid \text{ for all possible } i\}.
\end{aligned}$$

We will now explain the work of the system. Observe first that the rules $\alpha \to \alpha$ from membrane 1 change nothing, can be used forever, and prevent the use of the rule $E \to \lambda(out)$, which sends the string out of the system. So, we can use the rule $E \to \lambda$ only after all nonterminal symbols have been rewritten into terminal ones.

Let us assume that in membrane 1 we have a string of the form XwE (initially, we have the string $\bar{X}\bar{A}E$). In membrane 1 one chooses the matrix to be simulated, m_i, and one simulates its first rule, $X \to Y$, by introducing Y_i (the subscript i keeps the information about what rule we are simulating); and then the string is sent to membrane 2.

Let us discuss now the case of matrices of type 2: In membrane 2 we can use the rule $r_i : Y_i \to Y_i$ forever. The only way to exit this membrane is by using the rule $A \to x$ appearing in the second position of a matrix of type 2 (we cannot use $p_0 : E \to E(out)$ because r_i has priority over p_0, and we cannot use a rule p''_i because r_i has again priority). Due to the priority relation ρ_2, this matrix should be exactly m_i as specified by the subscript of Y_i (every other rule cannot be applied because then r_i have priority over all r'_j with $j \neq i$). Therefore, we can continue the computation only when the matrix is correctly simulated (we use the rule r'_i).

The process is similar for matrices of type 3: The rules $Y_i \to Y'_i, Y'_i \to Y_i$ can be used forever and we remain in membrane 2. We can quit this membrane either by using a rule $A \to \dagger(out)$ or by using the rule $E \to E(out)$, we cannot use a rule r'_i because $p_i > r'_j$ and $p'_i > r'_j$. In the first case the computation will never lead to a terminal string (we introduced the trap-symbol $\dagger$ that can never be removed). Because of the priority relation, such a rule must be used if the corresponding symbol A appears in the string. If this is not the case, then the rule $Y_i \to Y'_i$ can be used. If we now use the rule $Y'_i \to Y_i$, then we get nothing. If we use the rule $E \to E(out)$, and this is possible because Y_i is no longer present, so the higher priority rule p_i cannot be applied, then we send out a string of the form $Y'_i wE$.

In membrane 1 we replace Y_i or Y'_i by Y, and thus the process of simulating the use of matrices of types 2 and 3 can be iterated.

A slightly different procedure is followed for the matrices of type 4′; they are of the form $m_i : (X \to X'x_1,\ A \to x_2)$. In membrane 1 we use $X \to X'_i x_1(in)$, which already introduces the string x_1, and the string arrives in membrane 2. Again the only way to leave this membrane is by using the associated rule $A \to x_2(out)$. In membrane 1 we have to apply $X'_i \to \lambda$. If no symbol different of E and the terminal symbols is present, then we can apply the rule $E \to \lambda(out)$. Thus, a terminal string is sent out of the system.

Therefore, in the language $L(\Pi)$ we collect exactly the terminal strings generated by the grammar G, that is $L(G) = L(\Pi)$. □

By appropriately modifying the proof of Theorem 4.1, we can now show that rewriting P systems with global rules and only two components are computationally universal:

Theorem 4.2 $RPLG_2(Pri) = RE$

Proof: We use the notations from the previous proof. Starting from the system Π constructed in the proof of theorem 4.1, we construct the following rewriting P system with global rules:

$$\Pi' = (V, T, \mu, L_1, L_2, R = R'_1 \cup R'_2, \rho'_1 \cup \rho'_2),$$

where

$$R'_1 = R_1, \text{ with the rules } Y_i \to Y \text{ and } Y'_i \to Y \text{ changed to}$$

$$\begin{array}{rcl} & & s_i : Y_i \to Y(in) \text{ and } s'_i : Y'_i \to Y(in), \text{ respectively,} \\ \rho'_1 & = & \rho_1 \cup \{t_i > r_j,\ t_i > r'_j,\ t_i > p_j,\ t_i > p'_j,\ t_i > p''_j,\ t_i > p_Y, \\ & & s_i > r_j,\ s_i > r'_j,\ s_i > p_j,\ s_i > p'_j,\ s_i > p''_j,\ s_i > p_Y, \\ & & s'_i > r_j,\ s'_i > r'_j,\ s'_i > p_j,\ s'_i > p'_j,\ s'_i > p''_j,\ s'_i > p_Y \mid \text{for all } i, j, Y\}, \\ R'_2 & = & R_2 \cup \{p_Y : Y \to Y(out)\}, \\ \rho'_2 & = & \rho_2 \cup \{p_Y > r_i,\ p_Y > r'_i,\ p_Y > p''_i \mid i \geq 0\}. \end{array}$$

The idea of the construction is the following: one can see that all the rules from R'_1, with the exception of $E \to \lambda(out)$ and $r_\alpha : \alpha \to \alpha$, have the target indication (in), so they cannot be applied in membrane 2. The additional priorities added to ρ'_1 make sure that no rule from R'_2 will be applied in membrane 1 (the rules from R'_1 have priority and always a rule from R'_1 can be applied: that is r_0) (the only exception to this is the rule p_0 that can be applied in membrane 1, but when it is applied it sends out a string containing E, so that string will not contribute to the language, thus the language is unchanged).

The only change in the rewriting rules was to introduce the target indication (in) to the rules $Y_i \to Y$ and $Y'_i \to Y$, but introducing this we had to add also the rule $p_Y : Y \to Y$ to R'_2. As we can see this doesn't change the language generated by the system, so because the rules R'_1 can only be applied in membrane 1 and the rules from R'_2 in membrane 2 using the previous proof we get that $L(\Pi') = L(G)$, which means that $RE \subseteq RPLG_2(Pri)$. $\square$

5 Final Remarks

We have considered rewriting and splicing P systems with global rules and we have proved that this restriction does not decrease the generative power: characterizations of recursively enumerable languages are obtained also in this case, likewise to the case of systems with local rules (that is, rules associated with each membrane). This partially solves a problem formulated in [8]. The case of P systems of other types (for instance, using symbol-objects) remains to be investigated.

We also need to improve the result about splicing P systems in what concerns the diameter of splicing rules: Theorem 3.2 uses a system with three membranes. What is the smallest diameter in the case od two membranes? If we start the proof of Theorem 3.1 from a type-0 grammar in the Kuroda normal form, we get a system Π such that $dia(\Pi) = (3, 3, 1, 1)$. We believe that this result can be improved.

References

[1] J. Dassow, Gh. Păun, *Regulated Rewriting in Formal Language Theory*, Springer-Verlag, Berlin, 1989.

[2] J. Dassow, Gh. Păun, On the power of membrane computing, *Journal of Universal Computer Science*, **5**, 2 (1999), 33–49.

[3] P. Frisco, Membrane computing based on splicing: improvements, *Pre-proc. Workshop on Multiset Processing*, Curtea de Argeş, Romania, TR 140, CDMTCS, Univ. Auckland, 2000, 100–111.

[4] T. Head, Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviors, *Bull. Math. Biology*, **49** (1987), 737–759.

[5] A. Păun, Controlled H systems of small radius, *Fundamenta Informaticae*, **31**, 2 (1997), 185 – 193.

[6] A. Păun, M. Păun, On the membrane computing based on splicing, *Where Mathematics, Computer Science, Linguistics and Biology Meet* (C. Martin-Vide, V. Mitrana, Eds.), Kluwer, Dordrecht, 2001, 409–422.

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

[8] Gh. Păun, Computing with membranes (P systems): Twenty six research topics, *Auckland University, CDMTCS Report* No 119, 2000 (www.cs.auckland.ac.nz/ CDMTCS).

[9] Gh. Păun, T. Yokomori, *Membrane Computing Based on Splicing, Preliminary Proc. of Fifth Intern. Meeting on DNA Based Computers* (E. Winfree, D. Gifford, eds.), MIT, June 1999, 213–227.

[10] G. Rozenberg, A. Salomaa, eds., *Handbook of Formal Languages*, Springer-Verlag, Berlin, 1997.