Journal of Universal Computer Science, vol. 6, no. 1 (2000), 178-193
submitted: 16/6/99, accepted: 2/11/99, appeared: 28/1/00 © Springer Pub. Co.

Simulating H Systems by P Systems[1]

Gheorghe Păun
(Institute of Mathematics of the Romanian Academy
PO Box 1-764, 70700 Bucureşti, Romania
Email: gpaun@imar.ro.)

Takashi Yokomori
(Department of Mathematics, School of Education
Waseda University, 1-6-1 Nishi-waseda, Shinjuku-ku
Tokyo 169-8050, Japan
Email: yokomori@mn.waseda.ac.jp.)

Abstract: H systems are DNA computing models, based on the operation of splicing. P systems are membrane computing models, where objects can evolve in parallel in a hierarchical membrane structure. In particular, the objects can be strings and the evolution rules can be based on splicing. Both H systems with certain controls on the use of splicing rules and P systems of various types are known to be computationally universal, that is, they characterize the recursively ennumerable languages. So, they are equivalent as the generative power.
The present paper presents a direct simulation of some controlled H systems by splicing P systems. We achieve this goal for three basic regulation mechanisms: H systems with permitting contexts, H systems with forbidding contexts, and communicating distributed H systems. We can say that in this way we get a uniform "implementation" of the three types of H systems in the form of a "computing cell".

Key Words: H systems, splicing rule, universal models of computation
Category: F.1.1.

1 Introduction

H systems and P systems are two different classes of computing devices recently introduced in the very active area of Molecular Computing.

An H system is a grammar-like device, based on the operation of splicing. This operation, first considered in [4], is a model of the recombination operation which takes place among DNA molecules which are cut by restriction enzymes (such that we get sticky ends which match in the Watson-Crick sense) and the fragments are pasted together by ligases. Thus, a *splicing rule* is a quadruple (u_1, u_2, u_3, u_4) of strings, such that u_1, u_2 indicate the pattern recognized by an enzyme and the place where the cut takes place, and u_3, u_4 indicates the pattern where the second enzyme can cut. The prefix of a string cut at u_1u_2 is pasted with the suffix of a string cut at u_3u_4 (see formal definitions below). Starting from a set of strings and a set of splicing rules, by iteratively applying these rules to the current set of strings we get a language. It is known that if the sets of axioms and of rules are finite, then in this way we generate only regular

[1] C. S. Calude and G. Ştefănescu (eds.). *Automata, Logic, and Computability. Special issue dedicated to Professor Sergiu Rudeanu Festschrift.*

languages, but if a "weak" control is imposed to the use of splicing rules, then we get characterizations of recursively enumerable languages. Among the basic restrictions of this type are the *permitting contexts* (a rule has associated two sets of strings and it can be applied only to strings which contain as substrings all the elements of the corresponding set of strings), the *forbidding contexts* (exactly the opposite: the rule can be applied only to strings which contain as substrings no element of the corresponding sets). The same increasing of the power is obtained for many distributed H systems, where several usual H systems cooperate in generating a common language; we consider here only the so-called *communicating distributed* H systems (CD H systems, for short), where the components send to each other strings obtained by splicing and these strings are accepted by destination components only if they pass certain filters (they do not contain symbols from specified sets, that is, we have a sort of forbidding context filtering). Details can be found in the monograph [8].

The P systems are a class of distributed parallel computing devices of a biochemical type introduced in [6], which can be seen as a general computing architecture where various types of objects can be processed by various operations.

In short, in the basic model one considers a *membrane structure* consisting of several cell-membranes which are hierarchically embedded in a main membrane, called the *skin* membrane. The membranes delimit *regions*, where we place *objects*.

The objects evolve according to given *evolution rules*, which are associated with the regions. A rule is applied to objects in the region where it is placed and can modify the objects, send them outside the current membrane or to an inner membrane, and can also dissolve the membrane. When such an action takes place, all the objects of the dissolved membrane remain free in the membrane placed immediately outside, but the evolution rules of the dissolved membrane are lost. The skin membrane is never dissolved. Note that the membranes are both separators and channels of communication.

The application of evolution rules is done in a maximally parallel manner: at each step, all objects which can evolve should evolve.

Starting from an initial configuration and using the evolution rules, we get a *computation*. In the basic model, a computation is considered completed when it halts, no further rule can be applied. There are two possible ways of assigning a result to a computation: by considering the multiplicity of objects present in a designated membrane in a halting configuration, or by concatenating the symbols which leave the system, in the order they are sent out of the skin membrane (if several symbols are expelled at the same time, then any ordering of them is accepted). Thus, in the first case we compute vectors of natural numbers, while in the second case we generate a language.

Many variants are considered in [3], [6], [7], [9], [10], [12]. In most of them, the objects are described by symbols from a given alphabet.

In this paper we are interested in the case when the objects are not "atomic" symbols, but they are described by strings. The case when the evolution rules of such objects are based on the splicing operation was investigated both in [6] and [11]. Characterizations of recursively enumerable languages were obtained for various simple forms of such splicing P systems.

Thus, the two types of mechanisms, controlled or distributed H systems and splicing P systems, are equivalent in power. We give here a direct proof of this equivalence, by "implementing" the mentioned types of H systems as P systems. This is a significant "programming exercise" in the "language of computing cells", showing not only the power, but also the versatility of the computing architecture inherent to P systems.

2 H Systems

Let us first remind the splicing operation as introduced in [4]; we follow the formalism from [8]. (For formal language theory elements we use below we refer to the many monographs in the area, in particular to [14].)

Consider an alphabet V and two symbols $\#, \$$ not in V. A *splicing rule* over V is a string $r = u_1\#u_2\$u_3\#u_4$, where $u_1, u_2, u_3, u_4 \in V^*$ (V^* is the set of all strings over V; the empty string is denoted by λ). For such a rule r and for $x, y, w, z \in V^*$ we define

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

(One cuts the strings x,y in between u_1, u_2 and u_3, u_4, respectively, and one recombines the fragments obtained in this way.) When r is understood, we write $\vdash$ instead of $\vdash_r$. For clarity, we usually indicate by a vertical bar the place of splicing: $(x_1u_1|u_2x_2, y_1u_3|u_4y_2) \vdash (x_1u_1u_4y_2, y_1u_3u_2x_2)$.

A pair $\sigma = (V, R)$, where V is an alphabet and R is a set of splicing rules over V, is called an *H scheme.* With respect to an H scheme $\sigma = (V, R)$ and a language $L \subseteq V^*$ we define

$$\begin{aligned}&\sigma(L) = \{w \in V^* \mid (x,y) \vdash_r (w,z) \text{ or } (x,y) \vdash_r (z,w),\ x,y \in L, r \in R, z \in V^*\},\\ &\sigma^*(L) = \bigcup_{i \geq 0} \sigma^i(L), \text{ for}\\ &\sigma^0(L) = L,\\ &\sigma^{i+1}(L) = \sigma^i(L) \cup \sigma(\sigma^i(L)),\ i \geq 0.\end{aligned}$$

An *extended H system* is a construct $\gamma = (V, T, A, R)$, where V is an alphabet, $T \subseteq V, A \subseteq V^*$, and $R \subseteq V^*\#V^*\$V^*\#V^*$. ($T$ is the *terminal* alphabet, A is the set of *axioms*, and R is the set of *splicing rules.*) When $T = V$, the system is said to be non-extended. The pair $\sigma = (V, R)$ is the *underlying H scheme* of γ.

The language generated by γ is defined by $L(\gamma) = \sigma^*(A) \cap T^*$. (We iterate the splicing operation according to rules in R, starting from strings in A, and we keep only the strings composed of terminal symbols.)

It is known that extended H systems with finite sets of axioms and of splicing rules characterize the regular languages, [2], [13]. This makes necessary to consider *controlled* extended H systems, able to generate non-regular languages.

In an *H system with permitting contexts*, $\gamma = (V, T, A, R)$, the rules from R are of the form $p = (r; D_1, D_2)$, where $r = u_1\#u_2\$u_3\#u_4$ is a usual splicing rule over V and D_1, D_2 are finite sets of strings over V. For $x, y, w, z \in V^*$ we write $(x,y) \vdash_p (w,z)$ only if $(x,y) \vdash_r (w,z)$, all strings from D_1 appear as substrings of x and all strings from D_2 appear as substrings of y.

In an *H system with forbidding contexts* the rules have the same form as above, but $(x, y) \vdash_p (w, z)$ only if $(x, y) \vdash_r (w, z)$, no string from D_1 appears as a substring of x and no string from D_2 appears as a substring of y.

We denote by pEH, fEH the families of languages generated by extended H systems with permitting and with forbidding contexts, respectively.

A *communicating distributed H system* (CD H system) (of degree $n, n \geq 1$) is a construct

$$\Gamma = (V, T, (A_1, R_1, V_1), \ldots, (A_n, R_n, V_n)),$$

where V is an alphabet, $T \subseteq V$, A_i are finite languages over V, R_i are finite sets of splicing rules over V, and $V_i \subseteq V$, $1 \leq i \leq n$.

Each triple $(A_i, R_i, V_i), 1 \leq i \leq n$, is called a *component* of Γ; A_i, R_i, V_i are the set of *axioms*, the set of *splicing rules*, and the *selector* (or *filter*) of the component i, respectively; T is the terminal alphabet of the system.

The pair $\sigma_i = (V, R_i)$ is the underlying H scheme associated with the component i of the system.

An n-tuple $(L_1, \ldots, L_n), L_i \subseteq V^*, 1 \leq i \leq n$, is called a *configuration* of the system; L_i is also called the *contents* of the ith component.

For two configurations $(L_1, \ldots, L_n), (L'_1, \ldots, L'_n)$, we define

$$\begin{aligned}(L_1, \ldots, L_n) \Longrightarrow (L'_1, \ldots, L'_n) \text{ iff} \\ L'_i = \sigma_i^*(L_i) \cup \bigcup_{j=1}^{n} (\sigma_j^*(L_j) \cap V_i^*), \\ \text{for each } i, 1 \leq i \leq n.\end{aligned}$$

In words, the contents of each component is spliced according to the associated set of rules (we pass from L_i to $\sigma_i^*(L_i), 1 \leq i \leq n$), and the result is redistributed among the n components according to the selectors $V_1, \ldots, V_n$; each component also keeps from a step to the next one the strings that were produced by itself by splicing. (This is a difference from the definition in [1] and [8], but it is easy to see from the proofs in [8] that the results about CD H systems are not changed by this change in the definition.)

Because we have imposed no restriction over the alphabets V_i, for example, we did not suppose that they are pairwise disjoint, when a string in $\sigma_j^*(L_j)$ belongs to several languages V_i^*, then copies of this string will be distributed to all components i with this property.

The language generated by Γ is defined by

$$\begin{aligned}L(\Gamma) = \{w \in T^* \mid w \in L_1 \text{ for some } L_1, \ldots, L_n \subseteq V^*, \text{ such} \\ \text{that } (A_1, \ldots, A_n) \Longrightarrow^* (L_1, \ldots, L_n)\}.\end{aligned}$$

That is, the first component of the system is designated as its *master* and the language of Γ is the set of all terminal strings generated (or collected by communications) by the master.

We denote by $cdEH_n$ the family of languages generated by CD (extended) H systems of degree at most $n, n \geq 1$. When n is not specified, the subscript n is removed. By RE we denote the family of recursively enumerable languages.

Proofs of the following equalities can be found in [8] and the references given there:

Theorem 1. $pEH = fEH = cdEH = RE.$

3 Splicing P Systems

We define here the splicing P systems in the restricted form considered in [11].

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. Graphically, a membrane structure is represented by a Venn diagram. To each membrane we uniquely associate a *region*, that delimited by the membrane and the immediately lower membranes, if any.

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

$$\Pi = (V, T, \mu, L_1, \dots, L_m, R_1, \dots, R_m),$$

where:

(i) V is an alphabet;
(ii) $T \subseteq V$ (the *output* alphabet);
(iv) μ is a membrane structure consisting of m membranes (labeled, with $1, 2, \dots, m$);
(v) $L_i, 1 \leq i \leq m$, are languages over V associated with the regions $1, 2, \dots, m$ of μ;
(vi) $R_i, 1 \leq i \leq m$, are finite sets of *evolution rules* associated with the regions $1, 2, \dots, m$ of μ, 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 $tar_1, tar_2 \in \{here, out, in\}$.

Note that, as usual in H systems, if a string is present in a region of our system, then 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 $(M_1, \dots, M_m)$ of languages over V is called a *configuration* of Π. For two configurations $(M_1, \dots, M_m), (M'_1, \dots, M'_m)$ of Π we write $(M_1, \dots, M_m) \Longrightarrow (M'_1, \dots, M'_m)$ if we can pass from $(M_1, \dots, M_m)$ to $(M'_1, \dots, M'_m)$ by applying the splicing rules from each region of μ, in parallel, to all possible strings from the corresponding regions, and following the target indications associated with the rules. More specifically (but not completely formal), if $x, y \in M_i$ and $(r = u_1\#u_2\$u_3\#u_4, tar_1, tar_2) \in R_i$ 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 $tar_j = in$, then the string is moved to any region which is immediately below in membrane i; if no such a region exists, then the rule cannot be applied. If in a given region there are several rules which can be applied, then each of them is applied to an arbitrarily large number of strings. If a string can enter no splicing, then it remains unchanged in its region, but if a string can enter a splicing, then *all* its copies are consumed by splicing. (Biochemically speaking, this amounts to assume that when a reaction can take place, then it is *complete*, we wait until all items which can be processed are actually processed. This is a natural assumption, but it is also very powerful, because by using it we can distinguish between the presence and the absence of a given string in a given place, which is very much similar to – but not at all the same as – the way the multisets are used in proving that H systems with multiplicities associated with strings are computationally universal, see [8].) Similarly, if a string is produced by splicing and it is sent out of a region, then no copy of it remains in the region where it is produced.

A sequence of transitions between configurations of a given P system Π, starting from the initial configuration $(L_1, \ldots, L_m)$, 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 which leave it.

We denote by SPL the family of languages $L(\Pi)$ generated by splicing P systems as above.

Proofs of the following equality (even with restrictions on the number of membranes and their arrangement: linear tree or star tree) can be found in [11].

Theorem 2. $SPL = RE$.

4 Simulating Permitting H Systems by Splicing P Systems

We now pass to proving the three simulation results we have announced above. We start with the case which turned out to be easier to solve:

Theorem 3. *Each permitting H system can be simulated by a splicing P system.*

Proof. Consider a permitting H system $\gamma = (V, T, A, R)$, with

$$\begin{aligned} R = \{ & r_i = (u_{i,1}\#u_{i,2}\$u_{i,3}\#u_{i,4}; D_{i,1}, D_{i,2}) \mid \\ & D_{i,1} = \{s_{i,1}, \ldots, s_{i,p_i}\}, D_{i,2} = \{t_{i,1}, \ldots, t_{i,q_i}\}, \\ & 1 \leq i \leq n, u_{i,1}, u_{i,2}, u_{i,3}, u_{i,4} \in V^*, s_{i,j} \in V^*, 1 \leq j \leq p_i, \\ & t_{i,j} \in V^*, 1 \leq j \leq q_i, \text{ and } p_i, q_i \geq 1\}. \end{aligned}$$

(Because the empty string, λ, is a substring of any string, if it appears in a set $D_{i,1}, D_{i,2}$, then it imposes no restriction on the use of the rule. Thus, if one of $D_{i,1}, D_{i,2}$ above would be empty, then by introducing the permitting string λ we can set up $D_{i,1} = \{\lambda\}$ or $D_{i,2} = \{\lambda\}$, which does not change the generated language. In this way, it was possible to assume that each of p_i, q_i are greater than or equal to one.)

We construct a P system Π as follows. Instead of a completely formal construction, we present its idea and only parts of the system are given in full details.

The total alphabet of Π is

$$V \cup \{c_1, c_2, c_1', c_2', d_1, d_2, g, g'\} \cup \{e_{i,1}, e_{i,2}, f_{i,1}, f_{i,2} \mid 1 \leq i \leq n\},$$

the terminal alphabet is T, and the membrane structure has the shape suggested in Figure 1.

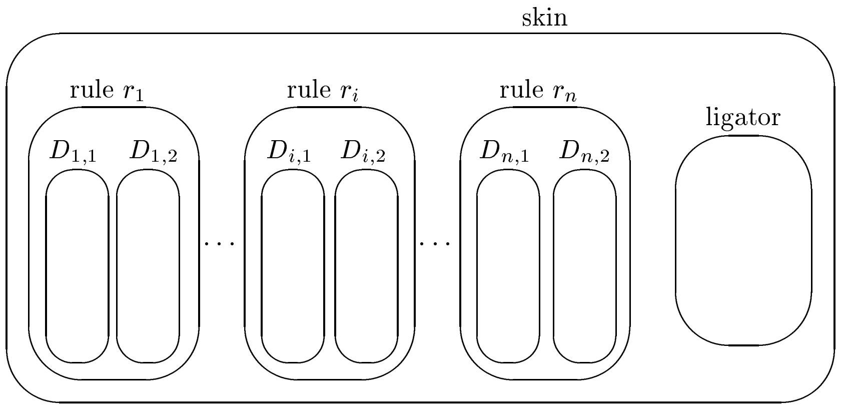

Figure 1. The shape of the membrane structure of the splicing P system from the proof of Theorem 3.

In the skin membrane we place $n+1$ membranes, one associated with each rule in R, and a "ligator". In the membrane associated with r_i we place two membranes, one associated with the set $D_{i,1}$ and one with the set $D_{i,2}$, of the strings which regulate the application of the splicing rule $u_{i,1}\#u_{i,2}\$u_{i,3}\#u_{i,4}$ from r_i. In these two membranes we introduce strings which come from the skin membrane and we check whether or not they contain all the substrings indicated by $D_{i,1}, D_{i,2}$, respectively. If this is the case, then, in the left membrane we also cut the string at the pattern $u_{i,1}u_{i,2}$ and we mark the ends of the fragments in such a way to know that this operation was done, while in the right membrane we cut the string at the pattern $u_{i,3}u_{i,4}$ and also mark the ends of the string, in a way different from the marking in the left membrane. Thus, we know which is the first term of the splicing to be done and which is the second term. The fragments produced in this way are sent out to the skin membrane and from here they are sent to the "ligator"; here the ligation is performed, the resulting strings are sent back to the skin membrane and the process can be iterated.

Initially, in the skin membrane we introduce the language

$$\{c_1wc_2 \mid w \in A\} \cup \{gc_2, g'\} \cup \{ge_{i,1}, gf_{i,1}, e_{i,2}g, f_{i,2}g \mid 1 \leq i \leq n\}.$$

Therefore, we start with the strings in A marked at the ends with the new symbols c_1, c_2; these symbols or primed variants of them will mark the strings during all their path through the system.

The rules present in the skin membrane are:

$$(\#c_2\$g\#c_2; in, here)$$

(we send to any of the immediately lower membranes all the strings present in the skin membrane and ended with c_2; note that from the "ligator" each such string can be sent back, which means that after two steps we can dispose again of all strings which were initially placed in the skin membrane),

$(\#c_2\$g\#; here, here)$,
$(c_1\#\$\#g'; here, out)$

(using these rules we can remove the non-terminal symbols c_1, c_2 and send the remaining string out of the system; if it is a terminal string, then it is an element of the language generated by Π, otherwise the string is "lost"),

$(\#e_{i,1}\$g\#e_{i,1}; in, here)$,
$(\#f_{i,1}\$g\#f_{i,1}; in, here)$,
$(e_{i,2}\#\$e_{i,2}\#g; here, in)$,
$(f_{i,2}\#\$f_{i,2}\#g; here, in)$, for all $1 \leq i \leq n$

(by these rules we send to the "ligator" the strings prepared by the membranes associated with the splicing rules; if these strings arrive again in membranes associated with rules in R, then they are immediately sent back to the skin membrane).

In the membrane associated with rule $r_i, 1 \leq i \leq n$, we introduce the initial strings $\{gc_2, gc_2', c_1'g\}$, as well as the following rules:

$(\#c_2\$g\#c_2; in, here)$

(we send the strings received from the skin membrane to any of the two immediately inner membranes),

$(\#c_2'\$g\#c_2'; out, here)$,
$(c_1'\#\$c_1'\#g; here, out)$

(the strings prepared by the membranes associated with $D_{i,1}$ and $D_{i,2}$ are sent out to the skin membrane).

The membrane associated with $D_{i,1}$ has inside $2p_i+1$ membranes arranged hierarchically, in a linear mode, as indicated in Figure 2. For each string $s_{i,j}$ we have two membranes. In the first one we initially place the strings $\{d_1d_2, gd_2, gc_2', c_1'g\}$ and the rules

$(s_{i,j}\#\$d_1\#d_2; in, in)$

(if a string $c_1x_1s_{i,j}x_2c_2$ arives here, then we can splice $(c_1x_1s_{i,j}|x_2c_2, d_1|d_2) \vdash (c_1x_1s_{i,j}d_2, d_1x_2c_2)$, and both the obtained strings are sent to the lower membrane),

$(\#d_2\$g\#d_2; here, here)$

(this rule just reproduce the axiom d_1d_2, to be used at subsequent steps),

$(\#c_2'\$g\#c_2'; out, here)$,
$(c_1'\#\$c_1'\#g; here, out)$

(the strings prepared by the innermost membranes are sent up).

In the second membrane associated with $s_{i,j}$ we place no axiom, but only the rules

$(\#d_2\$d_1\#; in, here)$

(the strings $c_1x_1s_{i,j}d_2, d_1x_2c_2$ are pasted together and the string $c_1x_1s_{i,j}x_2c_2$ goes to the next membrane, for checking the presence of the substring $s_{i,j+1}$),

$(\#c_2'\$g\#c_2'; out, here)$,
$(c_1'\#\$c_1'\#g; here, out)$

(the strings prepared by the innermost membranes are sent up).

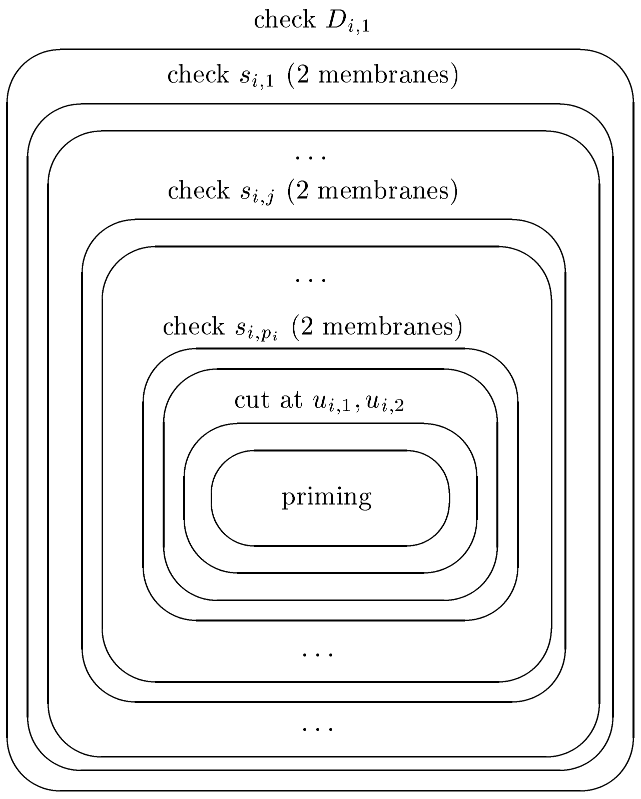
check $D_{i,1}$
check $s_{i,1}$ (2 membranes)
...
check $s_{i,j}$ (2 membranes)
...
check s_{i,p_i} (2 membranes)
cut at $u_{i,1}, u_{i,2}$
priming
...
...

Figure 2. The membrane sub-structure associated with $D_{i,1}$.

Note that if a string does not contain a substring $s_{i,j}$, then it remains forever in the first membrane associated with $s_{i,j}$, because it can enter no splicing which sends it further. Thus, if a string arrives in the second central membrane, that around the innermost one, then we know that all conditions imposed by $D_{i,1}$ are fulfilled. In this membrane we initially introduce the strings $\{e_{i,1}e_{i,2}, gc_2', c_1'g\}$ and the rules

$(u_{i,1}\#u_{i,2}\$e_{i,2}\#e_{i,1}; in, in)$

(from a string $c_1x_1u_{i,1}u_{i,2}x_2c_2$ we pass to the strings $c_1x_1u_{i,1}e_{i,1}, e_{i,2}u_{i,2}x_2c_2$, which are sent to the innermost membrane),

$(e_{i,1}\#e_{i,2}\$e_{i,1}\#e_{i,2}; here, here)$

(for reproducing the axiom $e_{i,1}e_{i,2}$),

$(\#c_2'\$g\#c_2'; out, here)$,
$(c_1'\#\$c_1'\#g; here, out)$

(the strings with the markers c_1, c_2 primed in the innermost membrane are sent out).

In the central membrane we put the initial strings $\{gc_2', c_1'g\}$ and the rules

$(\#c_2\$g\#c_2'; out, here)$,
$(c_1\#\$c_1'\#g; here, out)$

(we change c_1, c_2 with c_1', c_2', respectively, and we send the obtained strings to the upper membrane),

$(g\#c_2'\$g\#c_2'; here, here)$,
$(c_1'\#g\$c_1'\#g; here, here)$

(the axioms are reproduced).

The membrane sub-structure associated with the set $D_{i,2}$ has exactly the same contents as that associated with $D_{i,1}$, but we now check the presence of the substrings $t_{i,j}, 1 \leq j \leq q_i$, and, in the successful case, we use the markers $f_{i,1}, f_{i,2}$ instead of $e_{i,1}, e_{i,2}$, respectively. Using these markes, the strings are sent from the skin membrane to the "ligator".

The "ligator" is a structure of the form $[_1[_2[_3\ ]_3]_2]_1$ (we have labeled the three membranes for reference below).

In membrane 1 we introduce the rules

$(u_{i,1}\#e_{i,1}\$f_{i,2}\#u_{i,4}; in, here)$
$(u_{i,3}\#f_{i,1}\$e_{i,2}\#u_{i,2}; in, here)$

(we ligate the fragments prepared in the membrane sub-structure associated with the rule r_i; the resulting strings are sent to membrane 2, where c_2' will be replaced with c_2),

$(\#c_2\$g\#c_2; out, here)$

(the string coming from the inner membrane is sent out to the skin membrane).

In membrane 2 we introduce the string $\{gc_2\}$ and the rules

$(\#c_2'\$g\#c_2; in, here)$,
$(\#c_2\$g\#c_2; out, here)$,

while in membrane 3 we introduce the string $\{c_1g\}$ and the rules

$(c_1'\#\$c_1\#g; here, out)$,
$(c_1\#g\$c_1\#g; here, here)$

(the marker c_1' is replaced by c_1, the initial string is reproduced).

From the previous explanations one can see that the system Π simulates, indeed, the work of γ: all terminal strings produced by γ are also produced by Π (and sent out of the system), while nothing else than the terminal strings produced by γ can be produced by Π. That is, $L(\gamma) = L(\Pi)$. □

We stress the fact that the interest of the previous theorem is not the fact that $pEH \subseteq SPL$, as simply concluded at the end of the proof, but the construction itself from the proof, and the way the P system behaves. We may say that we have encoded in the form of a splicing P system the algorithm itself of functioning of an extended H system with permitting contexts.

5 Simulating Forbidding H Systems by Splicing P Systems

Somewhat unexpected (because the forbidding context H systems are just the dual of permitting context H systems, so the proofs of their computational universality are quite similar, see [8]), the simulation of an H system with forbidding contexts is more difficult than the simulation of a system with permitting contexts. Still, this can be done:

Theorem 4. *Each extended H system with forbidding contexts can be simulated by a splicing P system.*

Proof. Consider an H system $\gamma = (V, T, A, R)$ with forbidding contexts, with the set R exactly as in the previous proof:

$$\begin{aligned} R = \{ & r_i = (u_{i,1}\#u_{i,2}\$u_{i,3}\#u_{i,4}; D_{i,1}, D_{i,2}) \mid \\ & D_{i,1} = \{s_{i,1}, \ldots, s_{i,p_i}\}, D_{i,2} = \{t_{i,1}, \ldots, t_{i,q_i}\}, \\ & 1 \leq i \leq n, u_{i,1}, u_{i,2}, u_{i,3}, u_{i,4} \in V^*, s_{i,j} \in V^*, 1 \leq j \leq p_i, \\ & t_{i,j} \in V^*, 1 \leq j \leq q_i, \text{ and } p_i, q_i \geq 1\}. \end{aligned}$$

We construct a splicing P system Π with the membrane structure of the same shape as that in Figure 1, with the same strings and rules in the skin membrane and in the "ligator" membrane sub-structure, but with differences in the membrane sub-structures associated with the rules r_i in R.

For each rule r_i we consider again a membrane which contains two membranes, one associated with $D_{i,1}$ and one with $D_{i,2}$. In these membranes we place sub-structures which check the non-appearance of strings in $D_{i,j}, j = 1, 2$, as substrings of strings of the form c_1wc_2 received from the skin membrane.

Consider the case of $D_{i,1}$. For each string $s_{i,j}, 1 \leq j \leq p_i$, we consider three membranes – let us label them by $(j,1), (j,2), (j,3)$ – in the order $[_{(j,1)}[_{(j,2)}[_{(j,3)}\ ]_{(j,3)}]_{(j,2)}]_{(j,1)}$. All these membranes are again arranged in a linear tree manner.

In membrane $(j,1)$ we place the strings $\{gg, fc_2, \bar{f}\bar{c}_2\}$ and the rules:

$(s_{i,j}\#\$g\#g; here, here)$,
$(f\#c_2\$\bar{f}\#\bar{c}_2; here, here)$,
$(\alpha\#c_2\$\bar{f}\#c_2; in, in)$, for all $\alpha \in V$,
$(f\#\bar{c}_2\$\bar{f}\#c_2; here, here)$.

(Suppose that a string c_1wc_2 arrives in membrane $(j,1)$ when we have here the strings $fc_2, \bar{f}\bar{c}_2$. This is the case at the beginning of the computation, because from the skin membrane to membrane $(1,1)$ we make two steps, exactly the time for the rules $(f\#c_2\$\bar{f}\#\bar{c}_2; here, here), (f\#\bar{c}_2\$\bar{f}\#c_2; here, here)$ to be applied. Because $\bar{f}c_2$ is not present, the string c_1wc_2 can be spliced at most with the rule $(s_{i,j}\#\$g\#g; here, here)$. If this rule is applied, then we get two strings of the form c_1w_1g, gw_2c_2 which will never lose the non-terminal symbol g. If w does not contain the substring $s_{i,j}$, then the string c_1wc_2 has to wait. At the same time, the rule $(f\#c_2\$\bar{f}\#\bar{c}_2; here, here)$ is applied, producing the string $\bar{f}c_2$. Using this string, at the next step c_1wc_2 is passed unchanged to membrane $(j,2)$.

In membrane $(j,2)$ we introduce the string $\{gc_2\}$ and the rule

$(\#c_2\$g\#c_2; in, here)$

(the strings which end with c_2 are passed unchanged to membrane $(j, 3)$), where we introduce the string $\{c_1g\}$ and the rule

$(c_1\#\$c_1\#g; here, in)$

(the strings starting with c_1 are passed unchanged to the next membrane, of the type $(j+1, 1)$; note that the possible strings gw_2c_2 can be sent to membrane $(j, 2)$, but they cannot go further).

Consequently, the string c_1wc_2 "survives" if and only if it does not contain the substring $s_{i,j}$. It is important to note that in the positive case, that is when c_1wc_2 is passed to membrane $(j+1, 1)$, the operations in membranes $(j, 1), (j, 2), (j, 3)$ take four steps.

We do not mention here also the rules which have only the role of passing unchanged from a step to the next one the strings initially placed in membranes. Such details are similar to those in the previous proof.

When we have checked all strings in $D_{i,1}$ and none of them is a substring of w, the string c_1wc_2 arrives in the second innermost membrane (again like in the previous proof, we have two further membranes in the center of the sub-structure associated with $D_{i,1}$; in these membranes we cut the string at the pattern $u_{i,1}u_{i,2}$ and mark accordingly the ends of the fragments). Here we introduce the strings $\{e_{i,1}e_{i,2}, gc_2'', c_1''g\}$ and the following rules:

$(u_{i,1}\#u_{i,2}\$e_{i,1}\#e_{i,2}; in, in)$

(we cut the string and we send the fragments to the innermost membrane, where the markers c_1, c_2 are replaced by c_1', c_2', respectively, and the strings are sent out again),

$(\#c_2'\$g\#c_2''; here, here)$,
$(\#c_2''\$g\#c_2'; out, here)$,
$(c_1'\#\$c_1''\#g; here, here)$,
$(c_1''\#\$c_1'\#g; here, out)$.

(the strings are sent out, in the form $c_1'w_1u_{i,1}e_{i,1}, e_{i,2}u_2w_2c_2'$).

In the innermost membrane, we introduce the strings $\{gc_2', c_1'g\}$ and the rules:

$(\#c_2\$g\#c_2'; out, here)$,
$(c_1\#\$c_1'\#g; here, out)$.

Again, we have left to the reader the task of completing the sets of rules with rules for passing the axioms from a step to the next one.

Note that we need four steps from the moment when we get the string c_1wc_2 until producing the strings $c_1'w_1u_{i,1}e_{i,1}, e_{i,2}u_{i,2}w_2c_2'$.

In all membranes placed above the two innermost membranes, including the membrane associated with the rule r_i, we also introduce the strings $\{gc_2'', c_1''g\}$, as well as the folowing four rules:

$(\#c_2'\$g\#c_2''; here, here)$,
$(\#c_2''\$g\#c_2'; out, here)$,
$(c_1'\#\$c_1''\#g; here, here)$,
$(c_1''\#\$c_1'\#g; here, out)$

(the strings marked with c_1', c_2' are sent out, in two steps).

The membrane sub-structure associated with $D_{i,2}$ is similar, but we check the appearance of strings $t_{i,j}, 1 \leq j \leq q_i$, as substrings of the strings c_1wc_2 sent here, and, when no such substring is present in w, we cut at the pattern $u_{i,3}u_{i,4}$ and we mark the produced fragments with $f_{i,1}, f_{i,2}$ instead of $e_{i,2}, e_{i,2}$.

From the skin membrane, these fragments are sent to the "ligator" sub-structure, where the simulation of the splicing is completed. It is important to note that from sending the strings from the skin membrane to the "ligator" first membrane until sending back to the skin membrane strings of the form c_1zc_2 we need six steps.

Thus, always we accomplish a task in an even number of steps. This is important for checking the presence of strings $s_{i,j}, t_{i,j}$, in membranes of the type $(j, 1)$ as discussed above: at odd steps there is no string $\bar{f}c_2$ present in this membrane, so the only rule which can involve the string c_1wc_2 to be checked is that of the form $(s_{i,j}\#\$g\#g; here, here)$. Thus, if this rule *can* be applied, then it *must* be applied, which ensures the correct checking of the forbidding context condition.

Thus, we can conclude that the P system Π precisely simulates the functioning of the H system γ and that $L(\gamma) = L(\Pi)$. □

6 Simulating a Communicating Distributed H System by a Splicing P System

The condition imposed by a filter associated with a component (A_i, R_i, V_i) of a CD H system is similar to that imposed by the forbidding contexts: a string x is accepted by the previous component only if it contains no symbol from $V - V_i$. Thus, we can simulate a CD H system by using the techniques already developed for forbidding context H systems. That is, the following result is as expected:

Theorem 5. *Each communicating distributed H system can be simulated by a splicing P system.*

Proof. For a CD H system $\Gamma = (V, T, (A_1, R_1, V_1), \ldots, (A_n, R_n, V_n))$, we construct the splicing P system Π as follows. In a skin membrane, we place n membrane sub-structures, associated with each of the components of Γ. The shape of the membrane structure is as indicated in Figure 3 (we have given no detail for membranes associated with other components than the ith one).

The checking of the non-appearance of symbols from $V - V_i$ in the string under inspection is done exactly as in the previous proof. Essential differences appear when simulating the rules in R_i.

First, at the beginning we introduce all strings of the form c_1wc_2' in the innermost membrane, that called "simulator" in Figure 3. For all rules $r = u_1\#u_2\$u_3\#u_4 \in R_i$, we introduce here the rules

$(u_1\#u_2\$d_1\#d_2; here, here)$,
$(u_3\#u_4\$d_3\#d_4; here, here)$,
$(u_1\#d_2\$d_3\#u_4; here, here)$,
$(u_3\#d_4\$d_1\#u_2; here, here)$,
$(u_1\#d_2\$d_3\#u_4; out, here)$,
$(u_3\#d_4\$d_1\#u_2; out, here)$.

In this way, in the simulator we can perform any number of splicing operations exactly as in Γ, where we compute $\sigma_i^*(L_i)$, for the current set of strings L_i. At any moment, any string can be sent out. (Such strings are of the form c_1xc_2'.) However, because we can perform any number of splicings and because we have both rules $(u_1\#d_2\$d_3\#u_4; here, here), (u_3\#d_4\$d_1\#u_2; here, here)$ and rules $(u_1\#d_2\$d_3\#u_4; out, here), (u_3\#d_4\$d_1\#u_2; out, here)$, copies of all strings remain in the simulator (and this is permitted by the mode of work of Γ). This is important for the strings which are not accepted by any other component and must remain in the component which has produced them.

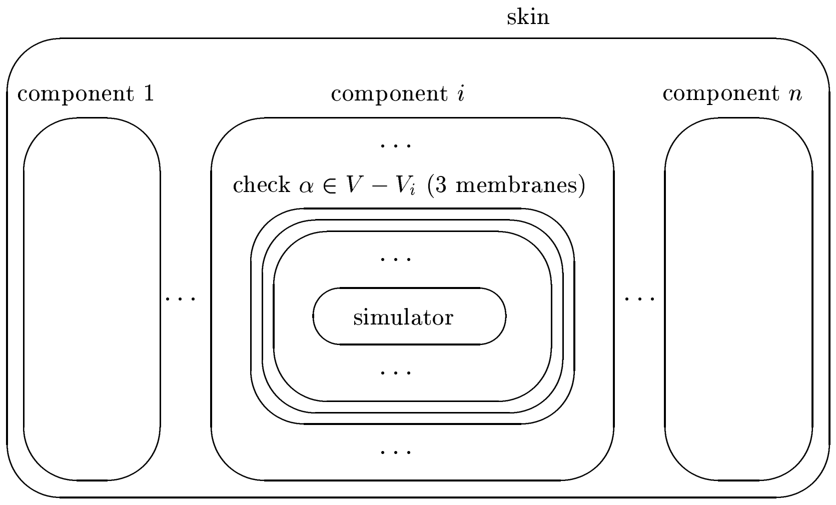

Figure 3. The shape of the membrane structure of the splicing P system from the proof of Theorem 5.

Note also that each splicing operation with respect to Γ is simulated in two steps in Π, one when we introduce the symbols d_1, d_2, d_3, d_4 and one when these symbols are removed. Remember that we have to be careful with the parity of the number of steps in order to check correctly the non-appearance of symbols in the examined strings. In all upper membranes, we provide rules which sends up the strings, in two steps in each membrane.

In this way, all strings obtained after an arbitrary number of splicing operations in any component can be brought together in the skin membrane. From the skin membrane, all these strings are sent back to the lower membranes, those associated with the components of Γ (we can arrange to do this in two steps, to keep the parity). The strings are now checked whether or not they are accepted by the filters and, if they arrive in a simulator, new splicings are performed on them.

In the simulator associated with the component (A_1, R_1, V_1) we also consider rules which replace c_2 with $\bar{c}_2$. In all upper membranes we introduce rules which send up strings of the form $c_1w\bar{c}_2$. In the skin membrane we remove c_1 and $\bar{c}_2$, and we send out of the system the produced strings. If they are terminal, then they are accepted in the language $L(\Pi)$.

From the above explanations and details, from the construction in the proof of Theorem 4, and completing the missing details (the initial strings in each membrane, rules for passing the axioms from a step to another one – when necessary –, or parity preserving tricks), the reader can see that the system Π simulates the work of Γ in the proper way and that $L(\Gamma) = L(\Pi)$. $\square$

7 Final Remarks

We have illustrated here the power of the membrane computing paradigm by simulating the functioning of three basic types of H systems – with permitting contexts, with forbidding contexts, and communicating distributed H systems – by means of splicing P systems. We stress the fact that the focus was the simulation itself, not the equivalence of the two mechanisms (in the linguistics terminology, we are not interested in their *weak generative capacity*). In other words, through the simulation by P systems one can expect to get a unified view to understand the differences (e.g., different shapes of membrane structures) among the variations of controlled H systems and related models.

Of course, it is possible to simulate other types of H systems by splicing P systems, but this will only emphasize the versatility of P systems, a fact already convincingly proved.

Acknowledgement

Research supported by "Research for Future" Program no. JSPS-RFTF 96I00101, from the Japan Society for the Promotion of Science.

References

[1] E. Csuhaj-Varju, L. Kari, Gh. Păun, Test tube distributed systems based on splicing, *Computers and AI*, 15, 2-3 (1996), 211–231.
[2] K. Culik II, T. Harju, Splicing semigroups of dominoes and DNA, *Discrete Appl. Math.*, 31 (1991), 261–277.
[3] J. Dassow, Gh. Păun, On the power of membrane computing, *J. Univ. Computer Sci.*, 5, 2 (1999), 33–49.
[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] Gh. Păun, Computing with membranes. An introduction, *Bulletin of the EATCS*, 67 (Febr. 1999), 139–152.
[6] Gh. Păun, Computing with membranes, submitted, 1998 (see also *TUCS Research Report* No. 208, November 1998, http://www.tucs.fi).
[7] Gh. Păun, Computing with membranes. A variant, submitted, 1999 (see also *CDMTCS Report* No. 098, 1999, of CS Department, Auckland Univ., New Zealand, www.cs.auckland.ac.nz/CDMTCS).
[8] Gh. Păun, G. Rozenberg, A. Salomaa, *DNA Computing. New Computing Paradigms*, Springer-Verlag, Heidelberg, 1998.
[9] Gh. Păun, G. Rozenberg, A. Salomaa, Membrane computing with external output, submitted, 1999 (see also *TUCS Research Report* No. 218, December 1998, http://www.tucs.fi).
[10] Gh. Păun, Y. Sakakibara, T. Yokomori, P systems on graphs of restricted forms, submitted, 1999.

[11] 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.

[12] Gh. Păun, S. Yu, On synchronization in P systems, *Fundamenta Informaticae*, 38 (1999) (see also *CS Department TR* No 539, Univ. of Western Ontario, London, Ontario, 1999, www.csd.uwo.ca/faculty/syu/TR539.html).

[13] D. Pixton, Regularity of splicing languages, *Discrete Appl. Math.*, 69 (1996), 101–124.

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