COMMUNICATING DISTRIBUTED H SYSTEMS WITH SIMPLE SPLICING RULES

KAMALA KRITHIVASAN, PRAHLADH HARSHA and MURALIDHAR TALUPUR
Department of Computer Science and Engineering,
Indian Institute of Technology, Madras,
Chennai - 600036, India,
E-mail : kamala@iitm.ernet.in

***Abstract*— In this paper we define communicating distributed H systems with simple splicing rules of types (1,3), (1,4) and (2,3) and study the generative capacity.**

keyword: Splicing systems, contextfree languages, simple H systems, test tube systems, communicating distributed H systems

I. INTRODUCTION

Splicing systems were defined to model the recombinant behaviour of DNA strands [6], [7], [8]. Test tube systems were defined in [1] as a symbol processing mechanism having a parallel architecture with the components being test tubes working as splicing schemes. The communication is performed by redistributing the contents of the tubes according to a specified protocol. It was shown that in [1] test tube systems have power equal to that of Turing Machines. They are also called communicating distributed H systems.

Simple H systems were defined in [2] by restricting the form of the rules. Four types of simple H systems were defined in [2] are (1,3), (2,4), (1,4) and (2,3). (1,3) and (2,4) essentially define the same systems. It is known that simple H systems of all the 3 types generate only regular languages and the family of languages generated by them are not comparable.

In [3], a detailed study of simple H systems with target alphabet and permitting context is made. The generative power of these families are compared and presented in the form of a table. Simple H systems with target alphabet are called simple extended H systems. It is shown in [3] that $SEH_{(2,3)}$ is more powerful than $SEH_{(1,4)}$ and $SEH_{(1,3)}$. It is also shown in [3] that every context free language can be generated by a $SEH_{(2,3)}$ system with permitting context.

In [4] it is shown that $SEH_{(2,3)}$ systems with permitting context are as powerful as $EH(FIN, p[1])$ systems. In [14], it is shown that $SEH_{(2,3)}(p) \subseteq CF$. Thus we get a characterisation of CFL in terms of $SEH_{(2,3)}(p)$. Controlled and distributed H systems of a small diameter are considered in [5] and some characterizations of recursively enumerable sets are given.

In this paper we define simple test tube systems(STT) of four types (1,3), (2,4), (1,4) and (2,3). The definition of (1,3),(2,4) being equivalent, this essentially reduces to three types. We consider (2,3) type in detail and show that this generates nonregular languages. We also show that every context free language can be generated by Simple Test Tube Systems of (2,3) type. Then [14], it is shown that every language generated by a STT (2,3) type is generated by a CFG. This gives the characterisation of CFL in terms of STT of type (2,3). We also give examples of regular languages which cannot be generated by STTs of type (1,3) and (1,4).

In the next section we give some preliminary definitions and define simple test tube systems. In section 3, we show that every context free language can be generated by a simple test tube system. The directions for further work are mentioned in section 4.

II. SIMPLE TEST TUBE SYSTEMS

In this section a simple processing mechanism having a parallel architecture with the components being test tubes working as simple splicing schemes is presented. The increase in generative capacity of *simple test tube systems* over simple H systems is discussed. In the following subsection, the basic definitions (H systems, EH systems, simple H systems and test tube systems) and notations employed are outlined. In section II-B, simple test-tube systems are introduced.

A. Splicing Systems

We present below the basic notations and definitions required for building the concept of simple test tube systems. For more information on splicing systems the reader is directed to some of the excellent material in this field [1], [6], [7], [8].

1) H systems and EH systems[8]:

Definition 1: A splicing rule (over an alphabet V) is a string $r = u_1\#u_2\$u_3\#u_4$, where $u_i \in V^*, 1 \leq i \leq 4$, and $\#, \$$ are special symbols not in V. For such a rule r and the strings $x, y, z \in V^*$, the splicing operation is defined as follows:

$$(x, y) \vdash_r z \quad \text{iff} \quad x = x_1u_1u_2x_2, y = y_1u_3u_4y_2, \; z = x_1u_1u_4y_2 \text{ for some } x_1, x_2, y_1, y_2 \in V^*$$

x, y are then said to be *spliced* at the *sites* u_1u_2, u_3u_4 respectively to obtain the string z. The strings x, y are called the *terms* of splicing. When understood from context, the index r is omitted from $\vdash_r$

A *splicing scheme* (or an *H scheme*) is a pair $\sigma = (V, R)$ where V is an alphabet and R is a set of splicing rules (over

V). For a language $L \subseteq V^*$, the following are defined:

$$\sigma(L) = \{w \in V^* | (x, y) \vdash_r w \text{ for } x, y \in L, r \in R\}$$

The following are also defined for the language L

$$\begin{aligned} \sigma^0(L) &= L \\ \sigma^{i+1}(L) &= \sigma^i(L) \cup \sigma(\sigma^i(L)) \\ \sigma^*(L) &= \bigcup_{i \geq 0} \sigma^i(L) \end{aligned}$$

Thus, $\sigma^*(L)$ is the smallest language containing L and closed under the splicing operation.

Definition 2: An extended splicing system is a quadruple

$$\gamma = (V, T, A, R)$$

where V is an alphabet, $T \subseteq V$ (the terminal alphabet), $A \subseteq V^*$ the set of axioms and $R \subseteq V^*\#V^*\$V^*\#V^*$ the set of rules. The pair $\sigma = (V, R)$ is called the underlying H schema of γ. The language generated by γ is defined as follows

$$L(\gamma) = \sigma^*(A) \cap T^*$$

An H system $\gamma = (V, T, A, R)$ is said to be of *type* (F_1, F_2) for two families of languages F_1, F_2 if $A \in F_1, R \in F_2$. $EH(F_1, F_2)$ is used to denote the family of languages generated by extended H systems of type (F_1, F_2). An H system $\gamma = (V, T, A, R)$ with $V = T$ is said to be *non-extended;*, here the H system is denoted by $\gamma = (V, A, R)$. The family of languages generated by non-extended H systems of type (F_1, F_2) is denoted by $H(F_1, F_2)$. Obviously, $H(F_1, F_2) \subseteq EH(F_1, F_2)$.

The following are a few important results regarding H systems and EH systems [9], [10], [11].

Theorem 1: (i) $H(FIN, FIN) \subseteq REG$.
(ii) $EH(FIN, FIN) = REG$
(iii) $EH(FIN, REG) = RE$
Thus these EH systems are as powerful as TMs. However it is not realistic to deal with a infinite number of rules even if they were a regular set of rules. To preserve the universal computational power while maintaining the infiniteness of all components involved, everal variants were proposed. One such variant is the working of several of these H systems in unison in a parallel manner. These parallel H systems are called test-tube systems, which will be discussed in detail in Section 5

2) Simple H systems[2], [8]:

Definition 3: A simple H system is a triple

$$\Gamma = (V, A, M)$$

where V is an alphabet, $A \subseteq V^*$ is a finite set of axioms and $M \subseteq V$
The elements of M are called *markers*. One can consider four ternary relations on the language V^*, corresponding to the splicing rules of the form

$$a\#\$a\#, \#a\$\#a, a\#\$\#a, \#a\$a\#$$

where a is an arbitrary element of M. These four rules are respectively called splicing rules of type $(1, 3), (2, 4), (1, 4), (2, 3)$.

Clearly, rules of type $(1, 3)$ and $(2, 4)$ define the same operation, for $x, y, z \in V^*$ and $a \in M$ we obtain

$$(x, y) \vdash^a_{(1,3)} z \text{ iff } x = x_1 a x_2, y = y_1 a y_2, z = x_1 a y_2, \text{ for some } x_1, x_2, y_1, y_2 \in V^*.$$

For the other types, the splicing is performed as follows:

$$(x, y) \vdash^a_{(1,4)} z \text{ iff } x = x_1 a x_2, y = y_1 a y_2, z = x_1 a a y_2, \text{ for some } x_1, x_2, y_1, y_2 \in V^*.$$

$$(x, y) \vdash^a_{(2,3)} z \text{ iff } x = x_1 a x_2, y = y_1 a y_2, z = x_1 y_2, \text{ for some } x_1, x_2, y_1, y_2 \in V^*.$$

Then for $L \subseteq V^*$ and $(i, j) \in \{(1, 3), (1, 4), (2, 3)\}$, the following are defined.

$$\sigma_{(i,j)}(L) = \{w \in V^* | (x, y) \vdash^a_{(i,j)} w \text{ for some } x, y \in L, a \in M\}$$

$$\begin{aligned} \sigma^0_{(i,j)}(L) &= L \\ \sigma^{k+1}_{(i,j)}(L) &= \sigma^k_{(i,j)}(L) \cup \sigma_{(i,j)}(\sigma^k_{(i,j)}(L)), k \geq 0 \\ \sigma^*_{(i,j)}(L) &= \bigcup_{k \geq 0} \sigma^k_{(i,j)}(L) \end{aligned}$$

Definition 4: The language generated by $\Gamma = (V, A, M)$ in case (i, j) is defined as follows

$$L_{(i,j)}(\Gamma) = \sigma^*_{(i,j)}(A)$$

As in all the cases mentioned, both the axiom set and the rule set are finite, it follows that for any simple H system Γ, the languages $L_{(i,j)}(\Gamma)$ are regular.[9], [10] It has been shown that each two of the three family of languages obtained in this way are incomparable.[2]

3) Test tube systems[8]:

Definition 5: A test tube (TT) system, (it is also called communicating distributed H systems) (of degree $n, n \geq 1$) is a construct

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

where V is an alphabet, $A_i \subseteq V^*, R_i \subseteq V^*\#V^*\$V^*\#V^*$ and $V_i \subseteq V$ for each $1 \leq i \leq n$
Each triple (A_i, R_i, V_i) is called a *component* of the system or a *tube*. A_i is the set of axioms of tube i, R_i the set of splicing rules of the tube i and V_i the *selector* of the tube i.

Let

$$B = V^* - \bigcup_{i=1}^{n} V_i^*$$

The pair $\sigma_i = (V, R_i)$ is the underlying H schema associated with the i-th component 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 i-th tube.

Definition 6: For any two configurations $(L_1, \ldots, L_n), (L'_1, \ldots, L'_n)$, the following is defined

$$(L_1, \ldots, L_n) \vdash (L'_1, \ldots, L'_n) \text{ iff for each } i, 1 \leq i \leq n$$

$$L'_i = \left[\bigcup_{j=1}^{n} (\sigma^*_j(L_j) \cap V_i^*)\right] \cup (\sigma^*_i(L_i) \cap B)$$

In other words, the contents of each tube is spliced according to the associated set of rules and the result is redistributed among the n tubes according to the selectors $V_1, \ldots V_n$. The part which cannot be redistributed remains in the tube.

Definition 7: The language generated by the test tube system
$\Gamma = (V, (A_1, R_1, V_1), \ldots, (A_n, R_n, V_n))$ is
$L(\Gamma) = \{w \in V^* | w \in L_1^{(t)} for\ some\ (A_1, \ldots, A_n) \Rightarrow^*$
$(L_1^{(t)} \ldots L_n^{(t)}), t \geq 0\}$
where $\Rightarrow^*$ is the reflexive and transitive closure of $\Rightarrow$.
Given two families of languages F_1, F_2, $TT_n(F_1, F_2)$ denotes the family of languages $L(\Gamma)$, for some $\Gamma = (V, (A_1, R_1, V_1), \ldots, (A_m, R_m, V_m))$ with $m \leq n, A_i \in F_1, R_i \in F_2$ for each $i, 1 \leq i \leq m$. Γ is then said to be of type (F_1, F_2). When n is not specified, Γ is said to belong to the family $TT_\infty(F_1, F_2)$.

The generative power of the splicing systems is increased when working in parallel as mentioned earlier. This is seen from the following results.[1]

Theorem 2: (i) $TT_7(FIN, FIN) = TT_\infty(FIN, FIN) = RE$.
(ii) $TT_6(FIN, FIN)$ contains non-recursive languages.

B. Simple test tube systems

In this section, we consider a set of simple H systems working in parallel. Each component of this system, referred to as a *simple test tube* is a simple H system. As in simple H systems, simple test tubes can also work in any of the four modes(types) $(1,4), (1,3), (2,4), (2,3)$. The formal definition of simple test tube systems is given below.

Definition 8: A simple test tube (or distributed simple H system) (STT) system (of degree $n, n \geq 1$) is a construct

$$\Gamma = (V, (A_1, M_1, V_1), \ldots, (A_n, M_n, V_n))$$

where V is an alphabet, $A_i \subseteq V^*, M_i \subseteq V$ and $V_i \subseteq V$ for each $1 \leq i \leq n$
Each triple (A_i, R_i, V_i) is called a *component* of the system or a *simple tube*. A_i is the set of axioms of tube i, M_i the set of markers of the tube i and V_i the *selector* of the tube i.

The pair $\sigma_i = (V, A_i, M_i)$ is the underlying simple H schema associated with the i-th component of the system.

Definition 9: For a STT system $\Gamma == (V, (A_1, M_1, V_1), \ldots, (A_n, M_n, V_n))$, depending on the type $(1,4), (1,3), (2,3), (2,4)$, we associate 4 different TT systems.
(i) Type $(1,4)$:
$\Gamma^{(1,4)} = (V, (A_1, R_1^{(1,4)}, V_1), \ldots, (A_n, R_n^{(1,4)}, V_n))$ where each $R_i^{(1,4)} = \{a\#\$\#a | a \in M_i\}, 1 \leq i \leq n$.
(i) Type $(1,3)$:
$\Gamma^{(1,3)} = (V, (A_1, R_1^{(1,3)}, V_1), \ldots, (A_n, R_n^{(1,3)}, V_n))$ where each $R_i^{(1,3)} = \{a\#\$a\# | a \in M_i\}, 1 \leq i \leq n$.
(i) Type $(2,3)$:
$\Gamma^{(2,3)} = (V, (A_1, R_1^{(2,3)}, V_1), \ldots, (A_n, R_n^{(2,3)}, V_n))$ where each $R_i^{(2,3)} = \{\#a\$a\# | a \in M_i\}, 1 \leq i \leq n$.
(i) Type $(2,4)$:
$\Gamma^{(2,4)} = (V, (A_1, R_1^{(2,4)}, V_1), \ldots, (A_n, R_n^{(2,4)}, V_n))$ where each $R_i^{(2,4)} = \{\#a\$\#a | a \in M_i\}, 1 \leq i \leq n$.
Now that we have associated STT systems with TT systems, we can proceed to define the actual working of the STT system.

Definition 10: A STT system $\Gamma = (V, (A_1, M_1, V_1), \ldots, (A_n, M_n, V_n))$ working in type $(i,j), (i,j) \in \{(1,4), (1,3), (2,3), (2,4)\}$ is defined to be the TT system $\Gamma^{(i,j)}$.
Given a family of languages F, $STT_n^{(i,j)}(F)$ denotes the family of languages $L(\Gamma)$, for some STT system $\Gamma = (V, (A_1, M_1, V_1), \ldots, (A_m, M_m, V_m))$ working according to type (i,j) with $m \leq n, A_k \in F$, for each $k, 1 \leq k \leq m$ and $(i,j) \in \{(1,4), (1,3), (2,3), (2,4)\}$. Γ is then said to be of type (F). When n is not specified, Γ is said to belong to the family $STT_\infty^{(i,j)}(F)$. As noted earlier, (1,3) and (2,4) systems are equivalent.

C. Nonreguar sets produced by STT

It is interesting to note that though simple H systems do not generate languages beyond REG, distributed simple H systems do produce languages not in REG.

We give an example of a language generated by STT of type (2,3) which is not regular.
Example 1: Let Γ be the following STT system of type $(2,3)$.

$$\Gamma = (V, (A_1, M_1, V_1), (A_2, M_2, V_2))$$

where

$$\begin{aligned} V &= \{a, b, c, d, c', d'\} \\ A_1 &= \{cabd, c'ac, dbd'\} \\ M_1 &= \{c, d\} \\ V_1 &= \{a, b, c, d\} \\ A_2 &= \{cc', d'd\} \\ M_2 &= \{c', d'\} \\ V_2 &= \{a, b, c', d'\} \end{aligned}$$

Let us consider the sequence of operations performed on the string of type ca^ib^id. We have such a string to start with namely, $cabd \in A_1$. In simple tube 1, $(c'ac, ca^ib^id) \Rightarrow c'a^{i+1}b^id$, which then performs the following: $(c'a^{i+1}b^id, dbd') \Rightarrow c'a^{i+1}b^{i+1}d'$ which is then accepted by tube 2. $(cc', c'a^{i+1}b^{i+1}d') \Rightarrow ca^{i+1}b^{i+1}d'$ is then performed by tube 2, the product of which performs the operation $(ca^{i+1}b^{i+1}d', d'd) \Rightarrow ca^{i+1}b^{i+1}d$ which is accepted by simple tube 1. This process repeats and we thus note that

$$L(\Gamma) \cap ca^+b^+d = \{ca^nb^nd | n \geq 1\}$$

Hence $L(\Gamma)$ is not a regular language.

Thus we see that simple test tube system generate nonregular languages, indicating that simple test tube systems have higher generative power than simple H systems.

D. Examples of regular languages not in STT of type (1,3) and (1,4)

In this section we give examples of regular languages which are not present in STT of (1,3) and (1,4) types.

1) Example of STT of (1,3) type: **Claim:** The regular language $L = \{\ 0^n \mid n \geq 2\ \}$ is not generated by STT of (1,3) type.

Proof: We prove this by contradiction. Assume there is a STT of (1,3) type which generates L. Let k be the largest integer such that 0^k is in some axiom set of STT system. Hence 0^{k+1} must be generated in some test tube using (1,3) splicing. Say it is generated in testtube t. Then it must have been generated from two strings of the form 0α and $\beta 0$, $\alpha, \beta \in V_t^*$ using a rule $r = 0$. But

$$(0\alpha, \beta 0) \vdash^r_{(1,3)} 0$$

(The sites are underlined.) Then $0 \in L$, which is a contradiction. So there in no STT of (1,3) type which can generate L.

2) Example of STT of (1,4) type: **Claim:** The regular language L=$(10)^*1$ is not generated by any STT of (1,4) type.

Proof: We again prove by contradiction. Let there be a STT, S, of (1,4) type which hgenerates this language. Let k be the largest integer such that $(10)^k1$ is present in some axiom set of S. So $(10)^{k+1}1$ must be produced by (1,4) splicing. Also, it can be seen that $(10)^{k+1}1$ can be generated only by splicing strings of the form 1α and $\beta 1$ using either 0 or 1 as the splicing site. It is easy to note that in case 0 is used as splicing site then the resulting string will have two consequtive 0s which is not possible. The same holds if 1 is used as the splicing site. Hence the proof.

III. Representation of Context Free Languages

In this section we show that, every context free language can be generated by a simple test tube system of type (2,3).

Theorem 3: For each $L \in CF$, there exists a STT system of type (2,3) Γ such that $L = L(\Gamma)$.

Proof Consider a CF grammar $G = (N, T, P, S)$ such that $L = L(G)$ in the Chomsky normal form where rules are of the form

$$\begin{aligned}(r): &\quad A \to BC, A, B, C \in N\\ (r'): &\quad A \to a, A \in N, a \in T\end{aligned}$$

Let there be k rules of type (r) in the grammar G. Consider the STT system (of type (2,3))

$$\begin{aligned}\Gamma = \quad & (V, (A_1, M_1, V_1), \cdots, (A_4, M_4, V_4)\\ & (A_{r_1 1}, M_{r_1 1}, V_{r_1 1}), \ldots, (A_{r_1 5}, M_{r_1 5}, V_{r_1 5}),\\ & \vdots\\ & (A_{r_k 1}, M_{r_k 1}, V_{r_k 1}), \ldots, (A_{r_k 5}, M_{r_k 5}, V_{r_k 5}))\end{aligned}$$

where

$$V = T \cup \{X_1, X_2 | X \in N\} \cup \{r | r : A \to BC \in P\}$$

$$\begin{aligned}A_1 &= \phi\\ M_1 &= \phi\\ V_1 &= T\\ A_2 &= \{S_1\}\\ M_2 &= \{S_1\}\\ V_2 &= \{S_1\} \cup T\\ A_3 &= \{S_1\}\\ M_3 &= \{S_2\}\\ V_3 &= \{S_1, S_2\} \cup T\\ A_4 &= \{A_1 a A_2 | A \to a \in P\}\\ M_4 &= \Phi\\ V_4 &= T \cup \{S_1, S_2\}\end{aligned}$$

For each rule (r_i) of the type $(r_i) : A \to BC, 1 \leq i \leq k$, we have the following 5 simple tubes.

$$\begin{aligned}A_{r_i 1} &= \{B_2 r_i\}\\ M_{r_i 1} &= \{B_2\}\\ V_{r_i 1} &= T \cup \{B_1, B_2\}\end{aligned}$$

$$\begin{aligned}A_{r_i 2} &= \{r_i C_1\}\\ M_{r_i 2} &= \{C_1\}\\ V_{r_i 2} &= T \cup \{C_1, C_2\}\end{aligned}$$

$$\begin{aligned}A_{r_i 3} &= \{A_1 B_1\}\\ M_{r_i 3} &= \{B_1\}\\ V_{r_i 3} &= T \cup \{B_1, r_i\}\end{aligned}$$

$$\begin{aligned}A_{r_i 4} &= \{C_2 A_2\}\\ M_{r_i 4} &= \{C_2\}\\ V_{r_i 4} &= T \cup \{r_i, C_2\}\end{aligned}$$

$$\begin{aligned}A_{r_i 5} &= \Phi\\ M_{r_i 5} &= \{r_i\}\\ V_{r_i 5} &= T \cup \{A_1, A_2, r_i\}\end{aligned}$$

We shall show that if $A \Rightarrow^* w (\in T^*)$ for some $A \in N$, then $A_1 w A_2$ is generated by one of the tubes. To start with we have for all $A \to a \in P$, $A_1 a A_2$ as axioms in simple tube 1.

Suppose, we are able to generate strings $B_1 w_1 B_2$ and $C_1 w_2 C_2$ and we have the rule $(r_i) : A \to BC \in P, 1 \leq i \leq k$, it is sufficient if we show that $A_1 w_1 w_2 A_2$ is also generated by one of the simple tubes. Simple tube $(r_i 1)$ accepts $B_1 w_1 B_2$ and by splicing $(B_1 w_1 B_2, B_2 r_i) \vdash B_1 w_1 r_i$. $B_1 w_1 r_i$ is accepted by simple tube $(r_i 3)$ and by splicing $(A_1 B_1, B_1 w_1 r_i) \vdash A_1 w_1 r_i$ is generated. Similarly simple

tube $(r_i 2)$ accepts $C_1 w_2 C_2$ and generates $r_i w_2 C_2$ which is then further spliced in simple tube $(r_i 4)$ along with $C_2 A_2$ to give $r_i w_2 A_2$. Strings $A_1 w_1 r_i$ and $r_i w_2 A_2$ are then spliced in simple tube $(r_i 5)$ to give $A_1 w_1 w_2 A_2$.

Thus, all strings of the form $S_1 w S_2, w \in L(G)$ are generated in due course and are filtered in simple tubes 3 and 4. In test tube 3 the following reaction takes place:

$$(S_1 w S_2, S_2) \vdash S_1 w$$

The product is transfered to test tube 2 and the following reaction takes place:

$$(S_1, S_1 w) \vdash w$$

This terminal string w is then filtered into test tube 1. Thus all terminal strings derived from S in the CFG get collected in test tube 1. Hence, we can see that

$$L(\Gamma) = L$$

It should be noted that only strings of the form $S_1 w S_2$ generated in test tube 4 get transformed into strings w in the test tube 1. Hence if w is generated in test tube 1, $S_1 w S_2$ is generated in 4 which menas $S \Rightarrow^* w$ in the CFG. □

Theorem 4: For each STT system of type (2,3) Γ there exists a CFG G such that $L(\Gamma) = L(G)$.
The proof is lengthy and is given in [14].

IV. Conclusions

In this paper we have defined simple test tube systems and shown that every context free language can be generated by a simple test tube system of (2,3) type. In [3] it is shown that every CFL can be generated by a simple extended H system of (2,3) type with permitting context. In [3] it is also shown that simple extended H system of (2,3) type is more powerful than (1,4) or (1,3) type.

It will be an interesting topic to find out the power of simple test tube systems of types (1,4) and (1,3). We conjecture that any language generated by STT of type (1,3) or (1,4) is regular. Hence the families STT(1,4) and STT(1,3) will form subclasses of regular sets. Hence it looks as though simple test tube systems of (2,3) type are more powerful than simple H systems of the other types. It would also be worthwhile considering hybrid systems where some test tubes follow one type of rules and some other test tubes follow other type of rules.

References

[1] E. Csuhaj-Varju, L. Kari and Gh. Păun, Test Tube distributed systems based on splicing, *Computers and AI,* 15, 2-3(1996), 211-232.

[2] A. Mateescu, Gh. Păun, G. Rozenberg and A. Salomaa, Simple Splicing Systems, *Discrete Appl. Math.,* 84(1998), 145-163.

[3] V. T. Chakaravarthy and K. Krithivasan, Some Results on Simple Extended H-systems, *Romanian Journal of Information Science and Technology*, Vol. 1, Number 3(1998), 203-215.

[4] S. Lakshminarayanan, T.Muralidharan, K. Krithivasan and C.Pandu Rangan, On the generative power of simple H systems, *JALC*, 5(2000) 4, 457-473.

[5] A.Paun and M.Paun, Controlled and Distributed H systems of a small diameter, *Computing with Biomolecules*, ed. G.Paun, Springer (1998), 239-254.

[6] T. Head, Formal Language Theory and DNA: an analysis of the generative capacity of specific recombinant behaviors, *Bull. Math. Biology,* 49(1987), 737-759.

[7] Gh. Păun, G. Rozenberg and A. Salomaa, DNA Computing-New Computing Paradigms, *Springer*(1998).

[8] T. Head, Gh. Păun and D. Pixton, Language Theory and Molecular genetics, generative mechanisms suggested by DNA recombination, chapter 7 in vol 2 of Handbook of Formal Languages,ed., G.Rozenberg and A.Salomaa, Springer Verlog, Berlin(1997), 295-360.

[9] D.Pixton, Regularity of Splicing Languages, *Discrete Appl. Math.*, 69(1996),101-124.

[10] K.Cullik II and T.Harju, Splicing Semigroups of Dominoes and DNA, *Discrete Appl. Math.*, 31(1991),261-277.

[11] Gh. Paun, Regular extended H systems are computationally universal *J. Automata, Languages, Combinatorics,* 1(1996), 27-36.

[12] Gheorge Paun, Computing by Splicing. How simple rules?. *Bulletin of the EATCS*, 60(1996), 67-86.

[13] V. T. Chakaravarthy and K. Krithivasan, A note on Extended H systems with permitting/ forbidden context of radius one, *Bulletin of EATCS*, 62(1997), 208-213.

[14] T. Muralidhar, Generative Power splicing Systems, *B.Tech Project Report,Indian Institute of Technology, Madras*(2000).