Communicating Distributed H Systems: Optimal Results with Efficient Ways of Communication

Shankara Narayanan Krishna

Department of Computer Science and Engineering,
Indian Institute of Technology, Bombay,
Powai, Mumbai 400 076, India
krishnas@cse.iitb.ac.in

Abstract. Distributed H systems and several variants of distributed H systems have been studied extensively [1, 2, 3, 4]. This paper is an effort in the direction of obtaining efficient distributed systems. To this end, a universality result using 2 components is obtained using two-level distributed H systems. This is an improvement over the existing universality result with 3 components. Further, we propose *lazy* communicating distributed H systems (LCDH systems), a variant of communicating distributed H systems, with lesser communication. A universality result is obtained with this variant, using only 2 components. This improves the universality result $RE = CDH_3$ by reducing the number of components as well as the communication between components.

1 Introduction

Communicating distributed H (CDH) systems were introduced in [1] as efficient extensions of splicing systems. In CDH systems, parts of the model which are able to work independently can be separated, and the result can be obtained by synthesizing the partial results produced by the individual parts. However, the communication in CDH systems is rather inefficient since they allow transport of possibly the entire contents of each component in every step. Distributed H systems [2, 3, 4] have been studied extensively, with different means of communication, one of them being two-level distributed H systems. These systems do not allow communication between components in the sense of CDH systems, and so are more efficient.

In this paper, we concentrate on two-level distributed H systems and CDH systems. We introduce *lazy* CDH systems as a variant of CDH systems, wherein, some components are classified as *lazy*, depending on the way they communicate. The idea of having *lazy* components is to reduce the number of strings that can be considered for communication in every step. We also obtain an unexpected improved universality result for two-level distributed H systems (without any laziness conditions), as well as a universality result for *lazy* CDH systems, both in 2 components, which show that with better means of communication, the number of components can be reduced. In the following subsection, we give some basic definitions and notions of formal language theory used in this paper; more details can be found in [4].

A. Carbone and N.A. Pierce (Eds.): DNA11, LNCS 3892, pp. 181–192, 2006.

1.1 Prerequisites and Basic Definitions

An alphabet is a finite nonempty set of symbols. For an alphabet V, we denote by V^* the set of strings of symbols over V. The empty string is denoted by λ. V^* is the free monoid generated by V under the operation of concatenation. (The unit element of this monoid is λ). Each subset of V^* is called a *language* over V.

Let $x \in V^*$. If $x = x_1x_2$, for some $x_1, x_2 \in V^*$, then x_1 is called a *prefix* of x and x_2 is called a *suffix* of x. If $x = x_1x_2x_3$, for some $x_1, x_2, x_3 \in V^*$, then x_2 is called a *substring* of x. The length of a string x is denoted by $|x|$. The number of occurrences of a symbol a in x denoted $|x|_a$.

Consider an alphabet V and two special symbols $\#, \$ \notin V$. A splicing rule over V is a string $u_1\#u_2\$u_3\#u_4$ where $u_1, u_2, u_3, u_4 \in V^*$. For a splicing rule $r = u_1\#u_2\$u_3\#u_4$, the result of splicing two strings $x = x_1u_1u_2x_2, y = y_1u_3u_4y_2$ is defined as $(x, y) \models_r (z, w)$ where $z = x_1u_1u_4y_2, w = y_1u_3u_2x_2$.

An H scheme is a pair $\sigma = (V, R)$ where V is an alphabet and $R \subseteq V^*\#V^*\$V^*\#V^*$ is a set of splicing rules. For an H scheme $\sigma = (V, R)$, and a language L, the set obtained by using the splicing operation on L is denoted by $\sigma_2(L) = \{z \in V^* \mid (x, y) \models_r (z, w) \text{ or } (x, y) \models_r (w, z)\}$, for some $x, y \in L$, and $r \in R$. $\sigma_2^i(L)$ is defined inductively: $\sigma_2^0(L) = L$, $\sigma_2^{i+1}(L) = \sigma_2^i(L) \cup \sigma_2(\sigma_2^i(L)), i \geq 0$. Hence, $\sigma_2^*(L) = \bigcup_{i \geq 0} \sigma_2^i(L)$.

An extended H system is a quadruple $\gamma = (V, T, A, R)$ where $T \subseteq V$ is the terminal alphabet, R is the set of splicing rules and A is the set of axioms. Thus, γ has an underlying H scheme $\sigma = (V, R)$, augmented with a subset of V and a set of axioms. The language generated by γ is defined as $L(\gamma) = \sigma_2^*(A) \cap T^*$.

The power of extended H systems as well as some extensions of H systems have been studied extensively in the literature. In this paper, we are interested in two such extensions viz., communicating distributed H systems and two level H systems. We give the definitions of these systems in sections 2 and 3.

We denote by RE the family of recursively enumerable languages. A recursively enumerable language can be generated by a type-0 grammar $G = (N, T, S, P)$ where N is a set of non-terminals, $T \subseteq N$ is the set of terminal symbols, S is the start symbol, and P consists of productions of the form $u \rightarrow v, u, v \in (N \cup T)^*, |u|_N > 0$.

Notation: In the following sections, a splicing rule is represented by $x\#y\$a\#b$. However, while explaining the functionality of such a rule in the proofs, we represent them by $(x|y, a|b) \models (xb, ay)$.

2 Two-Level Distributed H Systems

Two-level distributed H systems were introduced in [2], [3]. In [2], two-level distributed systems were considered in the non-separated form, whereas in [3], separated systems were considered.

A two-level (non-separated) communicating distributed H system of degree $n, n \geq 1$ is a construct $\Gamma = (V, T, (w_1, A_1, I_1, E_1), \ldots (w_n, A_n, I_n, E_n))$, where V is the alphabet, $T \subseteq V$ is the terminal alphabet, $w_i \in V^*$, $A_i \subseteq V^*$, and

$I_i, E_i \subseteq V^*\#V^*\$V^*\#V^*$, for symbols $\#, \$$ not in V. All sets $A_i, I_i, E_i, 1 \leq i \leq n$ are finite; (w_i, A_i, I_i, E_i) is the ith component of the system; w_i is the active axiom, A_i is the set of passive axioms, I_i and E_i are the sets of internal and external splicing rules respectively.

The contents of a component i is described by a pair (x_i, M_i), where $x_i \in V^*$ is the active string and $M_i \subseteq V^*$ is the set of passive strings. An n-tuple $\pi = [(x_1, M_1), \ldots, (x_n, M_n)]$ is called a configuration of the system. For $1 \leq i \leq n$ and a given configuration π as above, we define $\mu(x_i, \pi) =$ external if there are $r \in E_i$ and $x_j, j \neq i$ such that $(x_i, x_j) \models_r (u, v)$ for some $u, v \in V^*$. Otherwise, $\mu(x_i, \pi)$ is internal.

For two configurations π, π' as above, we write $\pi \Rightarrow_{int} \pi'$ if the following conditions hold: (i) for all $i, 1 \leq i \leq n$, we have $\mu(x_i, \pi) =$ internal, (ii) for each $i, 1 \leq i \leq n$, either $(x_i, z) \models_r (x_i', z')$ for some $z \in M_i, z' \in V^*, r \in I_i$, and $M_i' = M_i \cup \{z'\}$, or (iii) no rule $r \in I_i$ can be applied to (x_i, z), for any $z \in M_i$, and then $(x_i', M_i') = (x_i, M_i)$.

The relation $\Rightarrow_{ext}$ defines an external splicing, and $\Rightarrow_{int}$ defines an internal splicing. In both cases, splicing is performed in parallel and all components not able to use a splicing rule do not change their contents. External splicing has priority over internal splicing and all operations have as their first term an active string; the first string obtained by splicing becomes the new active string of the component and the second string becomes an element of the set of passive strings of that component.

The language generated by a two-level distributed H system Γ is defined by $L(\Gamma) = \{w \in T^* \mid [(w_1, A_1), \ldots, (w_n, A_n)] \Rightarrow^* [(x_1, M_1), \ldots, (x_n, M_n)]\}$, for $w = x_1, x_i \in V^*, 2 \leq i \leq n$, and $M_i \subseteq V^*, 1 \leq i \leq n$. LDH_n denotes the family of languages generated by two level distributed H systems with utmost n components. If in the above, we consider all the sets E_i to be the same, i.e, if $E_i = E$, for all $1 \leq i \leq n$, then we get a separated two-level distributed H system. The family of languages generated by separated two-level distributed H systems with n components is denoted by $SLDH_n$. When no restriction is imposed on the number of components, n is replaced by $*$. In the following, we improve the universality result in [3, 4].

Theorem 1. *$RE = SLDH_n = LDH_n$ for all $n \geq 2$.*

Proof. The idea behind the proof is very close to the one used in [3] and the proof is much simpler. Consider a type-0 grammar $G = (N, T, S, P)$. We construct the $SLDH$ system $\Gamma = (V, T, (w_1, A_1, I_1), (w_2, A_2, I_2), E)$ with

$$V = N \cup T \cup \{X, Z, Z_s, Z_l, Z_r, C_1, C_2\},$$
$$w_1 = SXXC_1,$$
$$A_1 = \{ZvXZ_s \mid u \to v \in P\}$$
$$\cup \{ZXX\alpha Z_l, Z\alpha XXZ_r \mid \alpha \in N \cup T\}$$
$$I_1 = \{\#uXZ\$Z\#vXZ_s \mid u \to v \in P\}$$

(Replacing u by v simulating $u \to v$. Z_s is introduced after replacement)

$\cup \{\#\alpha XZ\$Z\#XX\alpha Z_l \mid \alpha \in N \cup T\}$
(shifting α to the right of X. Z_l is introduced after the right shift)
$\cup \{\#XZ\$Z\#\alpha XXZ_r \mid \alpha \in N \cup T\}$,
(shifting α to the left of X. Z_r is introduced after the left shift)

$w_2 = C_2Z$,

$A_2 = \{Z_sZ, Z_lZ, ZZ_r\}$,

$I_2 = \{C_2\#Z_s\$Z_s\#Z, C_2\#Z_l\$Z_l\#Z, C_2\#Z_r\$Z_r\#Z\}$,
(Changing Z_s, Z_l, Z_r back to Z to start a new simulation)

E consists of the rules

$E1$. $X\#X\$C_2\#Z, C_2\#Z\$X\#X$,
(First step while simulating a rule $u \to v$, or while shifting)

$E2$. $X\#Z_s\$C_2\#X, C_2\#X\$X\#Z_s$,
(Replace Z_s by a string ending in C_1 in w_1; replace the suffix of w_2 by Z_s)

$E3$. $XX\alpha\#Z_l\$C_2X\#, C_2\#X\$XX\alpha\#Z_l, \alpha \in N \cup T$,
(Replace Z_l by a string ending in C_1 in w_1; replace the suffix of w_2 by Z_l)

$E4$. $\alpha XX\#Z_r\$C_2X\alpha\#, C_2\#X\alpha\$\alpha XX\#Z_r, \alpha \in N \cup T$,
(Replace Z_r by a string ending in C_1 in w_1; replace the suffix of w_2 by Z_r)

$E5$. $\#XXC_1\$C_2Z\#$.
(To terminate, cut off the symbols XXC_1 from the right)

Component 1 simulates rules of P and also shifts symbols to the right and left of the marker XX. Component 2 saves the suffix of the active string w_1 that is cut while simulation and shifting, and also checks that the shifting done in component 1 is correct.

To start with, we have $w_1 = SXXC_1, w_2 = C_2Z$. In general, assume that $w_1 = z_1uXXz_2C_1, w_2 = C_2Z$, where $u = u'a$, where $a \in V, u \in V^*$. (initially, $z_1u' = \lambda, a = S, z_2 = \lambda$).

Case 1: Simulation of a rule $u \to v \in P$. To begin, $E1$ is the only applicable rule. $E1$ is applied in parallel to both components.

1. $E1 \Rightarrow w_1 = z_1uXZ, C_2Xz_2C_1 \in M_1$, $w_2 = C_2Xz_2C_1, z_1uXZ \in M_2$. In the next step, no external rules are applicable, since w_1 does not contain XX or XZ_s or XXC_1. Note that $E1$ cuts the suffix Xz_2C_1 of w_1 and appends it to w_2; it also cuts the suffix Z of w_2 and appends it to w_1.
2. Use the internal rule $(z_1|uXZ, Z|vXZ_s) \models (z_1vXZ_s, ZuXZ)$ in component 1 obtaining $w_1 = z_1vXZ_s, w_2 = C_2Xz_2C_1$, to simulate $u \to v$. Component 2 is idle. In the next step, only $E2$ is applicable, and it acts in parallel on both components.
3. Now, $E2 \Rightarrow w_1 = z_1vXXz_2C_1, w_2 = C_2Z_s$, re adjoining the suffix z_2C_1 to w_1. No external rules are applicable in the next step since $w_2 \neq C_2X, C_2Z$.
4. To get back to C_2Z, use the internal rule $(C_2|Z_s, Z_s|Z) \models (C_2Z, Z_sZ_s)$ in component 2 (component 1 is idle) giving $w_1 = z_1vXXz_2C_1, w_2 = C_2Z$.

Case 1 handles $w_1 = z_1uXXz_2C_1$, when there is a rule $u \to v \in P$. Assume now that for $u = u'a$, there exists no rule in P for a, but there exists $u' \to v' \in P$. To simulate $u' \to v'$ as above, we need u' to be adjacent to XX in w_1. To obtain this, we need to shift a to the right of XX obtaining $z_1u'XXaz_2C_1$. Case 2 handles this situation.
Case 2: Transforming $w_1 = z_1u'aXXz_2C_1$ into $z_1u'XXaz_2C_1$, given $w_2 = C_2Z$.

1. To begin, only $E1$ is applicable in both components. $E1 \Rightarrow w_1 = zu'aXZ, C_2Xz'C_1 \in M_1, w_2 = C_2Xz'C_1, zu'aXZ \in M_2$. In the next step, no external rules are applicable since w_1 does not contain XX, XZ_s, XC_1.
2. By assumption (since there is no rule in P for a), we choose any of the two internal rules (different from the one chosen in case 1, step 2). Component 2 will remain idle in this step. Using $(zu'|aXZ, Z|XXaZ_l) \models (zu'XXaZ_l, ZaXZ)$ in component 1, we obtain $w_1 = zu'XXaZ_l$, and $w_2 = C_2Xz'C_1$. $E3$ is only applicable in the next step, and it acts in parallel on both components.
3. Now, $E3 \Rightarrow w_1 = zu'XXaz'C_1, C_2XZ_l \in M_1, w_2 = C_2Z_l, zu'XXaXz'C_1 \in M_2$, shifting a to the right of XX in w_1. In the next step, no external rules are applicable, since $w_2 \neq C_2X, C_2Z$.
4. To get back to C_2Z, use the internal rule $(C_2|Z_l, Z_l|Z) \models (C_2Z, Z_lZ_l)$ in component 2 (component 1 remains idle) giving $w_1 = z_1u'XXaz_2C_1, w_2 = C_2Z$.

After cases 1 and 2, one more situation needs to be handled. Assume that we have $w_1 = z_1XXaz_2C_1$, with rules $z_1a \to z \in P$, and no rules in P for any substring of z_1. Clearly, case 2 is not useful, and to simulate a rule as in case1, we need z_1a to the left of XX. To do this, the a should be shifted to the left of XX obtaining $w_1 = z_1aXXz_2C_1$.
Case 3: Transforming $w_1 = z_1XXaz_2C_1$ into $z_1aXXz_2C_1$, given $w_2 = C_2Z$.

1. As in the above cases, we start with $E1$. $E1 \Rightarrow w_1 = z_1XZ, C_2Xaz_1'C_1 \in M_1, w_2 = C_2Xaz_1'C_1, z_1XZ \in M_2$. No external rules are applicable in the next step.
2. We can choose an internal rule in component 1 involving Z_l or Z_r, lets choose the one with Z_r. Using $(z_1|XZ, Z|\alpha XXZ_r) \models (z_1\alpha XXZ_r, ZXZ), \alpha \in N \cup T$ in component 1, we obtain $w_1 = z_1\alpha XXZ_r$. Component 2 is idle, and hence $w_2 = C_2Xaz_1'C_1$. $E4$ is only applicable in the next step to both components.
3. $E4 \Rightarrow w_1 = z_1aXXz_1'C_1, C_2XaZ_r \in M_1, w_2 = C_2Z_r, z_1aXXXaz_1'C_1 \in M_2$, shifting a to the left of XX. Note that $E4$ can be applied only if $\alpha = a$ in the previous step. No external rules are applicable in the next step since $w_2 = C_2Z_r$.
4. Using the internal rule $(C_2|Z_r, Z_r|Z) \models (C_2Z, Z_rZ_r)$ in component 2 (component 1 being idle), we obtain $w_1 = z_1aXXz_1'C_1, w_2 = C_2Z$.

Now, any of the three cases can be iterated. To terminate, we have only one choice: to remove the substring XX of w_1 which facilitates simulation of rules or shifting. This is done by using $E5$, when all symbols are to the left of XX. This will cut off from the active string $wXXC_1$ in component 1, the tail XXC_1,

leaving w as the active string. Now, no more rules can be applied to w. If $w \in T^*$, it gets listed in the language, otherwise, nothing is computed. □

Note the almost symmetric nature of the external rules in the above theorem. It helps in applying the external rules simultaneously in both components, with no waiting. Even when applying internal rules, there is a minimal wait of exactly one step for the other component.

3 Communicating Distributed H (CDH) Systems

CDH systems have been explored extensively in [1, 3, 5, 6], obtaining universality results with arbitrarily many components, six components, three components and nine components respectively. In the following section, we briefly recall the basics of CDH systems [4] and introduce the concept of laziness into CDH systems. We then consider an example and prove that universality can be obtained with 2 components.

A CDH system 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 sets of axioms, the sets of splicing rules, and the selector of the component i, respectively. Let $\mathcal{B} = V^* - \bigcup_{i=1}^{n} V_i^*$. The pair $\sigma^{(i)} = (V, R_i)$ is the underlying H scheme associated to the component i of the system.

An $n-$tuple $(L_1, L_2, \ldots, L_n), L_i \subseteq V^*$, is called a configuration of the system. The initial configuration of the system is $(A_1, \ldots, A_n)$. For two configurations $(L_1, \ldots, L_n), (L_1', \ldots, L_n')$, we define $(L_1, \ldots, L_n) \Rightarrow (L_1', \ldots, L_n')$ iff $L_i' = \bigcup_{j=1}^{n} ({\sigma_2^{(j)}}^*(L_j) \cap V_i^*) \cup ({\sigma_2^{(i)}}^*(L_i) \cap \mathcal{B})$, for each i, $1 \leq i \leq n$.

In words, the contents of each component are spliced according to the set of rules (we pass from L_i to ${\sigma_2^{(i)}}^*(L_i)$) and the result is redistributed among the n components according to the selectors $V_1, \ldots, V_n$; the part which cannot be redistributed remains in the component. As no conditions are imposed on the alphabets V_i, when a string in ${\sigma_2^{(j)}}^*(L_j)$ belongs to several languages V_i^*, then copies of the string will be distributed to all components i with this property.

The language generated by Γ is defined as $L(\Gamma) = \{w \in T^* \mid w \in L_1 \text{ for } L_1, \ldots, L_n \subseteq V^* \text{ such that } (A_1, \ldots, A_n) \Rightarrow^* (L_1, \ldots, L_n)\}$. The family of languages generated by communicating distributed H systems of degree utmost $n, n \geq 1$ is denoted by CDH_n. When n is not specified, then we replace n by $*$.

3.1 Introducing Laziness

Let Γ be a CDH_n system. We now define three kinds of strings viz., *active, passive* and *inactive* based on Γ as follows:

1. A string $w \in \sigma_2^{(i)^*}(L_i)$ is said to be *active* if there exists (i) a splicing rule $(w_2' \# w_2'' \$ a \# b)$ in R_i, (ii) a string $xaby$ in $\sigma_2^{(i)^*}(L_i)$, and (iii) a substring $w_2'w_2''$ of w. Note that we can also describe w having ab as a substring such that there exists a string $xw_2'w_2''y$ in $\sigma_2^{(i)^*}(L_i)$. Clearly, if w is an active string, it can be spliced using rules of R_i to obtain further strings.
2. A string $w \in \sigma_2^{(i)^*}(L_i)$ is said to be *passive* if for all splicing rules $(w_2' \# w_2'' \$ a \# b)$ in R_i, such that $w_2'w_2''$ (or ab) is a substring of w, there does not exist any string $xaby$ (or $x_1w_2'w_2''x_2$) in $\sigma_2^{(i)^*}(L_i)$.
3. A string $w \in \sigma_2^{(i)^*}(L_i)$ is said to be *inactive* if for all splicing rules $(w_2' \# w_2'' \$ a \# b)$ in R_i, w does not contain $w_2'w_2''$ or ab as a substring.

A *lazy* communicating distributed H system is a construct

$$\Gamma = (V, T, (A_1, R_1, V_1, \gamma_1), \ldots, (A_n, R_n, V_n, \gamma_n)),$$

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 tuple $(A_i, R_i, V_i, \gamma_i), 1 \leq i \leq n$, is called a component of Γ; A_i, R_i, V_i are the sets of axioms, the sets of splicing rules, and the selector of the component i, respectively; γ_i is a parameter taking values l or e, depending on whether the component is *lazy* or *eager*; T is the terminal alphabet of the system. Let $\mathcal{B} = V^* - \bigcup_{i=1}^{n} V_i^*$.

There are two kinds of components : *lazy* components and *eager* components. The two kinds of components differ in the way they communicate : *eager* components behave the same way as the components in a CDH system, whereas *lazy* components communicate only their *inactive* strings, provided they pass the necessary filters.

The pair $\sigma^{(i)} = (V, R_i)$ is the underlying H scheme associated to the component i of the system.

An $n-$tuple $(L_1, L_2, \ldots, L_n), L_i \subseteq V^*$, is called a configuration of the system. L_i is also called the contents of component i. The initial configuration of the system is $(A_1, \ldots, A_n)$. For two configurations $(L_1, \ldots, L_n), (L_1', \ldots, L_n')$, we define $(L_1, \ldots, L_n) \Rightarrow (L_1', \ldots, L_n')$ iff

1. $L_i' = \bigcup_{j=1}^{n}(S_j \cap V_i^*) \cup (\sigma_2^{(i)^*}(L_i) \cap \mathcal{B})$, for each eager $i, 1 \leq i \leq n$, and $S_j = \sigma_2^{(j)^*}(L_j)$ if j is eager, and $S_j \subseteq \sigma_2^{(j)^*}(L_j)$ is the set consisting of all *inactive* strings of $\sigma_2^{(j)^*}(L_j)$, if j is lazy.
2. $L_j' = \bigcup_{i=1}^{n}(S_i \cap V_j^*) \cup (S_j \cap \mathcal{B}) \cup (L_j \backslash S_j)$ for each lazy $j, 1 \leq j \leq n$, and $S_i = \sigma_2^{(i)^*}(L_i)$ if i is eager, and $S_j \subseteq \sigma_2^{(j)^*}(L_j)$ is the set of *inactive* strings of $\sigma_2^{(j)^*}(L_j)$, if j is lazy.

In words, the contents of a component i are spliced according to the associated set of rules, and,

– If i is eager, the result is redistributed among the n components according to the selectors $V_1, \ldots, V_n$; the part which cannot be redistributed (which does not belong to some $V_i^*, 1 \leq i \leq n$) remains in the component.

– If i is lazy, the subset of $\sigma_2^{(i)^*}(L_i)$ consisting of the *inactive* strings of $\sigma_2^{(i)^*}(L_i)$ is redistributed among the n components according to the selectors $V_1, \ldots, V_n$, and the part of the subset which cannot be redistributed remains in the component.

The language generated by Γ is defined by $L(\Gamma) = \{w \in T^* \mid w \in L_1$ for some $L_1, \ldots, L_n \subseteq V^*$ such that $(A_1, \ldots, A_n) \Rightarrow^* (L_1, \ldots, L_n)\}$.

We denote by LCDH_n the family of languages generated by lazy communicating distributed H systems of degree utmost $n, n \geq 1$. When n is not specified, we replace n by $*$.

Note that an LCDH system with all components *eager* is the same as a CDH system. Let us consider an example.

Example 1. Consider the system Γ

$$(\{a,b,c,X,Y,Z,Z',F_1,F_2,F\},\{a,b,c\},(A_1,R_1,V_1,e),(A_2,R_2,V_2,l),(A_3,R_3,V_3,l)),$$

$$\begin{aligned}
&A_1 = \{XY, aX\}, R_1 = \{a\#X\$X\#Y, c\#F\$\#aY, c\#F\$aY\#\}, V_1 = T \cup \{F\},\\
&A_2 = \{bZ, Z'Z'\}, R_2 = \{a\#Y\$\#bZ, ab\#Z\$Z'\#Z'\}, V_2 = T \cup \{Y\},\\
&A_3 = \{F_1cF_1, F_2, FF\}, R_3 = \{b\#Z'\$F_1\#cF_1, c\#F_1\$\#F_2, c\#F_2\$F\#F\},\\
&V_3 = T \cup \{Z'\}.
\end{aligned}$$

No communication between components is possible before any splicing, since $A_1 \cap V_j^* = \emptyset, j = 2, 3$; strings of A_2 are *passive*; $FF \in A_3$ is *passive*, and the rest of A_3 is *active*. Hence, splicing is possible only in the first and third components; $(a|X, X|Y) \models (aY, XX)$ in the first component and $(F_1c|F_1, |F_2) \models (F_1cF_2, F_1)$ in the third component. The string aY is communicated from component 1 to component 2, and in component 3, the string F_1 is a candidate for communication, since it is *inactive*. However, $F_1 \notin V_i^*, i = 1, 2, 3$, and hence remains in component 3.

In component 2, the string aY is spliced according to the rule $(a|Y, |bZ) \models (abZ, Y)$ and in component 3, the new splicings are $(F_1c|F_2, F|F) \models (F_1cF, FF_2)$ or $(F_1c|F_1, F_1c|F_2) \models (F_1cF_2, F_1cF_1)$. The string abZ in component 2 is *active*, whereas Y is *inactive*. Similarly, in component 3, the string F_1cF is *inactive*. Therefore, Y, F_1cF are candidates for communication in components 2,3. However, since $Y \notin V_1, V_3$, $F_1cF \notin V_1^*, V_2^*, V_3^*$, Y remains in component 2 and F_1cF in component 3. Continuing with abZ in component 2, we obtain $(ab|Z, Z'|Z') \models (abZ', Z'Z)$ or $(a|Y, a|bZ) \models (abZ, aY)$. Now the strings $abZ', Z'Z$ are *inactive* and therefore are candidates for communication. Of the two, abZ' is sent to component 3, while $Z'Z$ remains in component 2.

In component 3, abZ' is spliced as $(ab|Z', F_1|cF_1) \models (abcF_1, F_1Z')$. Now, F_1Z' is *inactive*; however since it does not belong to any V_i^*, it remains in component 3. The string $abcF_1$ is spliced as $(abc|F_1, |F_2) \models (abcF_2, F_1)$. Now, $abcF_2$ is *active*, and F_1 is *inactive*. F_1 remains in component 3 since it fails all filters, and we splice $abcF_2$. Some possible splicings are $(F_1c|F_1, abc|F_2) \models (F_1cF_2, abcF_1)$ or $(abc|F_1, abc|F_2) \models (abcF_2, abcF_1)$ or $(abc|F_2, F|F) \models (abcF, FF_2)$. All strings except $abcF$ are *active*, and $abcF$ is communicated to component 1.

In component 1, we have either the option of appending an aY to abc and thus continuing, or using $(abc|F, aY|) \models (abc, aYF)$. The string abc remains in component 1, and a copy is sent to components 2 and 3. Clearly, $L(\Gamma) = \{(abc)^n \mid n \geq 1\}$.

Theorem 2. $RE = LCDH_2$

Proof. Consider a type-0 grammar $G = (N, T, S, P)$. Let $N \cup T \cup \{B\} = \{D_1, \ldots, D_m\}$, where B is a new symbol. Since $N, T \neq \emptyset$, $m \geq 3$. Construct the LCDH system $\Gamma = (V, T, (A_1, R_1, V_1, e), (A_2, R_2, V_2, l))$, with

$V = N \cup T \cup \{X, Y, Z, Z', E_1, E_2, X_i, Y_i, X'_{2j}, Y'_{2j} \mid -1 \leq i \leq 2m, 1 \leq j \leq m\}$,

$A_1 = \{XBSY\} \cup \{ZvY \mid u \to v \in P\} \cup \{ZX'_{2i}Y'_{2i} \mid 1 \leq i \leq m\} \cup \{E_1E_2, XZ_0, Z_0Y\}$
$\cup \{X_{2i}Z, ZY_{2i} \mid 1 \leq i \leq m\}$, and R_1 consists of the following rules:

Simulating rules of P :

1. $\#uY\$Z\#vY,\ u \to v \in P$,

Rotation : For $1 \leq i, j, k \leq m$,

2. $D_j\#D_iY\$ZX'_{2i}\#Y'_{2i}$,
3. $X\#D_jD_k\$ZX'_{2i}D_i\#Y$,
4. $\#XY\$Z\#X'_{2i}D_iD_j$,

Updation of Indices (Odd to even) :

5. $X_{2j+1}\#D_i\$X_{2j}\#Z,\ 0 \leq j \leq m, 1 \leq i \leq m$,
6. $D_i\#Y_{2j+1}\$Z\#Y_{2j},\ 0 \leq j \leq m, 1 \leq i \leq m$,

Going back to end markers $X, Y,$ from X_0, Y_0

7. $X_0\#D_j\$X\#Z_0, 1 \leq j \leq m$,
8. $D_j\#Y_0\$Z_0\#Y, 1 \leq j \leq m$,

Possible Termination : For $D_j, D_k \in T, 1 \leq j, k \leq m$,

9. $D_j\#BY\$E_1\#E_2$,
10. $X\#D_k\$E_1\#BY$,
11. $E_1\#D_k\$\#E_1E_2$,
12. $D_j\#E_2\$E_1E_1E_2\#$,

$V_1 = N \cup T \cup \{B, X, Y, X_0, Y_0\} \cup \{X_{2i+1}, Y_{2i+1} \mid 0 \leq i \leq m-1\}$,

$A_2 = \{X_{2i}Z', Z'Y_{2i}, Z'Y_{2i-1}, X_{2i-1}Z', X_{-1}Z', Z'Y_{-1}\},\ 0 \leq i \leq m,$ and

R_2 consists of rules

Initialize : For $1 \leq i, j \leq m$,

13. $X'_{2i}\#D_i\$X_{2i}\#Z'$,
14. $D_j\#Y'_{2i}\$Z'\#Y_{2i}$,

Updation of Indices (Even to odd) : For $1 \leq i, j \leq m$,

15. $X_{2i}\#D_j\$X_{2i-1}\$Z'$,
16. $D_j\#Y_{2i}\$Z'\#Y_{2i-1}$,

Removal of X_0, Y_0 : For $1 \leq j \leq m$,

17. $X_0\#D_j\$X_{-1}\#Z'$,

18. $D_j\#Y_0\$Z'\#Y_{-1}$.

$V_2 = N \cup T \cup \{B, X_{2i}, Y_{2i}, X'_{2j}, Y'_{2j} \mid 0 \leq i \leq m, 1 \leq j \leq m\}$

Let us examine the work of Γ. The underlying idea is to rotate and simulate. We start from the string $XBSY$ in component 1, and in component 2, there are no rules that can be applied with respect to strings in A_2. However since all strings in A_2 are *passive*, and since none of the strings in A_1 pass the filter V_2, there is no communication before any splicing. In the first component, we can simulate rules of P by using the rule 1, replacing suffixes. Since the new strings obtained as a result of rule 1 do not pass the filter V_2, and since there are no *inactive* strings in component 2, there is no communication between the components.

This can go on as long as rule 1 is applied. If we choose to rotate a symbol at any point of time, then we choose rule 2, giving $(Xw|D_iY, ZX'_{2i}|Y'_{2i}) \models (XwY'_{2i}, ZX'_{2i}D_iY)$. Both of these strings $\notin V_2^*$, and hence cannot be communicated to component 2. We can choose next, $(X|D_j, ZX'_{2i}D_i|Y) \models (XY, ZX'_{2i}D_iD_jw_1Y'_{2i})$, provided $w = D_jw_1$. The two new strings obtained here also $\notin V_2^*$ and hence we continue in component 1. We can now use $(|XY, Z|X'_{2i}D_iDj) \models (X'_{2i}D_iD_jw_1Y'_{2i}, ZXY)$, and in this step, the string $X'_{2i}D_iD_jw_1Y'_{2i}$ is communicated to component 2. No string from component 2 is communicated to component 1.

In the next step, in component 2, we can use rules 13 or 14 to $X'_{2i}D_iD_jw_1Y'_{2i}$, resulting in $(X'_{2i}Z', X_{2i}D_iD_jw_1Y'_{2i})$ or $(X'_{2i}D_iD_jw_1Y_{2i}, Z'Y'_{2i})$. The strings $X'_{2i}Z', Z'Y'_{2i}$ in component 2 are *inactive*, and so are considered for communication. However, since they do not pass V_1, they remain in component 2. The strings $X'_{2i}D_iD_jw_1Y_{2i}$ or $X_{2i}D_iD_jw_1Y'_{2i}$ are *active* and so are not considered for communication. We can apply rule 14 or 15 to $X_{2i}D_iD_jw_1Y'_{2i}$ and rule 13 or 16 to $X'_{2i}D_iD_jw_1Y_{2i}$. In either case, we ultimately obtain the *inactive* string $X_{2i-1}D_iwY_{2i-1}$. Since $X_{2i-1}D_iwY_{2i-1} \in V_1^*$, it is sent to component 1.

Let $w' = D_iw$. In component 1, rules 5 and 6 are applicable to $X_{2i-1}w'Y_{2i-1}$. If we choose rule 5 first, we obtain $(X_{2i-1}|D_i, X_{2i-2}|Z) \models (X_{2i-1}Z, X_{2i-2}w'Y_{2i-1})$. Both these strings cannot be communicated to component 2, since they do not pass the filter. We continue with rule 6 to obtain $(D_k|Y_{2i-1}, Z|Y_{2i-2}) \models (X_{2i-2}w'Y_{2i-2}, ZY_{2i-1})$. We would obtain the same set of strings even if rule 6 is applied first. The string $X_{2i-2}w'Y_{2i-2}$ obtained after application of rules 5,6 is communicated to component 2, since it passes the filter.

In component 2, we now decrement the end markers using rules 15, 16. Observe that until both are used, we cannot communicate the intermediate string ($X_{2i-2}w'Y_{2i-3}$ or $X_{2i-3}w'Y_{2i-2}$), since it is *active*. The other strings obtained as a result of rules 15,16 are $X_{2i-2}Z', Z'Y_{2i-2}$, which cannot be communicated even though they are *inactive*, since they are not over V_1^*.

Continuing like this, a string $X_1w'Y_1$ is communicated to component 1. Now, using rules 5,6 as before we decrement X_1, Y_1 to X_0, Y_0. Note that before decrementing both X_1 and Y_1, we cannot communicate to component 2, since V_2 does not contain $X_{2i+1}, Y_{2i+1}, i \geq 0$. However, when we have $X_0w'Y_0$ in component 1, since V_1, V_2 contain X_0, Y_0, the string is communicated to component 2, and a copy is retained in component 1.

In component 1, the X_0 is replaced by X and Y_0 by Y by rules 7 and 8. Observe that the intermediate strings obtained (with X, Y_0 and X_0, Y as the end markers) cannot be communicated to component 2, since $X, Y \notin V_2$. But, we can start another simulation in component 1 using $Xw'Y$. Simultaneously, in component 2, rules 17,18 are applicable to $X_0w'Y_0$. We do not consider the intermediate strings for communication since they are *active*. But, even after application of 17,18, the string we obtain, viz., $X_{-1}w'Y_{-1}$ cannot be communicated, since it is not over V_1.

Note that, the first time a rotation is done in component 1, the indices of the end markers will be the same, since rule 3 can be applied only after applying rule 2, thus obtaining the correct string $ZX'_{2i}D_iY$. However, this is not the case for subsequent rotations. (since all strings $ZX'_{2i}D_iY$ produced in previous steps will be available). In general, it is possible to obtain a string $X'_{2i}w'Y'_{2j}, i \neq j$ in component 1. We communicate this string to component 2, and, after a sequence of communications, we will end up with a string $X_0w'Y_{2l}, l > 0$ or $X_{2k}w'Y_0, k > 0$. Let us examine how to handle this case.

Let us assume that we have the string $X_{2k}w'Y_0$ in component 1. Obviously, this is obtained after application of rules 5,6 in the two previous steps. This string is communicated to component 2 since it passes the filter V_2, without retaining a copy in component 1 ($X_{2k}w'Y_0 \notin V_1^*$). In component 2, rules 15,18 are applicable. This leads us to the intermediate strings $X_{2k-1}w'Y_0$ (15 applied first) or $X_{2k}w'Y_{-1}$ (18 applied first). In either case, both strings are *active*. We end up, in either case with $X_{2k-1}w'Y_{-1}$, which is *inactive*. But however, this string belongs to neither V_1^* nor V_2^* and so, remains in component 2, without contributing to the output.

Thus, we can continue a simulation iff we end up with $X_0w''Y_0$ in component 1, in which case, the copy sent to component 2 remains stuck there, but the copy in component 1 is useful by replacing X_0 by X and Y_0 by Y.

Let us now examine how a string over terminals can be generated, contributing to $L(\Gamma)$. Assume that we have in component 1, a string $XwBY$. We can choose to either rotate B using rule 2, or eliminate B using rule 9. Let us see what happens if rule 9 is chosen. We obtain $(Xw|BY, E_1|E_2) \models (XwE_2, E_1BY)$. Both these strings cannot be communicated, since they fail to pass the filter V_2. We can continue with rule 10, $(X|D_k, E_1|BY) \models (XBY, E_1D_kw'E_2)$, provided $w = D_kw'$. Now, rule 11, $(E_1|D_k, |E_1E_2) \models (E_1E_1E_2, D_kw'E_2)$ is used to remove E_1. This is followed by application of rule 12 removing E_2 and obtaining D_kw'. The only information we have about this string is that if $w' = w_1D_j$ or $w = D_kw_1D_j$, then $D_k, D_j \in T$. However, if this string is not over terminals, then it does not contribute to the language and is hence "lost". Thus, only terminal strings obtained starting from $XBSY$, which are rotated correctly every time (so that X_0wY_0 is obtained in component 1) can contribute to the language. □

Remark 1. To see how the above system communicates less, we will examine what happens if component 2 was eager in the above result. As long as no rotation takes place (for the first time) in component 1, there is no communication between components, irrespective of the nature of the individual components.

The number of strings communicated between components is the same (if component 2 is eager or lazy) even after rotation, in case, rotation takes place correctly, yielding X_0wY_0 in component 1. Now assume that rotation goes wrong, giving X_0wY_{2k} or $X_{2l}wY_0$, $k, l > 0$ in component 1. Either of these strings will be communicated to component 2. If component 2 was eager, then if rule 15 or 16 is chosen first, we get a string $X_{2l-1}wY_0$ or X_0wY_{2k-1}, which will be communicated to component 1, leading to wrong results. That means an extra communication is made, which also leads to wrong results in case component 2 was eager. But if component 2 is lazy, this communication will not be made, and the results also do not go wrong. The same is the case if X_0, Y_0 are not replaced in subsequent steps in component 1, when having X_0wY_0. ($X_0w'Y'_{2i}$ can be obtained in component 1, which will be communicated to component 2. Component 2 if eager, can then communicate $X_0w'Y_{2i-1}$ to component 1, and things go wrong).

4 Conclusion

We have improved the universality result of two-level distributed H systems, and conjecture that the result obtained is optimal. Likewise, by introducing laziness, we have proved that a better characterization of RE can be obtained, as compared to the result $CDH_3 = RE$ [5]. The power of $LCDH_2$, with both components being lazy, is open.

References

1. E. Csuhaj-Varju, L. Kari, Gh. Păun, Test tube distributed systems based on splicing, *Computers and AI*, 15, 5 (1996), 419–436.
2. Gh. Păun, Two-level distributed H systems, *Proc. of the Third Conf. on Developments in Language Theory*, Aristotle Univ. of Thessaloniki, 1997, 309-327.
3. Gh. Păun, DNA Computing: Distributed splicing systems, *Structures in Logic and Computer Science : A Selection of Essays in Honor of A. Ehrenfeucht*, LNCS 1261, 1997, 351–370.
4. Gh. Păun, G. Rozenberg, A. Salomaa, DNA Computing : New Computing Paradigms, Springer, 1998.
5. L. Priese, Y. Rogozhin, M. Margenstern, Finite H systems with 3 tubes are not predictable, *Pacific Symposium on Biocomputing*, Hawaii, 1998 (R. B. Altman, A. K. Dunker, L. Hunter, T. E. Klein, eds), World Sci, Singapore, 1998, 547-558.
6. C. Zandron, C. Ferretti, G. Mauri, A reduced distributed splicing system for RE languages, *New Trends in Formal Languages : Control, Cooperation, Combinatorics*, LNCS 1218, 1997, 346-366.