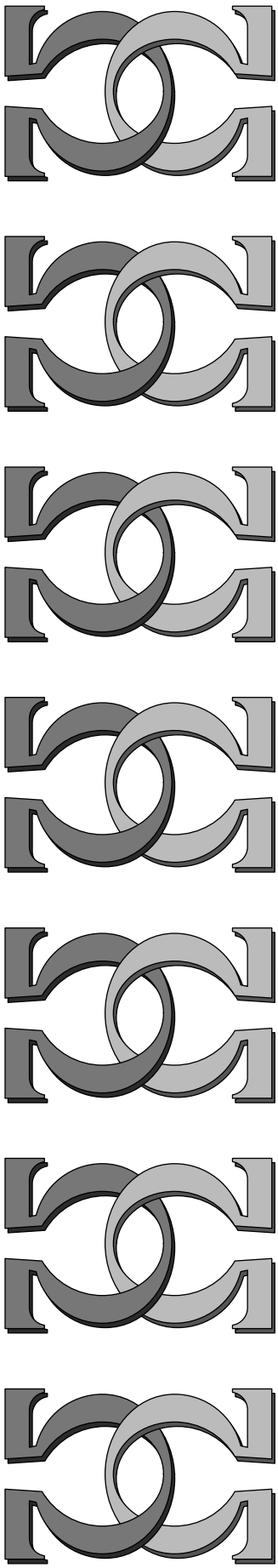

CDMTCS
Research
Report
Series

Cooperating Distributed Splicing Systems

Carlos Martin-Vide
Rovira i Virgili University

Gheorghe Păun
Institute of Mathematics of the Romanian Academy

CDMTCS-071
December 1997

Centre for Discrete Mathematics and
Theoretical Computer Science

Cooperating Distributed Splicing Systems

Carlos MARTIN-VIDE
Research Group on Mathematical Linguistics and Language Engineering (GRLMC)
Rovira i Virgili University
Pl. Imperial Tàrraco 1, 43005 Tarragona, Spain
E-mail: cmv@nil.fut.es

Gheorghe PĂUN
Institute of Mathematics of the Romanian Academy
PO Box 1 – 764, 70700 Bucureşti, Romania
E-mail: gpaun@imar.ro

Abstract. We introduce a new class of cooperating distributed H systems which consist of a given set of splicing systems (sets of splicing rules plus sets of axioms), similar in form to the cooperating distributed grammar systems. By applying iteratively the components of such a system (starting from a given initial string), in a sequence which runs nondeterministically, in such a way that a step is considered correctly finished only if no more splicing is possible, we obtain a language. Somewhat surprisingly if we take into account the loose control on the operations we carry out, a characterization of recursively enumerable languages is obtained, by mechanisms as above with only three components. We also characterize the recursively enumerable languages by cooperating distributed H systems with the components containing at most three splicing rules (in this case the number of components is no longer bounded).

KEYWORDS: Formal languages, Grammar systems, DNA computing, H systems

1 Introduction

The splicing operation has been introduced in [8] as a formal model of the recombinant behavior of DNA molecules under the influence of restriction enzymes and ligases. The passing from the biochemical reaction of recombination to the abstract operation of splicing is described in [9]. Informally speaking, two DNA sequences are cut by two restriction enzymes and the fragments are recombined such that possibly new sequences are produced. The sites where the enzymes can cut are encoded as pairs

$(u_1, u_2), (u_3, u_4)$, and the fact that they produce matching ends is represented by forming the quadruple $((u_1, u_2), (u_3, u_4))$. We say that this is a *splicing rule*. An *H system* is a generative device consisting of a set of axioms (initial strings) and a set of splicing rules. By an iterated application of these rules, starting from the axioms, we get a language. If also a terminal alphabet is provided and only strings on that alphabet are accepted, then we get the notion of an extended H system (introduced in [16]).

If only a finite set of rules are used, then even starting from a regular set of axioms, we can generate only regular languages. This has been proved in [5]; a simpler proof can be found in [18]. When using extended H systems, we obtain a characterization of regular languages ([16]).

If the set of splicing rules is a regular language (each rule $((u_1, u_2), (u_3, u_4))$ is written as a string $u_1 \# u_2 \$ u_3 \# u_4$, hence the set of rules is a language), then extended H systems (with finite sets of axioms) characterize the recursively enumerable languages ([11]).

However, working with infinite sets of rules, even regular, is not of much practical interest. Finite sets of rules give only regular languages, hence they stop at the level of finite automata/regular Chomsky grammars. It is therefore necessary to supplement the model with a feature able to increase its power. Many suggestions about how this can be done come both from the regulated rewriting area in formal language theory, see, e.g., [6], and from the very proof in [11]. Many characterizations of recursively enumerable languages by means of H systems with finite sets of splicing rules applied in a controlled way were obtained. Surveys of results of this type can be found in [12], [17].

Here we follow another idea, suggested by the distributed architectures in grammar systems area ([3]): we consider a given set of H systems (with finite sets of axioms and of splicing rules) and we apply them iteratively (starting with a special initial string), in a sequence which is not prescribed in advance. Applying an H system means to use its rules as much as it is possible for splicing the string currently available (produced at the previous step of the computation) with the axioms of the active system (this corresponds to the maximal mode of derivation in cooperating distributed grammar systems, [2], [3]). Moreover, we look for "computations", that is for the evolution of a given input string, when processed as above. This suggests to take only one output of the splicing operation (which is therefore defined as a ternary relation) and to use it as an input for the next splicing. (This can be interpreted as a higher reactivity of the strings obtained by splicing, in comparison with the strings provided as axioms – which are always present in the test tube.) Although it seems to be weak, these conditions on the correct use of our "operators" are enough in order to provide computational completeness: a characterization of recursively enumerable languages is obtained by systems as above composed of three components only.

This reminds the similar characterizations of recursively enumerable languages (but not by systems with three components) which are obtained for other types of distributed H systems (see [4], [14]). However, the model considered here has some features which make it more realistic from the control point of view: the components are nondeterministically enabled and there is no communication among them (for instance, by

transmitting strings from one to another) different from the communication carried out through the strings on which they work (this is similar to the blackboard model in distributed AI, see [10].)

The use of distributed architectures in the splicing area is also motivated from a biochemical point of view: a splicing rule corresponds to two restriction enzymes producing sticky ends which match. It is known that in general several restriction enzymes cannot work together, each one requests specific reaction conditions. Thus, it is natural to try to compute by using H systems which are as small as possible (as the number of splicing rules). Also a result of this type is given: cooperating distributed H systems with each component containing at most three splicing rules are computationally complete, they also characterize the recursively enumerable languages. No bound is obtained in this case on the number of components.

These results could suggest that a trade-off exists between the two parameters, the number of components and the number of splicing rules in each component. Actually, such a trade-off is not true for arbitrary values of the two parameters, it is possible to bound both of them: the proofs in the following sections start from type-0 grammars and effectively construct cooperating distributed H systems simulating them; if we start from a universal type-0 grammar, then a fixed cooperating distributed H system is obtained (in an effective way) which is also universal. Just changing its axiom we can generate any given recursively enumerable language. Changing the axiom does not change the two parameters mentioned above, hence they are simultaneously bounded and still a characterization of recursively enumerable languages is obtained.

2 Formal Language Prerequisites

We fix here some notations and we introduce the cooperating distributed (CD) grammar systems, the grammatical counterpart of the architectures we will later extend to H systems.

For an alphabet V, by V^* we denote the free monoid generated by V under the operation of concatenation; the the empty string is denoted by λ. Two languages are considered equal if they differ by at most the empty string.

A Chomsky grammar is denoted by $G = (N, T, S, P)$, where N is the nonterminal alphabet, T is the terminal alphabet, $S \in N$ is the axiom, and P is the set of rewriting rules; the rules are written in the form $u \to v$. The language generated by G is denoted by $L(G)$. The families of finite, regular, linear, context-free, context-sensitive, and recursively enumerable languages are denoted by *FIN, REG, LIN, CF, CS, RE*, respectively. By MAT and $ET0L$ we denote the families of languages generated by matrix grammars with context-free rules (without using appearance checking) and by extended tabled interactionless Lindenmayer systems, respectively (in both cases, λ-rules are allowed).

For other elements of formal language theory we refer to [19].

A *CD grammar system* of degree $n, n \geq 1$, is a construct

$$\Gamma = (N, T, S, P_1, \ldots, P_n),$$

where N, T are disjoint alphabets, $S \in N$, and $P_1, \ldots, P_n$ are finite sets of context-free rules over $N \cup T$ (the elements of N are interpreted as nonterminals, those of T as terminals).

For $x, y \in (N \cup T)^*$ and $1 \leq i \leq n$ we write $x \Longrightarrow_i y$ when $x = x_1 A x_2, y = x_1 z x_2$ for some $A \to z \in P_i$ (a usual direct derivation step using a rule in P_i). We denote by $\Longrightarrow_i^{=k}$, $\Longrightarrow_i^{\leq k}$, $\Longrightarrow_i^{\geq k}$, $\Longrightarrow_i^*$ a derivation consisting of exactly k steps as above, at most k steps, at least k steps, an arbitrary number of steps, respectively. We also write

$$x \Longrightarrow_i^t y \text{ iff } x \Longrightarrow_i^* y \text{ and there is no } z \text{ such that } y \Longrightarrow_i z$$

(a maximal derivation in the component P_i).

Denote $D = \{*, t\} \cup \{\leq k, = k, \geq k \mid k \geq 1\}$. For $f \in D$ we define

$$L_f(\Gamma) = \{w \in T^* \mid S \Longrightarrow_{i_1}^f w_1 \Longrightarrow_{i_2}^f w_2 \Longrightarrow_{i_3}^f \ldots \Longrightarrow_{i_m}^f w_m = w, \\ m \geq 1,\ 1 \leq i_j \leq n,\ 1 \leq j \leq m\}.$$

The family of languages generated by (λ-free) CD grammar systems with at most n components working in the f mode is denoted by $CD_n(f)$. When the number of components is not restricted, we write $CD_\infty(f)$.

Proofs of the following relations can be found in [2], [3]:

Theorem 1. (i) $CD_\infty(f) = CF,\ f \in \{*, = 1, \geq 1\} \cup \{\leq k \mid k \geq 1\}$.
(ii) $CF = CD_1(f) \subset CD_2(f) \subseteq CD_3(f) \subseteq \ldots \subseteq CD_\infty(f) \subseteq MAT,$
$f \in \{= k, \geq k \mid k \geq 2\}$.
(iii) $CF = CD_1(t) = CD_2(t) \subset CD_3(t) = CD_\infty(t) = ET0L$.

3 Splicing Systems

We now introduce the basic variant of H systems, those without restrictions on the splicing operation.

Let us consider an alphabet V and two special symbols, $\#, \$$, not in 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 $r = u_1 \# u_2 \$ u_3 \# u_4$ and three strings $x, y, w \in V^*$ we write

$$(x, y) \vdash_r w \text{ iff } x = x_1 u_1 u_2 x_2,\ y = y_1 u_3 u_4 y_2, \\ w = x_1 u_1 u_4 y_2, \text{ for some } x_1, x_2, y_1, y_2 \in V^*.$$

We say that we *splice* the strings x, y at the *sites* $u_1 u_2, u_3 u_4$, respectively.

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 a splicing scheme $\sigma = (V, R)$ and a language $L \subseteq V^*$ we define

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

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.)

We denote by $EH(F_1, F_2)$ the family of languages generated by extended H systems $\gamma = (V, T, A, R)$, with $A \in F_1, R \in F_2$, where F_1, F_2 are two given families of languages. (Note that R is a language, hence the definition makes sense.)

Two basic results concerning the power of extended H systems are the following.

Theorem 2. ([5], [18]) $EH(FIN, FIN) = EH(REG, FIN) = REG$.

Theorem 3. ([11]) $EH(FIN, REG) = RE$.

4 Cooperating Distributed H Systems

The way of working in a CD grammar system can be considered also for H systems. Taking into consideration the nondeterministic behavior of restriction enzymes, we introduce here only the maximal mode of derivation (we asume that the use of a set of splicing rules ends when no further splicing is possible). From a mathematical point of view, also the other derivation modes can be considered (especially $= k, \geq k$ are probably interesting, like in the case of grammars), but we do not investigate them here.

An extended *cooperating distributed* (in short, CD) *H system* (of degree $n, n \geq 1$) is a construct

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

where V is an alphabet, $T \subseteq V$, $w \in V^*$, A_i is a finite subset of V^*, R_i is a finite subset of $V^*\#V^*\$V^*\#V^*$, $1 \leq i \leq n$.

V is the alphabet of Γ, T is the terminal alphabet, w is the axiom, (A_i, R_i) is called a *component* of the system; A_i is the set of axioms of the component i, R_i is the set of splicing rules of the component i, $1 \leq i \leq n$.

The work of Γ consists of iterated applications of the splicing schemes $\sigma_i = (V, R_i)$ to pairs $(x, z), (z, x)$ for $x \in A_i$, where z is the string obtained at the previous step, taking w as the starting string; moreover, the use of σ_i is *maximal* in the sense that we stop using it only if no further splicing is possible. Formally, for $x, y \in V^*$ and $i \in \{1, 2, \ldots, n\}$, we define

$$\begin{aligned}
x \to_i y \quad &\text{iff} \quad (x, z) \vdash_r y \text{ or } (z, x) \vdash_r y, \text{ for some } z \in A_i, r \in R_i, \\
x \to_i^* y \quad &\text{iff} \quad x = y, \text{ or } x = x_0 \to_i x_1 \to_i \ldots \to_i x_k = y, k \geq 1, x_j \in V^*, 1 \leq j \leq k, \\
x \Longrightarrow_i y \quad &\text{iff} \quad x \to_i^* y \text{ and there is no } u \in V^* \text{ such that } y \to_i u.
\end{aligned}$$

The language generated by Γ is defined by

$$\begin{aligned}
L(\Gamma) \quad = \quad & \{x \in T^* \mid w \Longrightarrow_{i_1} w_1 \Longrightarrow_{i_2} \ldots \Longrightarrow_{i_m} w_m, \\
& x = w_m, m \geq 1, w_j \in V^*, 1 \leq i_j \leq n, 1 \leq j \leq m\}.
\end{aligned}$$

Note that, although we work systematically on strings which are obtained from the axiom w, we do not have here multisets (sets with a specified number of copies of each element); each string is assumed to appear in an arbitrary number of copies. This is especially important for the sets $A_i, 1 \leq i \leq n$, which are continuously available in their initial form.

We denote by $CDEH_n$ the family of languages generated by extended cooperating distributed H systems of degree at most $n, n \geq 1$, and by $CDEH_\infty$ the union of all these families.

Let us consider a simple **example**: for the system

$$\Gamma = (\{a, b, c, c', d, d', e\}, \{a, b, c, d\}, cabd, (A_1, R_1), (A_2, R_2)),$$

with

$$\begin{aligned}
A_1 &= \{c'ae, ebd'\}, \qquad & R_1 &= \{c'a\#e\$c\#,\ \#d\$e\#bd'\}, \\
A_2 &= \{ce, ed\}, & R_2 &= \{c\#e\$c'\#,\ \#d'\$e\#d\},
\end{aligned}$$

we obtain

$$L(\Gamma) = \{ca^n b^n d \mid n \geq 1\}.$$

Indeed, it is easy to see that the only correct derivations $\Longrightarrow_i$ consist of applying both rules in R_i at the ends of the current string; thus, these relations are

$$\begin{aligned}
&ca^m b^m d \Longrightarrow_1 c'aa^m b^m bd', \text{ for } m \geq 1, \\
&c'a^m b^m d' \Longrightarrow_2 ca^m b^m d, \text{ for } m \geq 1.
\end{aligned}$$

Remark 1. The previous CD H system can also be considered as working in one of the $= 2$ or ≥ 2 modes and the generated language is the same. This proves that, as we mentioned above, these modes of derivation are of interest also for CD H systems: in these modes, CD H systems with only two (finite) components can generate non-regular languages.

5 The Power of Cooperating Distributed H Systems

We are here interested in the size of the families $CDEH_n, n \geq 1$.

We start with some preliminary results:

Lemma 1. (i) $CDEH_1 \subseteq CDEH_2 \subseteq \ldots \subseteq CDEH_\infty \subseteq RE$. (ii) $REG \subseteq CDEH_1$. (iii) $REG \subset CDEH_2$.

Proof. (i) These inclusions follow directly from the definitions.

(ii) The extended H system in the proof of the inclusion $REG \subseteq EH(FIN, FIN)$ in [16] works in the maximal mode as defined here (it simulates a regular grammar and stops when no nonterminal is present in the current string).

(iii) The strictness of this inclusion is proved by the example at the end of the previous section. □

The hierarchy in point (i) above is not infinite: it collapses at the third level.

Theorem 4. $RE = CDEH_3$.

Proof. We prove only the inclusion $\subseteq$. The opposite inclusion can be proved by a straightforward construction of a type-0 grammar simulating a CD H system, or we can invoke the Church-Turing thesis.

Consider a type-0 Chomsky grammar $G = (N, T, S, P)$. Consider a new symbol B and assume that

$$N \cup T \cup \{B\} = \{\alpha_1, \ldots, \alpha_n\},$$

with $B = \alpha_1$; because both N and T are nonempty, we have $n \geq 3$.

We construct the extended cooperating distributed H system

$$\Gamma = (V, T, w, (A_1, R_1), (A_2, R_2), (A_3, R_3)),$$

where

$$\begin{aligned} V &= N \cup T \cup \{X, X', \overline{X}, Y, Y', Y'', \overline{Y}, Z, B, C\}, \\ w &= XBSY, \\ A_1 &= \{ZvY \mid u \to v \in P\} \\ &\cup \{ZC^iY' \mid 1 \leq i \leq n\} \end{aligned}$$

$$
\begin{aligned}
&\cup \ \{ZY'', XZ, \overline{X}Z\},\\
R_1 &= \{\#uY\$Z\#vY \mid u \to v \in P\}\\
&\cup \ \{\#\alpha_i Y\$Z\#C^iY' \mid 1 \le i \le n\}\\
&\cup \ \{\#CY'\$Z\#Y'',\ X'\#\$X\#Z,\ \overline{X}\#\$\overline{X}\#Z,\ \#\overline{Y}\$XZ\#\},\\
A_2 &= \{X'CZ, ZY', ZY\},\\
R_2 &= \{X'C\#Z\$X\#,\ \#Y''\$Z\#Y',\ \overline{X}B\#\$\#ZY,\ \#Y\$Z\#Y\},\\
A_3 &= \{X\alpha_i Z, X\alpha_i CZ \mid 1 \le i \le n\}\\
&\cup \ \{ZY, \overline{X}BZ, Z\overline{Y}, ZCY, XCZ, ZY''\},\\
R_3 &= \{X\alpha_i\#Z\$X'C^i\# \mid 1 \le i \le n\}\\
&\cup \ \{\#Y'\$Z\#Y,\ \overline{X}B\#Z\$XC\#,\ \#Y'\$Z\#\overline{Y},\\
&\quad \#CY\$Z\#CY,\ XC\#\$XC\#Z,\ \#Y''\$Z\#Y''\}\\
&\cup \ \{\overline{X}\alpha_i C\#\$\overline{X}\alpha_i C\#Z \mid 1 \le i \le n\}.
\end{aligned}
$$

Let us examine the work of the system Γ.

Each component contains certain splicing rules which are meant to be trap rules: if they can be applied once, they can be applied for ever, hence the corresponding component cannot stop correctly its work. Such rules are $\overline{X}\#\$\overline{X}\#Z$ in the first component, $\#Y\$Z\#Y$ in the second one, and $\#CY\$Z\#CY$, $XC\#\$XC\#Z$, $\#Y''\$Z\#Y''$, $X\alpha_i C\#\$X\alpha_i C\#Z$, $1 \le i \le m$, in the third component.

Thus, if a string of the form $XxY, x \in (N \cup T \cup \{B\})^*$ (initially, we have $x = BS$) is processed by the second component, then the system is blocked. The third component cannot modify such a string, hence it is passed away unchanged.

In the first component, a string XxY is processed as follows. At the end of x we can simulate rules of G ($(Xx_1|uY, Z|vY) \vdash Xx_1vY$, for $x = x_1u$ and $u \to v \in P$; the vertical bars indicate the place of splicing). Also, a symbol α_i can be cut from the end of x ($(Xx_1|\alpha_i Y, Z|C^iY') \vdash Xx_1C^iY'$, when $x = x_1\alpha_i$); the removed symbol is replaced by i occurrences of the symbol C and the replacement is also indicated by replacing Y with Y'. We have to continue with $(Xx_1C^{i-1}|CY', Z|Y'') \vdash Xx_1C^{i-1}Y''$. Note that one occurrence of C has been removed and Y' has been replaced with Y''.

No further splicing can be done in the first component involving the obtained string $Xx_1C^{i-1}Y''$, so it has to be passed to another component. The only possibility which does not block the system is to start working in the second component. Two splicings can be done here, at the ends of the string: $(X'C|Z, X|x_1C^{i-1}Y'') \vdash X'Cx_1C^{i-1}Y'', (X'Cx_1C^{i-1}|Y'', Z|Y') \vdash X'Cx_1C^{i-1}Y'$. In this way, one occurrence of C has been introduced in the left end of the string and the markers at the ends of the string have become X', Y'. No further splicing can be done here. We have two cases:

(1) If the string is passed to the first component, then, after using the rule $X'\#\$X\#Z$, we obtain the string $XCx_1C^{i-1}Y'$ and again we have to cut one occurrence of C from its right hand and to replace Y' with Y''. As above, the string must be passed to the second component and the process can be iterated. In this way, any number of occurrences of the symbol C can be removed from the right hand end of the

string and reintroduced in the left hand end.

(2) If the string is passed to the third component, then again two splicings will be done here, at the ends of the string. Assume that we start working in (A_3, R_3) on a string of the form $X'C^jx_1C^kY'$ for some $j, k \geq 0$. The symbol Y' is replaced by Y. If $k \geq 1$, then the rule $\#CY\$Z\#CY$ can be used for ever. Therefore, we must start from a string $X'C^jx_1Y'$. If we perform a splicing $(X\alpha_s|Z, X'C^s|C^{j-s}x_1Y) \vdash X\alpha_sC^{j-s}x_1Y$ with $j > s$, then again the system is blocked: the rule $X\alpha_sC\#\$X\alpha_sC\#Z$ can be used indefinitely. A correct continuation is possible only when we have $s = j$; this leads to the string $X\alpha_sx_1Y$. In this way, all occurrences of C were replaced by the corresponding symbol in $N \cup T \cup \{B\}$. This means that exactly the symbol α_i which has been removed from the right hand end of the string has been reintroduced in the left hand end (with the notation above, $i = j = s$). Thus, any circular permutation of the string can be obtained.

The string produced by the third component is of the form XzY, hence it can only be processed by the first component, where it is possible to simulate other rules in P in the end of z.

By iterating this rotate-and-simulate procedure, we can simulate in Γ any derivation in G.

Note that B is circulated as any other symbol. Initially, B in introduced in the left hand of S, the axiom of G. Because at every moment exactly one occurrence of B is present, it indicates the actual beginning of the sentential form of G simulated by Γ in a permuted form: if the string produced by Γ is Xz_1Bz_2Y, possibly with X, Y replaced with some primed versions of them, then z_2z_1 is a sentential form of G.

After receiving a string $X'C^jx_1Y', j \geq 1$, the third component can also use the rules of the form $\overline{X}B\#\$XC\#, \#Y'\$Z\#\overline{Y}$. If both these rules are used, then we get the string $\overline{X}BC^{j-1}x_1\overline{Y}$. If $j - 1 \geq 1$, then the system is blocked by the rule $\overline{X}BC\#\$\overline{X}BC\#Z$. Therefore, the string must be $\overline{X}Bx_1\overline{Y}$. The first component is blocked if receiving this string; the second one can work one step, removing the prefix $\overline{X}B$. The obtained string has to be taken by another component. The third one cannot modify it, the first one will remove the symbol $\overline{Y}$ from its end. If the obtained string is terminal, then it is an element of the generated language. Because the unique occurrence of the symbol B has been removed when placed in the leftmost position, it follows that the string is in the same circular permutation as the corresponding sentential form of G.

Finally, assume that the third component produces a string with only one bar symbol. If this string is of the form $\overline{X}BxY$, then no component can process it without blocking the system: the first component contains the rule $\overline{X}\#\$\overline{X}\#Z$, the second one contains the rule $\#Y\$Z\#Y$. If the string is of the form $XBx\overline{Y}$, then it can reach both the first and the second components.

In the first component, the string $XBx\overline{Y}$ will lose the symbol $\overline{Y}$, then it has to be moved to the second component. Here, a symbol C is added in the left end, producing $X'CBx$. This string can go back to the first component; iterating these steps, we can produce $X'C^iBx$ for some $i \geq 1$. Eventually, the string will be processed by the third component, and a string $X\alpha_iBx$ is produced. If $\alpha_i \neq B$, then B can never be removed:

it cannot be moved near X, because no rotation phase is possible (the symbol Y is no longer present). If $\alpha_i = B$ (hence $i = 1$), then XB can be removed, but we get Bx, which is not a terminal string.

If the string $XBx\overline{Y}$ arrives in the second component, then again we can introduce an occurrence of C in its left end, the obtained string $X'CBx\overline{Y}$ is then processed by the first component, which removes $\overline{Y}$, and we have returned at a situation as above: no terminal string can be produced, because the symbol B cannot be removed.

In conclusion, every derivation in G can be simulated in Γ and, conversely, if a terminal string is produced by the system Γ, then it is an element of $L(G)$. That is, $L(G) = L(\Gamma)$. □

Open problem: Which of the inclusions $CDEH_1 \subseteq CDEH_2 \subseteq CDEH_3$ are proper ? In particular, is the result in Theorem 4 optimal, or we have $RE = CDEH_2$? In the case of grammars, systems with two components working in the maximal mode are as powerful as systems with one component – hence as usual grammars. We probably do not have such a result in our case: $REG \subset CDEH_2$ (Lemma 1 (iii)) and $EH(FIN, FIN) = REG$ (Theorem 2). Note however that the mode of working in usual H systems is different from that in the components of cooperating distributed H systems: in the latter case we use at each step an axiom and the derivation stops correctly only when no further step is possible. On the other hand, we also believe that $CDEH_2 \subset RE$ (hence that the result in Theorem 4 is sharp): if we have only two components, then a computation proceeds by cyclically using the two components, in a rather deterministic way. This is expected to decrease the generative power.

It is interesting to note the similarity of the result above with that pointed out in Theorem 1: in the maximal mode of using the components, CD grammar systems with three context-free components are as powerful as systems with any larger number of components (however, such systems do not characterize the family RE, but the family $ET0L$, which is strictly included in CS).

6 Limiting the Size of Components

The motivation of CD H systems steaming from the difficulty of putting several restriction enzymes to work in the same reaction conditions makes it natural the problem of bounding the number of splicing rules in each component of a system.

For a CD H system $\Gamma = (V, T, w, (A_1, R_1), \ldots, (A_n, R_n))$ we define

$$size(\Gamma) = \max\{card(R_i) \mid 1 \leq i \leq n\}.$$

We also denote by $deg(\Gamma)$ the number n of components of Γ.

Theorem 5. *For each type-0 grammar $G = (N, T, S, P)$ we can construct a cooperating distributed H system Γ such that $L(G) = L(\Gamma)$ and*

$$deg(\Gamma) = 2 \cdot card(N \cup T) + card(P) + 10,$$
$$size(\Gamma) = 3.$$

Proof. For a type-0 grammar $G = (N, T, S, P)$, consider a new symbol, B, and denote, for an easy reference, $N \cup T \cup \{B\} = \{\alpha_1, \ldots, \alpha_n\}$. Because $N \neq \emptyset, T \neq \emptyset$, we have $n \geq 3$. Assume the rules in P labelled in a one-to-one manner with $r_1, \ldots, r_m$, for $m = card(P)$. We construct the CD H system Γ with the total alphabet

$$V = N \cup T \cup \{X, X', X'', Y, Y', Y'', Z, B, C, E\},$$

the terminal alphabet T, the axiom

$$w = XBSY,$$

and with the components described below. We identify these components with elements π in the set

$$M = \{1,2,3,4,5,6,7,8\} \cup \{\alpha_i, \alpha_i' \mid 1 \leq i \leq n\} \cup \{r_i \mid 1 \leq i \leq m\}.$$

$$\begin{aligned}
\pi = r_i: \quad & A_\pi = \{Zv_iY'', XCZ\}, \\
& R_\pi = \{\#u_iY\$Z\#v_iY'',\ XC\#Z\$XC\#\}, \\
& \text{where } 1 \leq i \leq m, \\
\pi = 1: \quad & A_\pi = \{ZY, X''Z\}, \\
& R_\pi = \{\#Y''\$Z\#Y,\ X''\#Z\$X''\#\}, \\
\pi = \alpha_i: \quad & A_\pi = \{ZC^iY, XCZ, X'Z\}, \\
& R_\pi = \{\#\alpha_iY\$Z\#C^iY,\ XC\#Z\$XC\#,\ X'\#Z\$X'\#\}, \\
& \text{where } 1 \leq i \leq n, \\
\pi = 2: \quad & A_\pi = \{ZY', X'Z\}, \\
& R_\pi = \{\#CY\$Z\#Y',\ X'\#Z\$X'\#\}, \\
\pi = 3: \quad & A_\pi = \{X'CZ, ZY'', ZY\}, \\
& R_\pi = \{X'C\#Z\$X\#,\ \#Y''\$Z\#Y'',\ \#Y\$Z\#Y\}, \\
\pi = 4: \quad & A_\pi = \{ZY, XZ\}, \\
& R_\pi = \{\#Y'\$Z\#Y,\ X\#Z\$X\#\}, \\
\pi = 5: \quad & A_\pi = \{XZ, ZY', ZE\}, \\
& R_\pi = \{X\#Z\$X'\#,\ \#Y'\$Z\#Y',\ \#E\$Z\#E\}, \\
\pi = \alpha_i': \quad & A_\pi = \{X\alpha_iZ, X\alpha_iCZ, ZCY\}, \\
& R_\pi = \{X\alpha_i\#Z\$XC^i\#,\ X\alpha_iC\#Z\$X\alpha_iC\#,\ \#CY\$Z\#CY\}, \\
& \text{where } 1 \leq i \leq n, \\
\pi = 6: \quad & A_\pi = \{ZE, X'Z, XCZ\}, \\
& R_\pi = \{\#BY\$Z\#E,\ X'\#Z\$X'\#,\ XC\#Z\$XC\#\}, \\
\pi = 7: \quad & A_\pi = \{X''Z, ZY, ZY'\}, \\
& R_\pi = \{X''\#Z\$X\#,\ \#Y\$Z\#Y,\ \#Y'\$Z\#Y'\}, \\
\pi = 8: \quad & A_\pi = \{ZZ, ZY''\}, \\
& R_\pi = \{\#ZZ\$X''\#,\ \#E\$ZZ\#,\ \#Y''\$Z\#Y''\}.
\end{aligned}$$

Let us examine the work of Γ. The idea of the construction is similar to that used in the proof of Theorem 4: the rotate-and-simulate procedure, combined with the trap rules feature made possible by the maximal mode of working in each component; by such trap rules we can control the correct simulation of derivations in G by computations in Γ.

Specifically, the components identified by $\pi = r_i, 1 \leq i \leq m$, simulate the corresponding rules in P. A rule $\#u_iY\$Z\#v_iY''$ can be used only once, starting from a string Xwu_iY and obtaining Xwv_iY''; because of the trap rule $XC\#Z\$XC\#$, the string w cannot begin with C. The component (A_1, R_1) returns Xwv_iY'' to Xwv_iY (when X'' is not present), hence any number of rules of P can be simulated.

The components identified by $\pi = \alpha_i, 1 \leq i \leq n$, start the rotation of the string bounded by X, Y, by replacing an occurrence of α_i by C^i. The work of these components is correctly ended only when no symbol C is present near X and X' is not present in the string. The components with $\pi = 2, 3, 4, 5$ cut a symbol C from the rightmost position (replacing Y by Y'), reintroduce it in the leftmost position (replacing X by X'), then return Y' to Y and X' to X, respectively. Always the trap rules prevent the use of these rules in a wrong order (for splicing wrong strings). The reader can check the details. The components identified by $\pi = \alpha'_i, 1 \leq i \leq n$, conclude the move of the symbol α_i from the rightmost position to the leftmost one:

$$(X\alpha_i|Z, XC^i|wY) \vdash X\alpha_i wY.$$

Note that the presence of X is necessary and that w cannot begin or end with occurrences of C: in the latter cases, the trap rules can be aplied and the work of this component is never correctly finished.

The components with $\pi = 6, 7, 8$ conclude the work of Γ, by removing the control symbols X and Y. Specifically, Y is replaced by E provided that B is adjacent to Y; this means that the current string generated by Γ corresponds to a sentential form of G and, moreover, the two strings are in the same circular permutation. The operation is possible only if X' is not present and no occurrence of C appears adjacent to X. The component (A_7, R_7) cannot be applied to strings ended with Y or Y'. If it is applied to a string ended with Y'', then Y'' is never removed: the component (A_8, R_8) cannot be applied, hence X'' cannot be removed, and in the presence of X'' the component (A_1, R_1) is not applicable.

Consequently, $L(\Gamma) = L(G)$.

Because Γ contains $2 \cdot card(N \cup T \cup \{B\}) + card(P) + 8$ components, each of them with two or three splicing rules, the proof is complete. □

The maximum length of the strings $u_i, 1 \leq i \leq 4$, in a splicing rule $u_1\#u_2\$u_3\#u_4$ is called the *radius* of the rule. An H system (distributed or not) is said to be of radius k if k is the largest radius of its splicing rules. Because the pattern recognized by the restriction enzymes is of a restricted length, it is also important to bound the radius of the CD H systems we use. It is an *open problem* whether or not this can be achieved without increasing the size of the components of the obtained systems. (The techniques

used in [13] do not directly work in the new framework: instead of a codification rule $\#\alpha_i Y\$Z\#C^iY$ one uses a rule $\#\alpha_i Y\$Z\#Y_i$, which memorizes the removed symbol in the subscript of Y. Now, we have to introduce a symbol α_j in the leftmost position of a string XwY_i in such a way to ensure the fact that $j = i$; this can be done by considering trap rules for all Y_j with $j \neq i$, but this means involving $n-1$ trap rules, hence the size of the component increases in an uncontrolled way.)

7 Final Remarks

In general, in the descriptional complexity area, one cannot simultaneously improve in two parameters, a trade-off principle is almost always valid (see, e.g, [7]). This is not the case for the two measures considered in the previous sections, the number of components and the size of components of a CD H system. We can bound both these measures and still we can characterize the recursively enumerable languages. This is due, on the one hand, to the fact that there exist universal type-0 grammars (see an explicit construction in [1]) and, on the other hand, to the very constructions in the proofs of Theorem 4 and 5. A grammar $G_u = (N_u, T, -, P_u)$ (without an axiom) is called universal if for every type-0 grammar $G = (N, T, S, P)$ there is a string $code(G)$ such that the grammar $G_u(G) = (N_u, T, code(G), P_u)$ generates the language $L(G)$ (the work of $G_u(G)$ starts from the string axiom $code(G)$). Now, start the constructions of CD H systems in the proofs of Theorems 4 and 5 from a given G_u; we obtain a system Γ_u with all components fixed (depending on G_u), without an axiom $XBSY$. For a specific grammar G, consider the variant of Γ_u with the axiom $w = XBcode(G)Y$. From the universality of G_u it is clear that in this way we generate the language $L(G)$. Consequently, the size and the degree of Γ_u are bounds for all CD H systems associated as above to grammars G, hence to systems which are sufficient in order to characterize RE.

Results similar to Theorems 4, 5 are obtained also for the so-called *communicating distributed H systems* introduced in [4]: systems with seven components (and with the size not bounded in advance) ([14]) and systems with only one splicing rule per component (without a bound on the number of components; [15]) can characterize the recursively enumerable languages. Note the differences with the results in the present paper: here three components are enough, but when bounding the size we have used three rules in each component. We do not know whether or not this last bound can be improved.

Note. Long and useful discussions with Prof. Corrado Böhm during a visit of the second author to the University of Rome "La Sapienza" are gratefully acknowledged. Actually, the idea of a CD H system has appeared during these discussions, as a possible answer to the question of introducing also in the H system area of a counterpart of the way of computing in combinatory logic, by sequencing operators (combinators) from a finite set.

References

[1] C. Calude, Gh. Păun, Global syntax and semantics for recursively enumerable languages, *Fundamenta Informaticae*, 4, 2 (1981), 245 – 254.

[2] E. Csuhaj-Varju, J. Dassow, On cooperating distributed grammar systems, *J. Inform. Process. Cybern., EIK*, 26 (1990), 49 – 63.

[3] E. Csuhaj-Varju, J. Dassow, J. Kelemen, Gh. Păun, *Grammar Systems. A Grammatical Approach to Distribution and Cooperation*, Gordon and Breach, London, 1994.

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

[5] K. Culik II, T. Harju, Splicing semigroups of dominoes and DNA, *Discrete Appl. Math.*, 31 (1991), 261 – 277.

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

[7] J. Gruska, Descriptional complexity of context-free languages, *Proc. MFCS Symp.*, High Tatras, 1973, 71 – 83.

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

[9] T. Head, Gh. Păun, D. Pixton, Language theory and molecular genetics. Generative mechanisms suggested by DNA recombination, chapter 7 in vol. 2 of [19], 295 – 360.

[10] P. H. Nii, Blackboard systems, in *The Handbook of AI*, vol. 4 (A. Barr, P. R. Cohen, E. A. Feigenbaum, eds.), Addison-Wesley, Reading, Mass., 1989.

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

[12] Gh. Păun, Five (plus two) universal DNA computing models based on the splicing operation, *Second DNA Based Computers Workshop*, Princeton, 1996, 67 – 86.

[13] Gh. Păun, Computing by splicing: How simple rules ?, *Bulletin of the EATCS*, 60 (1996), 145 – 150.

[14] Gh. Păun, DNA computing; Distributed architectures, in *Structures in Logic and Computer Science. A Selection of Essays in Honor of A. Ehrenfeucht* (J. Mycielski, G. Rozenberg, A. Salomaa, eds.), *Lect. Notes in Computer Sci.* 1261, Springer-Verlag, 1997.

[15] Gh. Păun, Distributed architectures in DNA computing based on splicing: Limiting the size of components, *First Intern. Conf. on Unconventional Models of Computation*, Auckland, 1998.

[16] Gh. Păun, G. Rozenberg, A. Salomaa, Computing by splicing, *Theoretical Computer Sci.*, 168, 2 (1996), 321 – 336.

[17] Gh. Păun, A. Salomaa, DNA computing based on the splicing operation, *Mathematica Japonica*, 43, 3 (1996), 607 – 632.

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

[19] G. Rozenberg, A. Salomaa, eds., *Handbook of Formal Languages*, 3 volumes, Springer-Verlag, Heidelberg, 1997.