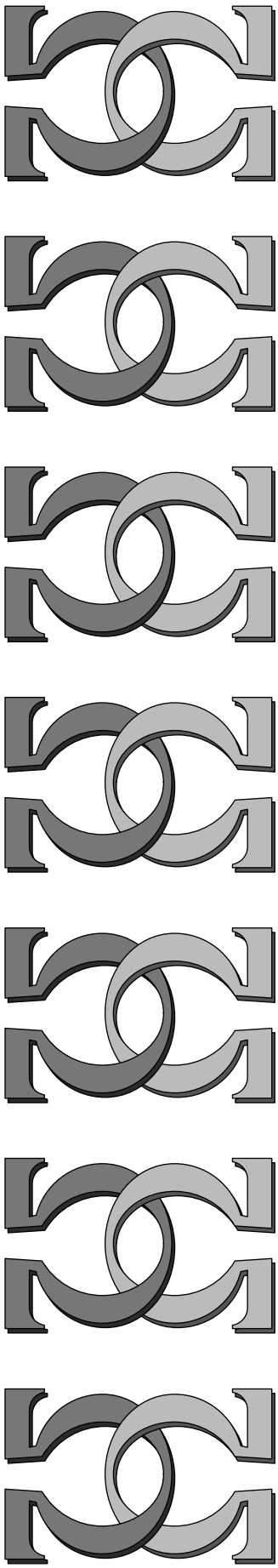

CDMTCS
Research
Report
Series

Two-Level Distributed H Systems

Gheorghe Păun
Institute of Mathematics of the Romanian Academy

CDMTCS-073
December 1997

Centre for Discrete Mathematics and
Theoretical Computer Science

Two-Level Distributed H Systems

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

Abstract. We deal here with the computational capacity of DNA when *splicing* it, by means of restriction enzymes and ligases. We introduce a new distributed structure of a computing system (we call it *two-level H system*), with each component working by splicing (according to *internal* splicing rules) and communicating, also by splicing, according to *external* splicing rules. This architecture is proven to be computationally universal, systems with three components characterize the recursively enumerable languages. The possibility of designing universal DNA computers based on splicing is inferred on this basis.

1. Introduction

This paper can be seen as a contribution to the fastly growing area of (mathematical foundations of) DNA computing. Specifically, we look for (1) universal computing devices which (2) are as simple as possible. Universality is understood at two levels: having the power of Turing machines (and of any other equivalent class of algorithms) and, moreover, having the possibility to design "universal programmable computers", devices with all components but one fixed and able to behave as any given particular device after introducing its *code* in the non-fixed component. On the other hand, as basic simplicity criteria we consider here at the same time the *size* of the machinery and its *homogeneity*. Precisely, our aim is to have a model with an as small as possible variety of elementary (types of) operations. This has both a mathematical and a practical motivation: a machinery making use of both biochemical-like operations (DNA recombination, cross-overing, etc.) and language-theoretic operations (substitution, insertion, checking context conditions, etc.) does not look, estetically and practically, very attractive.

From both these point of view, the *two-level* H systems we introduce here are quite successful: (1) we use only one operation, the splicing, in the sense of [12], and (2) systems with only three components are (3) computationally universal, they reach the full power of Turing machines. Moreover, the proof of these assertions directly provides a universal two-evel H system: start the construction in the proof from a universal Turing machine (in fact, a universal type-0 Chomsky grammar) and what we get is a universal two-level H system.

As we have said, the fundamental operation used here is that of *splicing*, introduced in [12] as a model of the recombinant behavior of DNA under the influence

of restriction enzymes and ligases. Namely, we consider quadruples (u_1, u_2, u_3, u_4), of strings over some given alphabet, where $(u_1, u_2), (u_3, u_4)$ encode the sites where enzymes can cut the DNA sequences; having these sites together in a quadruple means that the substrings obtained after cuting the DNA sequence can be pasted together, producing new strings. For instance, if $x = x_1u_1u_2x_2$ and $y = y_1u_3u_4y_2$, first we obtain $x_1u_1, u_2x_2, y_1u_3, u_4y_2$, then we build any of the strings $x_1u_1u_2x_2, x_1u_1u_4y_2, y_1u_3u_4y_2, y_1u_3u_2x_2$. The second and the last strings are (possibly) new. We say that we have *spliced* x, y at the *sites* u_1u_2, u_3u_4, according to the *splicing rule* (u_1, u_2, u_3, u_4).

The splicing operation has been investigated in a series of papers. We refer to [13], [15] and especially to [20] for details and bibliographical information. We only mention here the notion of *extended H systems*, [19], generative mechanisms based on splicing and consisting of a set of string *axioms* over a given *alphabet* and a set of *splicing rules*. Starting from the axioms and iteratively applying the rules we obtain a language; squeezing out of it only the strings over a given *terminal* alphabet we get the *language generated* by the system.

In view of the results in [7] and [22], such systems with finite sets of axioms and finite sets of splicing rules can produce only regular languages (all of them, as it was proved in [19]). If a regular set of rules is allowed (codifying a splicing rule as a string, in a natural way, we can speak about the type of the splicing rules set in the Chomsky hierarchy), then the extended H systems become computationally universal, [18].

However, this result is not of much practical interest, because one cannot manipulate regular sets of strings in a test tube. On the other hand, the real DNA language is not regular, even context-free, see [4], [25], in spite of the fact that the real DNA seems to be built by using finitely many axioms and finitely many splicing rules.

Several ideas how to overpass this contradiction, hence the regularity borderline, were explored in [17], [10], [6], [26], [20]. In all cases, extended H systems with finite sets of axioms and of splicing rules become computationally universal, providing that certain additional features are added to the system. In general, a regulation mechanism on the splicing rule application is considered. In [17] one work with multisets (sets with multiplicities associated to their elements and updated during the splicing operation), in [10] one considers permitting or forbidding symbols associated to the splicing rules (promoters and inhibitors), [26] uses circular strings. The fact that we jump directly from regular languages to recursively enumerable languages is worth emphasizing. An idea related to that of the present paper is proposed in [6]: to organize the process in a distributed way as suggested by the grammar system area, [5]. Again universal devices are obtained, but the systems in [6] does not fulfil the simplicity criterion mentioned above: the universal systems have a large number of components and, moreover, they are not *homogeneous*. Besides splicing, in [6] one essentially uses the operation of checking whether or not a string belongs to a regular language of the form V^*, where V is an alphabet. This is done for a unbounded number of times (not only for selecting the generated language), in order to control the cooperation among tubes, in a way similar to the *merge, separate, amplify* operations

of [2], [14].

The basic idea of the two-level H systems proposed here is to have two types of strings in each tube, one *active* and several *passive*, as well as two types of splicing rules, one set used *internally*, inside the tube, and the other one used *externally*, among active strings in different tubes. The external splicing have priority over the internal one. This is enough in order to reach again the full power of Turing machines *by two-level H systems with three components only*. It is an open problem whether or not two tubes are enough. Moreover, the proof gives directly a way of obtaining universal "programable" two-level H systems.

2. Basic definitions; the splicing operation

We use the following formal language notations: V^* is the free monoid generated by the alphabet V, λ is the empty word, $V^* - \{\lambda\}$ is denoted by V^+, *FIN, REG, RE* are the families of finite, regular, and recursively enumerable languages, respectively. For general formal language theory prerequisites we refer to [23]; for grammar systems area we refer to [5].

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 $x, y, w, z \in V^*$ we write

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

We say that we have *spliced* x, y at the *sites* u_1u_2, u_3u_4, respectively, obtaining the strings w, z; x, y are called the *terms* of the splicing. When understood from the context, we omit r 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 $L \subseteq V^*$, we define

$$\sigma(L) = \{w \in V^* \mid (x,y) \vdash_r (w,z) \text{ or } (x,y) \vdash_r (z,w), \text{ for some } x, y \in L, r \in R\}.$$

(By definition, $\sigma(L) = \emptyset$ if $L = \emptyset$ or $R = \emptyset$.) Then, we define

$$\sigma^*(L) = \bigcup_{i \geq 0} \sigma^i(L),$$

for

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

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

An *extended H 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 pair $\sigma = (V, R)$ is called the *underlying H scheme* of γ. The language generated by γ is defined by

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

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 $A \in F_1, R \in F_2$. We denote by $EH(F_1, F_2)$ the family of languages generated by extended H systems of type (F_1, F_2).

An H system $\gamma = (V, V, A, R)$ is said to be *non-extended*; 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 splicing operation is introduced in [12] for finite sets of rules; the case of arbitrarily large sets of splicing rules is considered in [16]; the extended H systems were introduced in [19].

In [7], [22] it is proved that

$$H(FIN, FIN) \subseteq REG.$$

(The inclusion is, in fact, proper.) Using this relation, in [19] it is proved that

$$EH(FIN, FIN) = REG.$$

Moreover, in [18] it is proved that the extended H systems with finite sets of axioms and regular sets of splicing rules are computationally complete, that is

$$EH(FIN, REG) = RE.$$

Therefore, such systems are as powerful as the Turing machines (and any other class of equivalent algorithms). However, from practical points of view, it is not realistic to deal with infinite – even regular – sets of rules. We have mentioned in the introduction several ways to handle this difficulty, in general by using finitely many rules controlled by an additional mechanism. One possibility is to organize the process in a distributed way, as usual in grammar systems theory, [5].

We propose here a new structure of a distributed H system, using no other ingredients than the splicing operation (according to sets of splicing rules among which a priority relation is assumed).

3. Distributed two-level H systems

A *two-level H system* (of degree $n, n \geq 1$), is a construct

$$\Gamma = (V, T, \tau_1, \ldots, \tau_n),$$

where V is an alphabet, $T \subseteq V$, and

$$\tau_i = (w_i, A_i, I_i, E_i),\ 1 \leq i \leq n,$$

where $w_i \in V^+, A_i \subseteq V^*, I_i, E_i \subseteq V^*\#V^*\$V^*\#V^*$, for $\#, \$$ symbols not in V.

The meaning of these elements is as follows: V is the *total alphabet*, T is the *terminal alphabet*, and $\tau_i, 1 \leq i \leq n$, are the *components* of the system (we also call them *test tubes*); for each component, w_i is the *active axiom*, A_i is the set of *passive axioms*, I_i is the set of *internal splicing rules*, and E_i is the set of *external splicing rules*.

One can imagine a two-level H system as consisting of n (active) DNA strings, say $z_1, \ldots, z_n$ (initially they are $w_1, \ldots, w_n$), with their left end fixed on a solid support, surrounded each by a "soup" containing arbitrarily many copies of passive strings (initially those in $A_1, \ldots, A_n$, respectively), and having around both strong restriction enzymes which can "see" only the active strings and weak restriction enzimes, acting only locally, on z_i and an associated string in the passive set. As a result of the local splicing, a string with the prefix of z_i will be obtained, hence again fixed on the support, and one more string which will be added to the surrounding set of passive strings. An external splicing has priority over the internal splicing. When an external splicing is performed, according to a rule in some set E_i, then the associated string z_i is the first term of the splicing, hence a new string fixed on the solid support is obtained, having a common prefix with z_i; the second term of the splicing, some $z_j, j \neq i$, remains unchanged after this operation – one can assume that a copy of z_j has been produced and sent to tube i just for participating to the splicing.

Formally, these operations are defined as follows.

The *contents* of a tube $\tau_i, 1 \leq i \leq n$, 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 denote

$$\mu(x_i, \pi) = \begin{cases} external, & \text{if there are } r \in E_i \text{ and } x_j, j \neq i, \\ & \text{such that } (x_i, x_j) \vdash_r (u, v), u, v \in V^*, \\ internal, & \text{otherwise.} \end{cases}$$

Then, for two configurations $\pi = [(x_1, M_1), \ldots, (x_n, M_n)]$ and $\pi' = [(x'_1, M'_1), \ldots, (x'_n, M'_n)]$, we write $\pi \Longrightarrow_{ext} \pi'$ if the following conditions hold:

1. there is $i, 1 \leq i \leq n$, such that $\mu(x_i, \pi) = external$,
2. for each $i, 1 \leq i \leq n$, with $\mu(x_i, \pi) = external$, we have $(x_i, x_j) \vdash_r (x'_i, z)$, for some $j, 1 \leq j \leq n, j \neq i$, $r \in E_i$, and $M'_i = M_i \cup \{z\}$,
3. for each $i, 1 \leq i \leq n$, with $\mu(x_i, \pi) = internal$, we have $(x'_i, M'_i) = (x_i, M_i)$.

For two configurations π and π' as above, we write $\pi \Longrightarrow_{int} \pi'$ if the following conditions hold:

1. for all $i, 1 \leq i \leq n$, we have $\mu(x_i, \pi) = \mathit{internal}$,
2. for each $i, 1 \leq i \leq n$, either $(x_i, z) \vdash_r (x_i', z')$, for some $z \in M_i, z' \in V^*, r \in I_i$, and $M_i' = M_i \cup \{z'\}$, or
3. 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 $\Longrightarrow_{ext}$ defines an external splicing, $\Longrightarrow_{int}$ defines an internal splicing. Note that in both cases all the splicing operations are performed in parallel and the components not able to use a splicing rule do not change their contents. We stress the fact that the external splicing has priority over the internal one and that all operations have as the first term an active string; the first string obtained by splicing becomes the new active string of the corresponding component, the second string obtained becomes an element of the set of passive strings of that component.

We write $\Longrightarrow$ for both $\Longrightarrow_{ext}$ and $\Longrightarrow_{int}$ and $\Longrightarrow^*$ for the reflexive and transitive closure of $\Longrightarrow$. The *language generated* by a two-level H system Γ is defined by

$$L(\Gamma) = \{w \in T^* \mid [(w_1, A_1), \ldots, (w_n, A_n)] \Longrightarrow^* [(x_1, M_1), \ldots, (x_n, M_n)], \text{ for } w = x_1, x_i \in V^*, 2 \leq i \leq n, \text{ and } M_i \subseteq V^*, 1 \leq i \leq n\}.$$

We denote by TLH_n the family of languages generated by two-level H systems with at most n components, $n \geq 1$, all of them having *finite sets of axioms and finite sets of splicing rules*. When no restriction is imposed on the number of components, we write TLH_∞.

Here is an *example*: Consider the system

$$\Gamma = (\{a, b, C, D\}, \{a, b\}, \tau_1, \tau_2),$$

with

$$\begin{array}{ll} w_1 = aD, & w_2 = DbC, \\ A_1 = \{aD, Da\}, & A_2 = \{bC\}, \\ I_1 = \{b\#C\$\#aD,\ b\#C\$D\#a\}, & I_2 = \{b\#C\$\#bC\}, \\ E_1 = \{a\#D\$D\#b\}, & E_2 = \emptyset. \end{array}$$

A computation in Γ runs as follows:

$$\begin{array}{ll} & [(aD, \{aD, Da\}), (DbC, \{bC\})] \\ \Longrightarrow_{ext} & [(abC, \{aD, Da, DD\}), (DbC, \{bC\})] \\ \Longrightarrow_{int} & [(abaD, \{aD, Da, DD, C\}), (Db^2C, \{bC, C\})] \\ \Longrightarrow_{ext} & [(abab^2C, \{aD, Da, DD, C\}), (Db^2C, \{bC, C\})] \\ \Longrightarrow_{int} & [(abab^2D, \{aD, Da, DD, C\}), (Db^3C, \{bC, C\})] \Longrightarrow^* \ldots \\ \Longrightarrow_{ext} & [(abab^2 \ldots ab^kC, \{aD, Da, DD, C\}), (Db^kC, \{bC, C\})] \\ \Longrightarrow_{int} & [(abab^2 \ldots ab^ka, \{aD, Da, DD, C, DC\}), (Db^{k+1}C, \{bC, C\})], \end{array}$$

for some $k \geq 1$.

After an alternate sequence of external and internal splicings, the active string of tube 1 becomes $abab^2 \ldots ab^kC$, which can be turned out to a terminal string by replacing C with a. Therefore,

$$L(\Gamma) = \{abab^2 \ldots ab^ka \mid k \geq 1\}.$$

This language is not semilinear, hence it is not context-free (it is neither a matrix one, in view of the fact that each one-letter matrix language is regular, [11], and by removing the symbol a from the strings of $L(\Gamma)$ – this amounts to using a restricted morphism, and the family of matrix languages is closed under such an operation, [8] – we get the one-letter non-regular language $\{b^{\frac{k(k+1)}{2}} \mid k \geq 1\}$).

Consequently, TLH_2 contains non-context-free languages, the cooperation of tubes in a two-level H system increases the power of the splicing operation. In fact, as we shall see in the next section, this increasing is maximal, $TLH_3 = RE$.

We close this section by pointing out the similarity of the architecture of a two-level H system with that of a *parallel communicating* (PC) *grammar system*, introduced in [21] as a grammatical model of parallel computing. The internal splicing corresponds to the rewriting steps and the external splicing corresponds to the communication steps in a PC grammar system; in both cases one starts from some axioms, one work synchronously on components, and one considers as the generated language the set of strings generated by a *master* component, the first tube here, the first grammar in [21]. Of course, the analogy stops at this global level, the two devices are essentially different in details.

4. Characterizing RE languages

We take here as a test bed for our devices the Chomsky hierarchy of languages, hence RE as the maximal level of algoritmicity, making use of the equivalence of Turing machines and the Type-0 Chomsky grammars.

Directly from the definitions and also using the Turing-Church thesis, we have

$$TLH_1 \subseteq TLH_2 \subseteq \ldots \subseteq TLH_\infty \subseteq RE.$$

Obviously, a two-level H system of degree 1 is a particular case of an extended H system: the external splicing rules are of no use, but each internal splicing must have as the first term the currently active string. Therefore,

$$TLH_1 \subseteq EH(FIN, FIN) = REG.$$

Moreover, from the proof of the inclusion $REG \subseteq EH(FIN, FIN)$ in [19], [10], we can see that $REG \subseteq TLH_1$ (when simulating a finite automaton by an extended H system, always an "active" string is used as the first term of the splicing, hence the obtained H system is, in fact, a two-level H system of degree 1). Consequently, we have

$$TLH_1 = REG.$$

From the example in the previous section, we get the fact that the inclusion $TLH_1 \subset TLH_2$ is proper. We do not know whether or not also $TLH_2 \subseteq TLH_3$ is a proper inclusion, but, anyway, the hierarchy collapses at this level, TLH_3 is the largest family in this framework.

Theorem 1. *$TLH_3 = TLH_n = TLH_\infty = RE$, for all $n \geq 3$.*

Proof. Of course, we have to prove only the inclusion $RE \subseteq TLH_3$.

Consider a language $L \in RE$, generated by a type-0 Chomsky grammar $G = (N, T, S, P)$. We construct the two-level H system

$$\Gamma = (V, T, \tau_1, \tau_2, \tau_3),$$

with

$$\begin{aligned}
V &= N \cup T \cup \{B, B', C, D, E, F, H, J, U, X, X', Y, Z\},\\
w_1 &= SXXE,\\
A_1 &= \{HvYY \mid u \to v \in P\}\\
&\cup \{HZZ\alpha,\ H\alpha UU \mid \alpha \in N \cup T\}\\
&\cup \{JXX, F\},\\
I_1 &= \{\#uXX\$H\#vYY \mid u \to v \in P\}\\
&\cup \{\#\alpha XX\$H\#ZZ\alpha \mid \alpha \in N \cup T\}\\
&\cup \{\#XX\alpha\$H\#\alpha UU \mid \alpha \in N \cup T\}\\
&\cup \{\#YY\$J\#XX,\ \#ZZ\$J\#XX,\ \#UU\$J\#XX,\ \#XXE\$F\#\},\\
E_1 &= \{Z\alpha\#\$XX\# \mid \alpha \in N \cup T\}\\
&\cup \{UU\#\$XX\alpha\# \mid \alpha \in N \cup T\}\\
&\cup \{YY\#\$XX\#,\ XX\#\$BZ\#,\ XX\#\$BU\#,\ XX\#\$BY\#\},\\
w_2 &= CXX',\\
A_2 &= \emptyset,\\
I_2 &= \emptyset,\\
E_2 &= \{X\#X'\$X\#X,\ C\#X\$\#B,\ C\#B\$D\#X,\\
&\quad B\#B'\$Z\#Z,\ B\#B'\$Y\#Y,\ B\#B'\$U\#U\},\\
w_3 &= DXX',\\
A_3 &= \{BB', DXX'\},\\
I_3 &= \{\#DXX'\$\#BB',\ \#BB'\$\#DXX'\},\\
E_3 &= \emptyset.
\end{aligned}$$

The intuition behind this construction is the following. To a sentential form w of G corresponds in Γ a string of the form w_1XXw_2E, where $w_1w_2 = w$; XX indicates the "working place" (the place of the read-write head in the style of Turing machines), in the sense that a rule of G can be simulated only in the presence of XX: if $w_1 = w_1'u$,

for some $u \to v \in P$, and $w_1'uw_2 \Longrightarrow w_1'vw_2$ in G, then a derivation of the form $w_1'uXXw_2E \Longrightarrow w_1'vXXw_2E$ will be realized in Γ. The symbol E is a marker for the right-hand end of the string. Only in its presence we can remove XX (by the last rule in I_1). After removing XX, no further splicing is possibile in τ_1, hence the string is lost if it contains nonterminal symbols. In order to simulate rules in P in any position of the current string, we have to freely move XX to the left and to the right. All these operations – simulation of a rule in P, moving XX one step to the right, moving XX one step to the left – are performed in a similar way: First, we copy in the tube τ_2 the suffix of the active string of τ_1 starting with the second occurrence of X. Then, in the first component we start performing the desired operation. This means changing XX for YY (simulation of a rule), ZZ (moving one step to the left), or UU (moving one step to the right). This is done by an internal splicing. Then again an external splicing is performed, bringing together the active strings of the first and the second components. The initial string is reconstructed, modulo two changes: XX is replaced by YY, ZZ, UU and the desired operation is performed. At the same time, the second tube performs an external splicing which "cleans up" its active string. This is done with the help of the third tube, which has only this role, of providing an appropriate string for splicings in the second tube. Again the second tube makes a copy of the second "half" of the active string of the first tube, starting now with the second occurrence of the double symbol Y, Z, U, respectively. By an internal splicing, the first component reintroduces the substring XX, then it brings the end of the string from the second tube, reconstructing the string. The whole procedure can be iterated, because we are again in the same situation as at the beginning. At every step, the passive string sets of τ_1 and τ_2 are increased, that of τ_3 remains constantly equal to $\{BB', DXX'\}$. However, τ_2 never performs an internal splicing, whereas the newly introduced passive strings of τ_1 never participate to a splicing, due to the blocking symbols H and J occurring in the internal splicing rules of τ_1.

From these explanations, it should be clear that $L(\Gamma) = L(G)$. However, we still consider in some details the work of Γ.

We start from the configuration

$$[(SXXE, A_1), (CXX', \emptyset), (DXX', A_3)].$$

Consider, in general, a configuration with a string w_1XXw_2E in the first component. As we have seen, we have to distinguish three cases: (1) to the left of XX we have u, for some rule $u \to v \in P$ to be simulated, (2) to the left of XX we have a symbol $\alpha \in N \cup T$ to be interchanged with XX, and (3) to the right of XX we have a symbol $\alpha \in N \cup T$ to be interchanged with XX. We consider all these possibilities together, separated by vertical bars; the reader should understand that exactly one of them is running in each step and that the corresponding strings/sets of the components are running together.

Thus, let us start with

$$[(w_1uXXw_2E \mid w_1\alpha XXw_2E \mid w_1XX\alpha w_2E, M_1), (CXX', M_2), (DXX', A_3)].$$

In all cases we have to perform external splicings in τ_2. Hence we get

$$\begin{aligned}\Longrightarrow_{ext}\ & [(w_1uXXw_2E \mid w_1\alpha XXw_2E \mid w_1XX\alpha w_2E, M_1),\\ & (CXXw_2E \mid CXXw_2E \mid CXX\alpha w_2E,\ M_2^{(1)} = M_2 \cup \{w_1uXX'\} \mid\\ & M_2^{(2)} = M_2 \cup \{w_1\alpha XX'\} \mid M_2^{(3)} = M_2 \cup \{w_1XX'\}),\\ & (DXX', A_3)].\end{aligned}$$

No further external splicing has to be performed, hence we can perform internal splicing in tubes 1 and 3:

$$\begin{aligned}\Longrightarrow_{int}\ & [(w_1vYY \mid w_1ZZ\alpha \mid w_1\alpha UU,\\ & M_1^{(1)} = M_1 \cup \{HuXXw_2E\} \mid M_1^{(2)} = M_1 \cup \{H\alpha XXw_2E\} \mid\\ & M_1^{(3)} = M_1 \cup \{HXX\alpha w_2\}),\\ & (CXXw_2E \mid CXXw_2E \mid CXX\alpha w_2E,\ M_2^{(1)} \mid M_2^{(2)} \mid M_2^{(3)}),\\ & (BB', A_3)].\end{aligned}$$

Now we have to perform external splicing both in τ_1 and τ_2:

$$\begin{aligned}\Longrightarrow_{ext}\ & [(w_1vYYw_2E \mid w_1ZZ\alpha w_2E \mid w_1\alpha UUw_2E,\\ & M_1'^{(1)} = M_1^{(1)} \cup \{CXX\} \mid M_1'^{(2)} = M_1^{(2)} \cup \{CXX\} \mid\\ & M_1'^{(3)} = M_1^{(3)} \cup \{CXX\alpha\}),\\ & (CBB' \mid CBB' \mid CBB',\ M_2'^{(1)} = M_2^{(1)} \cup \{XXw_2E\} \mid\\ & M_2'^{(2)} = M_2^{(2)} \cup \{XXw_2E\} \mid M_2'^{(3)} = M_2^{(3)} \cup \{XX\alpha w_2E\}),\\ & (BB', A_3)].\end{aligned}$$

One more external splicing must be performed, in τ_2:

$$\begin{aligned}\Longrightarrow_{ext}\ & [(w_1vYYw_2E \mid w_1ZZ\alpha w_2E \mid w_1\alpha UUw_2E,\ M_1'^{(1)} \mid M_1'^{(2)} \mid M_1'^{(3)}),\\ & (CBYw_2E \mid CBZ\alpha w_2E \mid CBUw_2E,\ M_2''^{(1)} = M_2'^{(1)} \cup \{w_1vYB'\} \mid\\ & M_2''^{(2)} = M_2'^{(2)} \cup \{w_1ZB'\} \mid M_2''^{(3)} = M_2'^{(3)} \cup \{w_1\alpha UB'\}),\\ & (BB', A_3)].\end{aligned}$$

An internal splicing in τ_1 and τ_3 is possible:

$$\begin{aligned}\Longrightarrow_{int}\ & [(w_1vXX \mid w_1XX \mid w_1\alpha XX,\ M_1''^{(1)} = M_1'^{(1)} \cup \{JYYw_2E\} \mid\\ & M_1''^{(2)} = M_1'^{(2)} \cup \{JZZw_2E\} \mid M_1''^{(3)} = M_1'^{(3)} \cup \{CBUw_2E\}),\\ & (CBYw_2E \mid CBZ\alpha w_2E \mid CBUw_2E,\ M''^{(1)}_2 \mid M''^{(2)}_2 \mid M''^{(3)}_2),\\ & (DXX', A_3)].\end{aligned}$$

We have to perform external splicings both in τ_1 and τ_2:

$$\begin{aligned}\Longrightarrow_{ext} \quad & [(w_1vXXw_2E \mid w_1XX\alpha w_2E \mid w_1\alpha XXw_2E, \\ & M_1''^{(1)} \cup \{CBY\} \mid M_1''^{(2)} \cup \{CBZ\} \mid M_1''^{(3)} \cup \{CBU\}), \\ & (CXX' \mid CXX' \mid CXX',\ M_2''^{(1)} \cup \{DBYw_2E\} \mid \\ & M_2''^{(2)} \cup \{DBZ\alpha w_2E\} \mid M_2''^{(3)}) \cup \{DBUw_2E\}), \\ & (DXX', A_3)].\end{aligned}$$

We have started from $w_1uXXw_2E, w_1\alpha XXw_2E$, or $w_1XX\alpha w_2E$ in the first component and we have obtained $w_1vXXw_2E, w_1XX\alpha w_2E$, or, respectively, $w_1\alpha XXw_2E$. Hence we have indeed simulated the rule $u \to v \in P$, or we have moved XX one step to the left, or we have moved XX one step to the right. On components 2 and 3 we have returned to the strings CXX', DXX', respectively. The sets of passive strings are modified but this has no influence on the work of Γ, the new strings will never enter a splicing. The process can be iterated.

When we have a configuration of the form

$$[(wXXE, M_1), (CXX', M_2), (DXX', A_3)],$$

we have first to perform an external splicing in τ_2, which does not change the first active string, then an internal splicing in τ_1 can remove XXE:

$$\begin{aligned}\Longrightarrow_{ext} \quad & [(wXXE, M_1), (CXXS, M_2 \cup \{wXX'\}), (DXX', A_3)] \\ \Longrightarrow_{int} \quad & [(w, M_1 \cup \{FXXE\}), (CXXE, M_2 \cup \{wXX'\}), (BB', A_3)].\end{aligned}$$

If $w \in T^*$, then, clearly, $w \in L(G)$; if $w \notin T^*$, then no further splicing can involve w, hence the work of Γ is blocked. Consequently, $L(G) \subseteq L(\Gamma)$. Conversely, if Γ produces a string $w \in T^*$, this means that Γ has simulated a derivation in G, hence $L(\Gamma) \subseteq L(G)$. This concludes the proof of the equality $L(\Gamma) = L(G)$, hence of the inclusion $RE \subseteq TLH_3$. □

Remarks. For a splicing rule $u_1\#u_2\$u_3\#u_4$, the maximum of $|u_i|, 1 \leq i \leq 4$, is called the *radius* of the rule. If we start the previous construction from a type-0 grammar G with the rules of the form $u \to v$ with $|u|, |v| \leq 2$ (this is always possible: take, for instance, G in Kuroda normal form), then the radius of the system Γ, that is maximal radius of its rules, is 4, reached by rules of the form $\#uXX\$H\#vYY$ in I_1. In this way, also the length of each site u_1u_2, u_3u_4 in rules of Γ is bounded, namely by 5. This could be important from a practical point of view, because one knows that enzymes define cutting sites of small length, less that eight in most cases.

Two practical further remarks: the final intersection with T^* is a feasible filtering operation (used also in [1]), whereas bounding DNA strings with their left end to a support is feasible, too, see [9].

Consider now the case when we start the previous construction from a universal type-0 Chomsky grammar. Such a grammar is a construct $G_u = (N_u, T, -, P_u)$,

that is a grammar without an axiom and having the following property. If we have an arbitrarily given type-0 grammar G, we encode it in a suitable way, obtaining a string $w(G)$, and consider the grammar (with a string as axiom) $G_u(w(G)) = (N_u, T, w(G), P_u)$. Then we have $L(G) = L(G_u(w(G)))$. A universal type-0 grammar can be obtained from a universal Turing machine – whose existence is proved already in [24] – or can be directly constructed, as in [3]. Because G_u has no axiom, the active string w_1 of τ_1 in the previous construction is missing. All the other components are built in the same way. Note that all of them depend on the components of G_u, hence they are fixed. This means that the obtained two-level H system Γ_u is universal, in the following sense: it has all components fixed, except for the axiom of τ_1, and can be "programmed" to simulate any type-0 grammar. Indeed, take an arbitrary type-0 grammar G and construct its code, $w(G)$ as in [3]. Introduce the string $w(G)XXE$ as axiom of the first tube of Γ_u. Denote by $\Gamma_u(w(G))$ the obtained system. Then $L(\Gamma_u(w(G))) = L(G)$. Consequently, we have

Theorem 2. *For each alphabet T, there is a two-level H system with three components, which is universal with respect to the type-0 grammars having the terminal alphabet T.*

5. Concluding remarks

We have introduced a distributed system based on the splicing operation, inspired from architectures much investigated in grammar systems area. The model is quite homogeneous and it is computationally complete. Moreover, universal distributed systems exist; the "program" to be added to such a "computer" consists of only one string. Three components are enough, all their elements being finite.

After Adleman shocking experiment, [1], a series of papers were written about the possibility of constructing universal DNA computers. Our work differs from most of them by the fact that (1) we explicitly look for universality results like Theorem 2, without confining ourselves to merely simulating any given Turing machine, (2) we are not concerned with the biochemical details of such a simulation, by definition such details being in a rapid development, and (3), maybe the most important, we *prove* our statements.

References

[1] L. M. Adleman, Molecular computation of solutions to combinatorial problems, *Science*, 226 (Nov. 1994), 1021 – 1024.

[2] L. M. Adleman, On constructing a molecular computer, in *DNA Based Computers* (R. J. Lipton, E. B. Baum, eds.), Proc. of a DIMACS Workshop, Princeton, 1995, Amer. Math. Soc., 1996, 1 – 22.

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

[4] J. Collado-Vides, The search for a grammatical theory of gene regulation is formally justified by showing the inadequacy of context-free grammars, *CABIOS*, 7 (1991), 321 – 326.

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

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

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

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

[9] E. Fahy, G. R. Davis, L. J. DiMichele, S. S. Ghosh, Design and synthesis of polyacrylamino-based oligonucleotide supports for use in nucleic acid diagnostics, *Nucleic Acids Research*, 21 (1993), 1819 – 1826.

[10] R. Freund, L. Kari, Gh. Păun, DNA computing based on splicing: the existence of universal computers, Technical Report 185-2/FR-2/95, Technical Univ. Wien, 1995.

[11] D. Hauschild, M. Jantzen, Petri nets algorithms in the theory of matrix grammars, *Acta Informatica*, 31 (1994), 719 – 728.

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

[13] T. Head, Gh. Păun, D. Pixton, Language theory and molecular genetics, Generative mechanisms suggested by DNA recombination, chapter 7 in volume 2 of *Handbook of Formal Languages* (G. Rozenberg, A. Salomaa, eds.), Springer-Verlag, Heidelberg, 1997, 295 – 360.

[14] R. J. Lipton, Speeding up computations via molecular biology, in *DNA Based Computers* (R. J. Lipton, E. B. Baum, eds.), Proc. of a DIMACS Workshop, Princeton, 1995, Amer. Math. Soc., 1996, 67 – 74.

[15] Gh. Păun, Splicing. A challenge for formal language theorists, *Bulletin of the EATCS*, 57 (1995), 183 – 194.

[16] Gh. Păun, On the splicing operation, *Discrete Appl. Math.*, 70 (1996), 57 – 79.

[17] Gh. Păun, On the power of the splicing operation, *Intern. J. Computer Math.*, 59 (1995), 27 – 35.

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

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

[20] Gh. Păun, G. Rozenberg, A. Salomaa, *DNA Computing. New Computing Paradigms*, Springer-Verlag, Heidelberg, 1998.

[21] Gh. Păun, L. Santean, Parallel communicating grammar systems: the regular case, *Ann. Univ. Buc., Matem.-Inform. Series*, 38, 2 (1989), 55 – 63.

[22] D. Pixton, Regularity of splicing languages, *Discrete Appl. Math.*, 1995.

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

[24] A. M. Turing, On computable numbers, with an application to the Entscheidungsproblem, *Proc. London Math. Soc.*, Ser. 2, 42 (1936), 230 – 265; a correction, 43 (1936), 544 – 546.

[25] D. B. Searls, The linguistics of DNA, *American Scientist*, 80 (1992), 579 – 591.

[26] T. Yokomori, S. Kobayashi, C. Ferretti, On the power of circular splicing systems and DNA computability, *Report CSIM 95-01*, Univ. of Electro-Comm., Chofu, Tokyo, 1995.