# Test Tube Distributed Systems Based on Splicing[1]

Erzsébet CSUHAJ-VARJÚ
Computer and Automation Institute
Hungarian Academy of Sciences
Kende utca 13 – 17, 1111 - Budapest, Hungary

Lila KARI
Department of Mathematics and Computer Science
University of Western Ontario
London, Ontario, N6A 5B7 Canada

Gheorghe PĂUN[2]
Institute of Mathematics of the Romanian Academy
PO Box 1 – 764, 70700 Bucureşti, Romania

**Abstract.** We define a symbol processing mechanism with the components (test tubes) working as splicing schemes in the sense of T. Head and communicating by redistributing the contents of tubes (in a similar way to the *separate* operation of Lipton-Adleman). (These systems are similar to the distributed generative mechanisms called Parallel Communicating Grammar Systems.) Systems with finite initial contents of tubes and finite sets of splicing rules associated to each component are computationally complete, they characterize the family of recursively enumerable languages. The existence of universal test tube distributed systems is obtained on this basis, hence the theoretical proof of the possibility to design universal programmable computers with the structure of such a system.

**Keywords.** DNA computing, Parallel communicating grammar systems, Turing machines, universality

[1]Research supported by the Academy of Finland, Project 11281, the Hungarian Scientific Research Fund OTKA grant no. T017105, and grant OGP0007877 of the Natural Sciences and Engineering Research Council of Canada

[2]All correspondence to this author. Current address: Turku Centre for Computer Science TUCS, Data City, Lemminkaisenkatu 14 A, 4th floor, FIN - 20520, Finland; E-mail: paun@sara.utu.fi

## 1. Introduction

This paper can be seen as a continuation of some previous papers investigating the power of the splicing operation and proving that certain classes of H systems are computationally complete (they have the same power as Turing machines) – see [11], [8] and their references. However, a new idea is brought here into the DNA computing area, that of distributed computing, in the sense of grammar systems theory (see [4]). More precisely, we introduce here distributed splicing systems working in a way similar to parallel communicating (PC) grammar systems of [15]. In this way we bring together the idea of splicing, in the sense of [7], that of *test tube computing*, in the sense of [2], [9], and that of a PC grammar system.

The splicing operation, as introduced in [7], is a model of the recombinant behavior of DNA under the influence of restriction enzymes and lygases. One gives pairs of the form $(u_1, u_2), (u_3, u_4)$, encoding the sites where enzymes can cut the DNA sequences, and pairs of such pairs – in total we have in this way quadruples $(u_1, u_2; u_3, u_4)$ – specifying the possibility to paste substrings obtained after such a cutting. Thus, from two strings $x, y$ having $u_1u_2, u_3u_4$ as substrings, that is $x = x_1u_1u_2x_2, y = y_1u_3u_4y_2$, we first obtain the substrings $x_1u_1, u_2x_2, y_1u_3, u_4y_2$, then we can 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 string are (possibly) new. We say that we have spliced $x, y$ and we have obtained $w$ and $z$, where $w = x_1u_1u_4y_2$, $z = y_1u_3u_2x_2$.

The splicing operation has been investigated in a series of papers; we refer to [11] and to [8] for bibliographical information (and surveys of results). For our purposes, important is the notion of an *extended H system*, a generative mechanism based on splicing, introduced in [14], and which turned out to be computationally complete, its power is equal to the power of Turing machines for certain rather weak variants; see [13], [6]. In particular, in [6] one shows that universal H systems exist, with finite components (universal here is understood in the same sense as for universal Turing machines: systems with all but one component fixed and which can behave as any particular H system when a code of the particular one is added to the non-fixed component).

In 1994, Adleman has shocked the computer science community by solving (a small instance of) the Hamiltonian path problem in a graph by DNA manipulation in a test tube. Extensions of this procedure were proposed by [10], [9]. Continuing the ideas in [9], [2] proposes a sort of "programming language" (a formalism) for writing algorithms dealing with test tubes. In fact, the basic primitive of this formalism is the *tube*, a set (or a multiset, with multiplicites associated to its elements) of strings over a given alphabet. The basic operations of this formalism describe operations with tubes: *merge* (put together the contents of two tubes), *separate* (produce two tubes from one, according to specified criteria), *amplify* (duplicate a tube), *check* whether a tube is empty or not. We shall use here mainly the *separate* operation, with a specific definition.

The parallel communicating (PC) grammar systems consist of several usual Chomsky grammars, working synchronously, each one on its own sentential form (initially equal to the axiom) and communicating by request in the following way: there are special symbols used only for starting communications (and called *query symbols*); when such a symbol is introduced by a component $i$ and pointing to another component $j$, then the current sentential form of the component $j$ is transmitted to component $i$. Thus, there are two types of steps of the work of such a system: *rewriting steps* (componentwise derivations) and *communication steps*. One component of the system is designed as the master, and the language generated by it, with or without the help of other components, is the language generated by the system. Details can be found in [4].

Here we consider a sort of PC grammar systems whose components are tubes in the sense of [2], working as splicing systems, and communicating their contents in the sense of the *separate* operation; more specifically, the result of the iterated splicing of the contents of each tube is redistributed to all tubes according to certain *selector* alphabets associated to the tubes: a string consisting of symbols in a selector alphabet will be transmitted to the tubes having that selector associated with them. Again one tube is designed as the master and its contents put together is the result of the system "computation". We call such a system a *test tube distributed system*, shortly, a TT system.

The large power of the splicing operation is confirmed also in this framework: TT systems with finite components characterize the recursively enumerable languages; more precisely, $n+8$ components are enough for generating all languages over an alphabet with $n$ symbols (whereas systems with two components can generate non-regular languages, systems with three components can generate non-context-free languages, and six components are enough for generating non-recursive languages). The proof directly entails the existence of universal TT systems. Therefore, a result as in [6] is obtained for this new mechanism: one can design universal programmable DNA computers based on splicing. In our case, the "computer" is a TT parallel communicating system able to simulate any other TT system (any Turing machine), after introducing its "program" in the "computer" as an axiom of a certain component. Of course, this is only a theoretical result, the possibility to practically implement the operations on which a TT system is based – *splicing* and *separation* – is a technical/biological question.

## 2. Basic definitions; the splicing operation

We use the following formal language notations: $V^*$ is the free monoid generated by the alphabet $V$, $\lambda$ is the empty word, *FIN, REG, CF, CS, RE* are the families of finite, regular, context-free, context-sensitive, and recursively enumerable languages, respectively. For general formal language theory prerequisites we refer to [18]; for grammar systems area we refer to [4].

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

$$\begin{aligned}\sigma(L) = \{w \in V^* \quad | \quad & (x, y) \vdash_r (w, z) \text{ or } (x, y) \vdash_r (z, w),\\ & \text{for some } x, y \in L, r \in R\}.\end{aligned}$$

(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 $\gamma$. The language generated by $\gamma$ 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 $\gamma = (V, T, A, R)$ with $V = T$ 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 [7] for finite sets of rules; the case of arbitrarily large sets of splicing rules is considered in [12]; the extended H systems were introduced in [14]. Details can be found in [8], [11].

For instance, in [5], [16] it is proved that

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

(The inclusion is, in fact, proper.) Using this relation, in [14] it is proved that

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

Moreover, in [13] 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).

## 3. Test tube systems

A *test tube* (TT, for short) *system* (of degree $n, n \geq 1$) is a construct

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

where $V$ is an alphabet, $A_i \subseteq V^*$, $R_i \subseteq V^*\#V^*\$V^*\#V^*$, and $V_i \subseteq V$, for each $1 \leq i \leq n$.

Each triple $(A_i, R_i, V_i)$ is called a *component* of the system, or a *tube*; $A_i$ is the set of axioms of the tube $i$, $R_i$ is the set of splicing rules of the tube $i$, $V_i$ is the *selector* of the tube $i$.

We denote

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

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

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

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

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

In words, the contents of each tube is spliced according to the associated set of rules (we pass from $L_i$ to $\sigma_i^*(L_i), 1 \leq i \leq n$), and the result is redistributed among the $n$ tubes according to the selectors $V_1, \ldots, V_n$; the part which cannot be redistributed (does not belong to some $V_i^*, 1 \leq i \leq n$) remains in the tube. Because we have imposed no restrictions over the alphabets $V_i$, for example, we did not supposed that they are pairwise disjoint, when a string in $\sigma_j^*(L_j)$ belongs to several languages $V_i^*$, then copies of this string will be distributed to all tubes $i$ with this property.

A *computation* (of length $k, k \geq 0$) in $\Gamma$ is a sequence of configurations

$$C = \{(L_1^{(t)}, \ldots, L_n^{(t)}) \mid 0 \leq t \leq k\}$$

such that

$$\begin{array}{ll} (i) & (L_1^{(0)}, \ldots, L_n^{(0)}) = (A_1, \ldots, A_n), \\ (ii) & (L_1^{(t)}, \ldots, L_n^{(t)}) \Longrightarrow (L_1^{(t+1)}, \ldots, L_n^{(t+1)}), \text{ with respect to } \Gamma, \\ & \text{for each } t, 0 \leq t \leq k-1. \end{array}$$

We denote by $C_k(\Gamma)$ the set of all computations of length $k, k \geq 0$, of $\Gamma$, and by $C_*(\Gamma)$ the set of all possible computations,

$$C_*(\Gamma) = \bigcup_{k \geq 0} C_k(\Gamma),$$

where $C_0(\Gamma) = \{(A_1, \ldots, A_n)\}$. The $i$th *result* of a computation $C = \{(L_1^{(t)}, \ldots, L_n^{(t)}) \mid 0 \leq t \leq k\}$ is the set of all strings which were present in the tube $i$, that is

$$\rho_i(C) = \bigcup_{0 \leq t \leq k} L_i^{(t)}.$$

One of the components of a TT system $\Gamma$ is designed as the *master* one and the union of its contents in all possible computations is the result of the system computations. By convention, we consider that the first tube is the master, hence the result of the work of $\Gamma$, the *language generated* by $\Gamma$, is

$$L(\Gamma) = \bigcup_{C \in C_*(\Gamma)} \rho_1(C).$$

We can also write, in a way closer to the style of formal language theory,

$$\begin{aligned} L(\Gamma) = \{w \in V^* \quad | \quad & w \in L_1^{(t)} \text{ for some} \\ & (A_1, \ldots, A_n) \Longrightarrow^* (L_1^{(t)}, \ldots, L_n^{(t)}), t \geq 0\}, \end{aligned}$$

where $\Longrightarrow^*$ is the reflexive and transitive closure of the relation $\Longrightarrow$.

Given two families of languages, $F_1, F_2$, we denote by $TT_n(F_1, F_2)$ the family of languages $L(\Gamma)$, for $\Gamma = (V, (A_1, R_1, V_1), \ldots, (A_m, R_m, V_m))$, with $m \leq n$, $A_i \in F_1, R_i \in F_2$, for each $i, 1 \leq i \leq m$. (We say that $\Gamma$ is of type $(F_1, F_2)$.) When $n$ is not specified, we replace it by $*$, that is we write

$$TT_*(F_1, F_2) = \bigcup_{n \geq 1} TT_n(F_1, F_2).$$

A TT system as above has a structure very similar to that of a parallel communicating grammar system. The rewriting steps in a PC grammar system correspond here to the splicing phases, that is to passing from $L_i^{(t)}$ to $\sigma^*(L_i^{(t)})$, whereas the communication steps correspond to the redistribution of $\sigma^*(L_i^{(t)})$ to the tubes according to the selectors $V_1, \ldots, V_n$. However, in a PC grammar system the communication is done *on request*: the receiving component starts the communication, by introducing a query symbol; here the communication is automatically performed after every splicing step. Note that a splicing step is an iterated one (maximal as regards the produced output), not a single rewriting step as in a PC grammar system.

Also note that in some sense we have used here three of the basic operations in Adleman's "programming language" ([2], [9]): the *separate* operation when redistributing the contents of a tube according to the selectors, the *merge* operation when constituting the new tube contents, as well as the *amplify* operation, when copies of the same string is sent to several tubes at the same time. The systems we obtain prove to be computationally complete, equivalent in power to Turing machines. On the other hand, we use a basic new ingredient: the splicing operation, which we already know that it is quite powerful ([13], [6], etc.).

## 4. An example

We illustrate the definitions above with an example discussed in some details. Consider the system

$$\Gamma = (V, (A_1, R_1, V_1), \ldots, (A_4, R_4, V_4)),$$

with

$$\begin{array}{lll}
V = \{a, b, c, d, e, f, g\}, & & \\
A_1 = \{cabd,\ gbe\}, & R_1 = \{b\#d\$g\#be\}, & V_1 = \{a, b, c, d\}, \\
A_2 = \{fag\}, & R_2 = \{fa\#g\$c\#a\}, & V_2 = \{a, b, c, e\}, \\
A_3 = \{gd\}, & R_3 = \{b\#e\$g\#d\}, & V_3 = \{a, b, e, f\}, \\
A_4 = \{cg\}, & R_4 = \{c\#g\$f\#a\}, & V_4 = \{a, b, d, f\}.
\end{array}$$

We start from the configuration

$$(A_1, A_2, A_3, A_4) = (\{cabd, gbe\}, \{fag\}, \{gd\}, \{cg\}).$$

Because we can perform only one splicing, in tube 1, $(cab|d, g|be) \vdash (cab^2e, gd)$, we have

$$\begin{aligned}
\sigma_1^*(A_1) &= A_1 \cup \{cab^2e, gd\},\\
\sigma_2^*(A_2) &= A_2,\\
\sigma_3^*(A_3) &= A_3,\\
\sigma_4^*(A_4) &= A_4.
\end{aligned}$$

(The vertical bars in $(cab|d, g|be)$ are added for an easier readability.) The string $cab^2e$ has to be transmitted to the second tube, $gd$ will remain in tube 1, hence the next configuration is

$$\begin{aligned}
&(L_1^{(1)}, L_2^{(1)}, L_3^{(1)}, L_4^{(1)}) =\\
&= (\{cabd, gbe, gd\}, \{cab^2e, fag\}, \{gd\}, \{cg\}).
\end{aligned}$$

Now, tube 1 can repeat the previous splicing, but also a splicing in tube 2 is possible:

$$\begin{aligned}
\sigma_1^*(L_1^{(1)}) &= L_1^{(1)} \cup \{cab^2e\},\\
\sigma_2^*(L_2^{(1)}) &= L_2^{(1)} \cup \{fa^2b^2e, cg\},\\
\sigma_3^*(L_3^{(1)}) &= L_3^{(1)},\\
\sigma_4^*(L_4^{(1)}) &= L_4^{(1)}.
\end{aligned}$$

The string $cab^2e$ will be again transmitted from tube 1 to tube 2, but tube 2 contains already this string. The string $fa^2b^2e$ will be moved from tube 2 to tube 3, hence

$$\begin{aligned}
&(L_1^{(2)}, L_2^{(2)}, L_3^{(2)}, L_4^{(2)}) =\\
&= (\{cabd, gbe, gd\}, \{cab^2e, fag, cg\}, \{fa^2b^2e, gd\}, \{cg\}).
\end{aligned}$$

We obtain

$$\begin{aligned}
\sigma_1^*(L_1^{(2)}) &= L_1^{(2)} \cup \{cab^2e\},\\
\sigma_2^*(L_2^{(2)}) &= L_2^{(2)} \cup \{fa^2b^2e\},\\
\sigma_3^*(L_3^{(2)}) &= L_3^{(2)} \cup \{fa^2b^2d, ge\},\\
\sigma_4^*(L_4^{(2)}) &= L_4^{(2)},
\end{aligned}$$

hence

$$\begin{aligned}
&(L_1^{(3)}, L_2^{(3)}, L_3^{(3)}, L_4^{(3)}) =\\
&= \{cabd, gbe, gd\}, \{cab^2e, fag, cg\}, \{fa^2b^2e, gd, ge\}, \{fa^2b^2d, cg\}).
\end{aligned}$$

Therefore,

$$\begin{aligned}
\sigma_1^*(L_1^{(3)}) &= L_1^{(3)} \cup \{cab^2e\},\\
\sigma_2^*(L_2^{(3)}) &= L_2^{(3)} \cup \{fa^2b^2e\},\\
\sigma_3^*(L_3^{(3)}) &= L_3^{(3)} \cup \{fa^2b^2d\},\\
\sigma_4^*(L_4^{(3)}) &= L_4^{(3)} \cup \{ca^2b^2d, fg\},
\end{aligned}$$

hence

$$\begin{aligned}
&(L_1^{(4)}, L_2^{(4)}, L_3^{(4)}, L_4^{(4)}) =\\
&= (\{cabd, ca^2b^2d, gbe, gd\}, \{cab^2e, fag, cg\}, \{fa^2b^2e, gd, ge\}, \{fa^2b^2d, cg, fg\}.
\end{aligned}$$

One can see that the system works as follows:

- The strings containing the symbol $g$ are never moved from a tube to another one.
- The first component adds one $b$ to strings of the form $ca^ib^jd$, and replaces $d$ by $e$. The obtained string is transmitted to the second tube, which adds one $a$ and replaces $c$ by $f$. The obtained string is passed to the third tube which replaces $e$ by $d$. The result is a string of the form $fa^{i+1}b^{j+1}d$, hence it will be transmitted to tube 4. Here, $f$ is replaced by $c$. In this way, we pass from $ca^ib^jd$ to $ca^{i+1}b^{j+1}d$, hence the process can be iterated. Because we start from $cabd$, we can obtain in this way all strings $ca^nb^nd, n \geq 1$.
- All the splicing rules involve a symbol $g$, but no one of the strings containing $g$ and not being an axiom ($gd, cg, ge, fg$ in tubes 1, 2, 3, 4, respectively) can be used in a splicing. Thus, at every splicing we have to use one of the axioms $gbe, fag, gd, cg$, that is we cannot splice two strings obtained at a previous splicing and not containing the symbol $g$. In this way, the strings of the form $ca^nb^md$ obtained in the first tube have $n = m$.

Consequently,

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

which is not a regular language.

Notice that $\Gamma$ is of type $(FIN, FIN)$, hence we have obtained

$$TT_4(FIN, FIN) - REG \neq \emptyset.$$

(We shall improve this result below.) Compare this result with the relation $H(FIN, FIN) \subseteq REG$ [5], [16]; we conclude that the distributed mode of working in a TT system, the cooperation of the components, is productive, it strictly increases the power of H systems.

In fact, the TT systems with finite components are computationally complete:

## 5. Characterizing recursively enumerable languages

The following inclusions are obvious:

**Lemma 1.** (i) $TT_n(F_1, F_2) \subseteq TT_{n+1}(F_1, F_2)$, *for all* $n \geq 1$ *and* $F_1, F_2$.
(ii) $TT_n(F_1, F_2) \subseteq TT_n(F_1', F_2')$, *for all* $n \geq 1$ *and* $F_1 \subseteq F_1', F_2 \subseteq F_2'$.

¿From Turing-Church thesis (or by a direct proof), we get

**Lemma 2.** $TT_*(F_1, F_2) \subseteq RE$, *for all* $F_1, F_2 \subseteq RE$.

By definition, we also have

**Lemma 3.** $H(F_1, F_2) = TT_1(F_1, F_2)$, *for all* $F_1, F_2$.

The following relation is the core result of this paper.

**Lemma 4.** $RE \subseteq TT_*(FIN, FIN)$.

*Proof.* Take a type-0 grammar $G = (N, T, S, P)$ with $T = \{a_1, \ldots, a_n\}$ and $N = \{Z_1, \ldots, Z_m\}, Z_1 = S$. Consider the new symbols $A, B$ and replace in each rule of $P$ each occurrence of $Z_i$ by $BA^iB, 1 \leq i \leq m$. We obtain a set $P'$ of rules such that $S \Longrightarrow^* w, w \in T^*$, according to the grammar $G$ if and only if $BAB \Longrightarrow^* w$ using the rules in $P'$. For uniformity, we denote $A = a_{n+1}, B = a_{n+2}$. Construct the TT system

$$\begin{aligned}\Gamma \quad = \quad & (V, (A_1, R_1, V_1), (A_2, R_2, V_2), (A_{3,1}, R_{3,1}, V_{3,1}), \ldots, (A_{3,n+2}, R_{3,n+2}, V_{3,n+2}), \\ & (A_4, R_4, V_4), (A_5, R_5, V_5), (A_6, R_6, V_6), (A_7, R_7, V_7)),\end{aligned}$$

with

$$V = T \cup \{A, B, X, X', Y, Z, Z'\} \cup \{Y_i \mid 1 \leq i \leq n+2\},$$

and

$$A_1 = \emptyset,$$
$$R_1 = \emptyset,$$
$$V_1 = T,$$

$$\begin{aligned}A_2 \quad &= \quad \{XBA^{m+1}BBABY, Z'Z\} \cup \\ & \quad\quad \{ZvY \mid u \to v \in P'\} \cup \\ & \quad\quad \{ZY_i \mid 1 \leq i \leq n+2\}, \\ R_2 \quad &= \quad \{\#uY\$Z\#vY \mid u \to v \in P'\} \cup \\ & \quad\quad \{\#a_iY\$Z\#Y_i \mid 1 \leq i \leq n+2\} \cup \\ & \quad\quad \{Z'\#Z\$XBA^{m+1}B\#\}, \\ V_2 \quad &= \quad T \cup \{A, B, X, Y\},\end{aligned}$$

$$\begin{aligned}A_{3,i} \quad &= \quad \{X'a_iZ\}, \\ R_{3,i} \quad &= \quad \{X'a_i\#\$X\# \mid 1 \leq i \leq n+2\}, \\ V_{3,i} \quad &= \quad T \cup \{A, B, X, Y_i\}, \text{ for } 1 \leq i \leq n+2,\end{aligned}$$

$$
\begin{aligned}
A_4 &= \{ZY\},\\
R_4 &= \{\#Y_i\$Z\#Y \mid 1 \le i \le n+2\},\\
V_4 &= T \cup \{A, B, X'\} \cup \{Y_i \mid 1 \le i \le n+2\},\\
\\
A_5 &= \{XZ\},\\
R_5 &= \{X\#Z\$X'\#\},\\
V_5 &= T \cup \{A, B, X', Y\},\\
\\
A_6 &= \{ZZ\},\\
R_6 &= \{\#Y\$ZZ\#\},\\
V_6 &= T \cup \{Y, Z'\},\\
\\
A_7 &= \{ZZ\},\\
R_7 &= \{\#ZZ\$Z'\#\},\\
V_7 &= T \cup \{Z'\}.
\end{aligned}
$$

Let us examine the work of $\Gamma$.

The first component only selects the strings produced by the other components and which are terminal according to $G$. No such terminal string can enter a splicing, because all rules in $R_2$ – $R_7$ involve symbols $X, Y, X', Z, Y_i$, for $1 \le i \le n$. (When speaking about $R_3$ we mean the union of all sets $R_{3,i}, 1 \le i \le n$.)

In the initial configuration $(A_1, \ldots, A_7)$, only the second component can execute a splicing. There are three possibilities: to use a rule of the form $\#uY\$\#vY$, for $u \to v \in P'$ (we say that this is a splicing of type 1), a rule of the form $\#a_iY\$Z\#Y_i$, for $1 \le i \le n+2$ (a splicing of type 2), or the rule $Z'\#Z\$XBA^{m+1}B\#$ (a splicing of type 3).

Consider the general case, of having in tube 2 a string $XwY$, with $w \in (T \cup \{A, B\})^* BA^{m+1}B(T \cup \{A, B\})^*$ (initially, $w = BA^{m+1}BBAB$). We have three possible splicings:

1. $(Xw_1|uY, Z|vY) \vdash_1 (Xw_1vY, ZuY)$, for $\to v \in P'$ providing that $w = w_1u$,
2. $(Xw_1|a_iY, Z|Y_i) \vdash_2 (Xw_1Y_i, Za_iY)$, $1 \le i \le n+2$, when $w = w_1a_i$,
3. $(Z'|Z, XBA^{m+1}B|w_1Y) \vdash_3 (Z'w_1Y, XBA^{m+1}BZ)$, for $w = BA^{m+1}Bw_1$.

The string $Xw_1vY$ is of the same form with $Xw_1uY$ and it will remain in tube 2, entering new splicings of one of the three types. Clearly, the passing from $Xw_1uY$ to $Xw_1vY$ corresponds to using the rule $u \to v$ in $P'$ on a suffix of the string bracketed by $X, Y$.

The string $ZuY$ will remain in tube 2, too. Such a string $ZuY$ can enter a splicing in three cases: (i) if $ZuY$ is already an axiom, hence nothing new appears in this way, (ii) as the first term of a splicing of the form $(Zu_1|u'Y, Z|v'Y) \vdash_1 (Zu_1v'Y, Zu'Y)$, for $u = u_1u'$ and $u' \to v' \in P'$; one obtains two strings of the same form $ZxY$ which will remain in tube 2, (iii) $(Zu_1|a_iY, Z|Y_i) \vdash_2 (Zu_1Y_i, Za_iY)$, for $u = u_1a_i, 1 \leq i \leq n+2$; the string $Zu_1Y_i$ cannot enter new splicings and cannot be transmitted to another tube. After any sequence of such splicings, the obtained strings will still be of the form $ZxY$ hence they will remain in tube 2 and will enter either "legal" splicings, when they are axioms, or splicings still producing "useless" strings $ZyY$.

Therefore, after a series of splicings of type 1, a splicing of type 2 will be eventually performed in tube 2, producing strings of the form $Xw_1Y_i$ and $Za_iY$. The second string behaves exactly as $ZuY$ discussed above. If a string $XwY_i$ enters a new splicing in tube 2, this can be only of type 3, $(Z'|Z, XBA^{m+1}B|w_2Y_i) \vdash_3 (Z'w_2Y_i, XBA^{m+1}BZ)$, for $w_1 = BA^{m+1}Bw_2$. The string $Z'w_2Y_i$ cannot enter new splicings in tube 2 and cannot be transmitted to another tube. The case of $XBA^{m+1}BZ$ will be discussed below. Any string $Xw_1Y_i$ is moved from tube 2 to the corresponding tube $(3, i)$, where we have to perform

$$(X'a_i|Z, X|w_1Y_i) \vdash (X'a_iw_1Y_i, XZ).$$

The second string, $XZ$, remains in tube $(3, i)$ and it will enter only splicings of the form

$$(X'a_i|Z, X|Z) \vdash (X'a_iZ, XZ),$$

hence producing nothing new. The first string cannot enter new splicings in tube $(3, i)$, it will be transmitted to tube 4, where the only possible splicing is

$$(X'a_iw_1|Y_i, Z|Y) \vdash (X'a_iw_1Y, ZY_i).$$

Again the second string remains in the tube and the possible splicings using it will produce nothing new, whereas the first string will be moved into tube 5. There we obtain

$$(X|Z, X'|a_iw_1Y) \vdash (Xa_iw_1Y, X'Z).$$

The second string remains in tube 5, and it produces nothing new, the first one has to be communicated to tube 2. We started from $Xw_1a_iY$ and we returned to tube 2 the string $Xa_iw_1Y$. A symbol from the right-hand end of the string bracketed by $X, Y$ has been moved to the left-hand end. In this way the string bracketed by $X$, $Y$ can be circularly permuted as long as we want. In this way, we can "rewind" the string till having as a suffix the left-hand member of any rule in $P'$ we want to simulate by a rule in $R_2$ of the form $\#uY\$Z\#vY$.

On the other hand, if we start from a string $XwBA^jBY$, we have to use tubes 2, $(3, n+2)$ or $(3, n+1)$, 4, 5, iteratively, until obtaining $XBA^jBwY$: no rule of $P'$ can be simulated in tube 2 after removing the rightmost occurrence of $B$ in $XwBA^jBY$, and, similarly, we cannot use the rule $Z'\#Z\$XBA^{m+1}B\#$ in $R_2$ after removing the rightmost occurrence of $B$ in $XwBA^jBY$. Because the substring $BA^{m+1}B$ is always present (and exactly one copy of it is present as long as we do not use the rule $Z'\#Z\$XBA^{m+1}B\#$ in $R_2$), we know in every moment where the "actual beginning" of the string is placed. In conclusion, using splicings of type 1 and the rewind technique made possible by the above described passing through tubes 2, 3, 4, 5, we can simulate every derivation according to $P'$.

Conversely, exactly strings $Xw_1BA^{m+1}Bw_2Y$ can be obtained in this way which corresponds to strings $w_2w_1$ which can be obtained starting from $BAB$ and using rules in $P'$.

Consider now the splicing of type 3 in tube 2. If the string $XBA^{m+1}BZ$ is used in further splicings, they are of the form

$$(Z'|Z, XBA^{m+1}B|Z) \vdash (Z'Z, XBA^{m+1}BZ),$$

therefore no new string is obtained in this way.

The first string produced by a splicing of type 3, $Z'w_1Y$, will be transmitted to tube 6; here we have only one possibility

$$(Z'w_1|Y, ZZ|) \vdash (Z'w_1, ZZY).$$

If $ZZY$ will enter new splicings, they are of the form

$$(Z'x|Y, ZZ|Y) \vdash (Z'xY, ZZY),$$
$$(ZZ|Y, ZZ|Y) \vdash (ZZY, ZZY),$$

hence no new string is obtained.

The string $Z'w_1$ cannot enter new splicings in tube 6. If $w_1 \in T^*$ (and only in this case), it will be moved in tube 7, where we perform

$$(|ZZ, Z'|w_1) \vdash (w_1, Z'ZZ).$$

The string $w_1$ is terminal. It will be transmitted to all tubes – including the first one. No splicing can be done on a terminal string. As we have seen above, such a terminal string $w_1$ is a string in $L(G)$.

If the string $Z'w_1Y$ will enter new splicings in tube 2, they can be of forms 1 and 2:

$$(Z'w_2|uY, Z|vY)) \vdash_1 (Z'w_2vY, ZuY), \text{ for } u \to v \in P', w_1 = w_2u,$$
$$(Z'w_2|a_iY, Z|Y_i) \vdash_2 (Z'w_2Y_i, Za_iY), \text{ for } 1 \le i \le n, w_1 = w_2a_i.$$

The behavior of $ZuY, Za_iY, Z'w_2Y_i$ is known, similar strings appeared in the previous discussion. The string $Z'w_2vY$ can be obtained by performing first $(XBA^{m+1}Bw_2|uY, Z|vY) \vdash_1 (XBA^{m+1}Bw_2vY, ZuY)$ and then $(Z'|Z, XBA^{m+1}B|w_2vY) \vdash_3 (Z'w_2vY, XBA^{m+1}BZ)$, hence also this string is a "legal" one. No parasitic string can reach the first tube, consequently, $L(\Gamma) = L(G)$. □

The above proof is based on the same idea as the proof of the inclusion $RE \subseteq EH(FIN, REG)$ in [13]. The same idea is used in [6] in order to prove that every type-0 grammar can be simulated by an extended H system of type $(FIN, FIN)$, providing that some control on the use of the splicing rules is considered. Namely, a random context-like control is used: each splicing rule has associated certain symbols and it can be applied to a pair of strings only when the associated symbols appear in these strings (the permitting context variant), or they do not appear (the forbiding context variant). Therefore, both here and in [6] we have to pay for the passing from $REG$ to $FIN$, as type of the sets of rules, by imposing some regulation on the rule using. In view of the result in [5], [16], this cannot be avoided: extended H systems of type $(FIN, FIN)$ generate only regular languages.

On the other hand, the condition we use here for regulation – the separation of strings according to their membership to alphabets $V_1, \ldots, V_n$ – although similar, is weaker than the random context condition: a string $w$ will be transmitted to tube $i$ when $w \in V_i^*$ without knowing that *all* symbols in $V_i$ are present in $w$ (but knowing indeed that no symbol outside $V_i$ appears in $w$). Another essential difference between a TT system and an extended H system with random context conditions is that we perform here an iterated splicing in each tube, thus applying the tube splicing rules as long as we can, even to strings which can be transmitted to another tubes. This seems to be more realistic than checking random context conditions after each single splicing.

Moreover, we have

**Lemma 5.** $EH(F_1, F_2) \subseteq TT_2(F_1, F_2)$.

*Proof.* For an extended $H$ systems $\gamma = (V, T, A, R)$, consider the $TT$ system

$$\Gamma = (V, (\emptyset, \emptyset, T), (A, R, V)).$$

Obviously, $L(\gamma) = L(\Gamma)$ : the second component of $\Gamma$ simulates the work of $\gamma$ (the splicing operation, starting from $A$) and the first component simply selects the strings in $T^*$ produced by the second component. □

**Lemma 6.** $TT_2(FIN, FIN) - REG \neq \emptyset$.

*Proof.* Consider the $TT$ system

$$\begin{aligned} \Gamma \;=\; & (\{a,b,c,d,e\}, (\{cabc, ebd, dae\}, \{b\#c\$e\#bd, da\#e\$c\#a\}, \{a,b,c\}), \\ & (\{ec, ce\}, \{b\#d\$e\#c, c\#e\$d\#a\}, \{a,b,d\}). \end{aligned}$$

We obtain

$$L(\Gamma) \cap ca^+b^+c = \{ca^nb^nc \mid n \geq 1\} \qquad (*)$$

Indeed, the only possible splicings in the first component are

$$\begin{aligned} &(\alpha a^i b^j | c, e | bd) \vdash_1 (\alpha a^i b^{j+1} d, ec), \quad \alpha \in \{c, d\}, \\ &(da | e, c | a^i b^j \alpha) \vdash_2 (da^{i+1} b^j \alpha, ce), \quad \alpha \in \{c, d\}. \end{aligned}$$

One further occurrence of $a$ and one further occurrence of $b$ can be added in this way; only when both one $a$ and one $b$ are added, hence the obtained string is $da^{i+1}b^{j+1}d$, we can move this string to the second tube. Here the possible splicings are

$$(\alpha a^i b^j | d, e | c) \vdash_1 (\alpha a^i b^j c, ed), \quad \alpha \in \{c, d\},$$
$$(c | e, d | a^i b^j \alpha) \vdash_2 (c a^i b^j \alpha, de), \quad \alpha \in \{c, d\}.$$

Only the string $ca^i b^j c$ can be moved to the first tube, hence the process can be iterated. No splicing in tube 1 can involve a string of the form $da^i b^j d$.

Consequently, we have the equality $(*)$, which proves that $L(\Gamma)$ is not a regular language. □

Synthesizing the previous lemmas (and using the fact that $H(FIN, FIN) \subset REG = EH(FIN, FIN)$, strict inclusion), we get

**Theorem 1.**
*$TT_1(FIN, FIN) \subset REG \subset TT_2(FIN, FIN) \subseteq TT_3(FIN, FIN) \subseteq \ldots \ldots \subseteq TT_*(FIN, FIN) = TT_*(F_1, F_2) = RE$, for all $F_1, F_2$ such that $FIN \subseteq F_i \subseteq RE$, $i = 1, 2$.*

**Open problem.** Which of the inclusions $TT_n(FIN, FIN) \subseteq TT_{n+1}(FIN, FIN)$ in Theorem 1 are proper ? (Is this hierarchy infinite ?)

¿From the proof of Lemma 4 we obtain

**Corollary 1.** *If $L \in RE, L \subseteq V^*, card(V) = n$, then $L \in TT_{m+8}(FIN, FIN)$.*

**Corollary 2.** *For every family $F$ such that $F \subset RE$ and $F$ is closed under intersection with regular sets, inverse morphisms, and restricted morphisms (or right and left derivatives) we have*

$$TT_7(FIN, FIN) - F \neq \emptyset.$$

*Proof.* Take a language $L \in RE - F$, $L \subseteq \{a, b\}^*$, and a type-0 grammar $G$ for $L$. (There is such a language: for $L' \subseteq V^*, L' \in RE$, with $V = \{a_1, \ldots, a_n\}$, consider the morphism $h : V^* \longrightarrow \{a, b\}^*$ defined by $h(a_i) = ba^i b, 1 \leq i \leq n$. The language $h(L')$ cannot be in the family $F$, because $L' = h^{-1}(h(L'))$, which, by the closure of $F$ under inverse morphisms, would imply that $L' \in F$, a contradiction. Take $L = h(L')$.)

We construct the system $\Gamma$ as in the proof of Lemma 4, starting from $G$, then we consider

$$\Gamma' = (V, (A_1', R_1', V_1'), \ldots, (A_7', R_7', V_7'))$$

taking

$(A'_1, R'_1, V'_1) = (A_2, R_2, V_2)$, $(A'_2, R'_2, V'_2) = (A_{3,1}, R_{3,1}, V_{3,1})$,
$(A'_3, R'_3, V'_3) = (A_{3,2}, R_{3,2}, V_{3,2})$, $(A'_4, R'_4, V'_4) = (A_{3,3}, R_{3,3}, V_{3,3})$,
$(A'_5, R'_5, V'_5) = (A_{3,4}, R_{3,4}, V_{3,4})$, $(A'_6, R'_6, V'_6) = (A_4, R_4, V_4)$,
$(A'_7, R'_7, V'_7) = (A_5, R_5, V_5)$,

where $(A_i, R_i, V_i)$ and $(A_{3,j}, R_{3,j}, V_{3,j})$ are components of $\Gamma$ in Lemma 4. Now, interchange systematically in the axiom and in the splicing rules of $\Gamma'$ the strings $BAB$ and $BA^{m+1}B$ ($m$ is the number of nonterminals of $G$), that is take $BA^{m+1}B$ as a "code" of the axiom and $BAB$ as marker of the string beginning. Consider also the regular set

$$M = XBAB\{a, b\}^*Y.$$

Because components 6, 7 in $\Gamma$ have only the role of removing the markers $Y, Z'$ (whereas $Z'$ is introduced only when replacing a prefix $XBAB$ of a string $XBABwY$) and component 1 has the role of collecting terminal strings, we have

$$L(\Gamma') \cap M = XBABL(G)Y.$$

By a morphism $h$ we erase all symbols $X, Y, A, B$; the morphism $h$ is restricted (specifically, it is 5-restricted). (Similarly, we can remove the prefix $XBAB$ and the suffix $Y$ of strings in $L(\Gamma') \cap M$ by a left and a right derivative. From the closure properties of $F$ we obtain $L(\Gamma') \notin F$ (otherwise $L(G) \in F$, which is contradictory). □

Because there are recursively enumerable languages on the one-letter alphabet which are non-recursive and the family of recursive languages is closed under left and right derivative, we obtain

**Corollary 3.** *$TT_6(FIN, FIN)$ contains non-recursive languages.*

Moreover, we have

**Lemma 7.** $TT_3(FIN, FIN) - CF \neq \emptyset$.

*Proof.* Consider the TT system

$$\Gamma = (V, (A_1, R_1, V_1), (A_2, R_2, V_2), (A_3, R_3, V_3)),$$

with

$$\begin{aligned}
V &= \{a, b, c, d, e, f\},\\
A_1 &= \{cbac, fd, fe, df, ef\},\\
R_1 &= \{\#ac\$f\#d,\ d\#f\$c\#a,\ d\#f\$c\#b,\\
&\qquad \#bc\$f\#e,\ e\#f\$c\#a,\ e\#f\$c\#b\},\\
V_1 &= \{a, b, c\},\\
A_2 &= \{ca^2f, fc\},\\
R_2 &= \{ca^2\#f\$d\#a,\ ca^2\#f\$d\#b,\ a\#d\$f\#c,\ b\#d\$f\#c\},
\end{aligned}$$

$V_2 = \{a, b, d\}$,
$A_3 = \{cbf, fc\}$,
$R_3 = \{cb\#f\$e\#a,\ a\#e\$f\#c\}$,
$V_3 = \{a, e\}$.

We obtain

$$L(\Gamma) \cap cba^+c = \{cba^{2^n}c \mid n \geq 1\}.$$

Indeed, let us take a string of the form $ca^iba^jc, i + j \geq 1$ (initially we have $i = 0, j = 1$). The following operations are possible in tube 1:

$$(ca^iba^{j-1}|ac, f|d) \vdash_1 (ca^iba^{j-1}d, fac), \text{ for } j \geq 1,$$
$$(ca^i|bc, f|e) \vdash_4 (ca^ie, fbc), \text{ for } j = 0.$$

Moreover, in any string $ca^iba^jc$ or $ca^iba^{j-1}d$ or $ca^ie$ as above, the left-hand occurrence of $c$ can be replaced either by $d$ or by $e$.

After removing both occurrences of $c$, no splicing on the obtained string can be done. If a string contains occurrences of both $c$ and $d$, $c$ and $e$, or $d$ and $e$, it cannot be moved from tube 1. We move to tube 2 the strings of the form $da^iba^{j-1}d$ and to tube 3 the strings of the form $ea^ie$.

In tube 1, the strings $fac, fbc$ can enter splicings of the forms

$$(f|ac, f|d) \vdash_1 (fd, fac),$$
$$(f|bc, f|e) \vdash_4 (fe, fbc),$$

hence no new string is produced in this way. The same for the strings obtained by splicings using rules in $R_1$ replacing the left-hand symbol $c$ by $d$ or by $e$.

In tube 2, a string $da^iba^{j-1}d$ will enter splicings of the form

$$(ca^2|f, d|a^iba^{j-1}\alpha) \vdash_r (ca^{i+2}ba^{j-1}\alpha, df), \text{ for } \alpha \in \{c, d\},$$

where $r$ is one of the first two rules in $R_2$, depending on whether $i > 0$ or $i = 0$,

$$(\alpha a^kba^{j-1}|d, f|c) \vdash_r (\alpha a^kba^{k-1}c, fd), \text{ for } \alpha \in \{c, d\},$$

$k = i$ or $k = i + 2$, and $r$ being one of the last two rules in $R_2$, depending on the value of $j$.

After replacing both occurrences of $d$ by $c$, no further splicing can be performed in tube 2, and the string is transmitted to tube 1. In this way, we pass from $ca^iba^jc$ to $ca^{i+2}ba^{j-1}c$. Continuing in this way we can produce $ca^{i+2j}bc$.

In tube 3, a string $ea^ie$ is transformed into $cba^ic$, which is transmitted to tube 1 (therefore we pass from $ca^ibc$ to $cba^ib$. This makes now possible the interplay of tubes 1 and 2, doubling the number of occurrences of the symbol $a$.

In conclusion, because we start from $cbac$, we can pass from $ca^ibc$ to $cba^ic$ and from $cba^ic$ to $ca^{2i}bc$, iteratively, hence we can produce all strings of the form $cba^{2^n}c, n \geq 0$. Conversely, all the strings present in tube 1 during $\Gamma$ computations and which are of the form $cba^tc, t \geq 1$, have $t = 2^n$ for some $n \geq 0$, which completes the proof. $\square$

Because they show a nice regularity of relationships between families in Chomsky hierarchy and the hierarchy of TT families, we collect the results in Lemmas 6, 7 and in Corollary 3 in a theorem.

**Theorem 2.** $TT_2(FIN, FIN) - REG \neq \emptyset$, $TT_3(FIN, FIN) - CF \neq \emptyset$, $TT_6(FIN, FIN) - CS \neq \emptyset$.

Related to the above mentioned open problem is the question whether the bounds 3, 6 in Theorem 2 can be improved or not. Because the generative power of TT systems proves to be so large, it would be also of interest to consider particular cases, maybe inspired from the theory of PC grammar systems, able, for instance, to generate only regular or only context-free languages.

## 6. The existence of universal TT systems

We understand the notion of a universal TT system in the same sense as for Turing machines (and other equivalent classes of algorithms): to fix all but one components of a system and, giving an arbitrary system, to encode it on the non-fixed component of the universal one in such a way that this particularization of the universal system can simulate the arbitrarily given system.

Examining the construction of the TT system $\Gamma$ in the proof of Lemma 4, we see that this system depends on the elements of the starting grammar $G$. *If the grammar $G$ is a universal type-0 grammar, then $\Gamma$ will be a universal TT system.*

A universal type-0 grammar is a construct $G_U = (N_U, T, -, P_U)$, where $N_U$ is the nonterminal alphabet, $T$ is the terminal alphabet, $P_U$ is the set of rewriting rules; given any grammar $G = (N, T, S', P)$, for a special encoding $w(G)$ of $G$, the grammar $G'_U = (N_U, T, w(G), P_U)$ is equivalent with $G$, that is $L(G) = L(G'_U)$.

A universal type-0 grammar can be obtained from a universal Turing machine [17], using the standard passing from Turing machines to Chomsky grammars. A direct construction of a universal type-0 grammar can be found in [3].

This suggests the following definition: a universal TT system for a given alphabet $T$ is a construct $\Gamma_U = (V_U, (A_{1,U}, R_{1,U}, V_{1,U}), \ldots, (A_{n,U}, R_{n,U}, V_{n,U}))$ with $V_{1,U} = T$, with the components as in a TT system, all of them being fixed, and with the following property: there is a specified $i, 1 \leq i \leq n$, such that if we take an arbitrary TT system $\Gamma$, then there is a set $A_\Gamma \subseteq V^*$ such that the system

$$\Gamma'_U = (V_U, (A_{1,U}, R_{1,U}, V_{1,U}), \ldots, (A_{i,U} \cup A_\Gamma, R_{i,U}, V_{i,U}), \ldots, (A_{n,U}, R_{n,U}, V_{n,U}))$$

is equivalent with $\Gamma$, that is $L(\Gamma'_U) = L(\Gamma)$.

Otherwise stated, encoding $\Gamma$ as new axioms to be added to the $i$th component of $\Gamma_U$, what we obtain is a system equivalent with $\Gamma$.

¿From practical points of view, the main result of our paper is

**Theorem 3.** *For every given alphabet T, there are universal TT systems of degree* $card(T) + 8$ *and of type* $(FIN, FIN)$.

*Proof.* Start the construction of the system $\Gamma$ in the proof of Lemma 4 from a universal type-0 grammar as constructed in [3]. According to the proof of Lemma 4, if $T$ is given, then the alphabet $V$ of $\Gamma$ is fixed,

$$V = T \cup \{A, B, X, X', Y, Z, Z'\} \cup \{Y_i \mid 1 \leq i \leq card(T) + 2\}.$$

Similarly, all other components of $\Gamma$ are fixed. Denote by $\Gamma_U$ the obtained system. Because $G_U$ contains no axiom, the axiom $XBA^{m+1}BBABY$ of the component $A_2$ of $\Gamma_U$ will be omitted, and this is the place where we will add the new axioms, encoding a given TT system.

More precisely, given an arbitrary TT system, $\Gamma_0$, in view of Theorem 1, there is a type-0 grammar $G_0 = (N, T, S, P)$ such that $L(\Gamma_0) = L(G_0)$. Take the code of $G_0$, a string $w(G_0)$ constructed as in [3], and add to $A_2$ the set $A_{\Gamma_0} = \{XBA^{m+1}Bw'(G_0)Y\}$,, where $w'(G_0)$ is obtained from $w(G_0)$ by replacing each nonterminal $Z_i$ with the "code" $BA^iB$, as in the proof of Lemma 4. What we obtain is a system $\Gamma'_U$ such that $L(\Gamma'_U) = L(G_0)$. Indeed, for $G'_U = (N_U, T, w(G_0), P_U)$ we have $L(G'_U) = L(G_0)$. From the construction in the proof of Lemma 4 we have $L(G'_U) = L(\Gamma'_U)$. As $G_0$ is equivalent with the arbitrarily given TT system $\Gamma$, we have $L(\Gamma'_U) = L(\Gamma)$. This proves that $\Gamma_U$ is universal, indeed. □

Observe that the "program" of the particular TT system $\Gamma$ introduced in the universal TT system (which behaves like a computer) consists of only one string, added as an axiom of the second component of the universal system.

The problem formulated after Theorem 1 can be reformulated for the universal systems: can we bound the degree of a universal TT system ? Another (practical) question is to build a universal TT system in a direct way, not using the construction in [3] for universal type-0 grammars (and making the universal TT system able to simulate any Turing machine, directly, not any TT system; this would be closer to the idea of a DNA *computer*, which is supposed to run arbitrary algorithms, not necessarily encoded as TT systems).

**Acknowledgement.** Thanks are due to Z. Fülöp, for very useful remarks about an earlier version of the paper.

## 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, Manuscript in circulation, January 1995.

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

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

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

[6] R. Freund, L. Kari, Gh. Păun, DNA computing based on splicing: the existence of universal computers, submitted, 1995.

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

[8] T. Head, Gh. Păun, D. Pixton, Language theory and molecular genetics, chapter 8 in volume 2 of *Handbook of Formal Languages* (G. Rozenberg, A. Salomaa, eds.), in preparation.

[9] R. J. Lipton, Speeding up computations via molecular biology, Manuscript in circulation, December 1994.

[10] R. J. Lipton, DNA solution of hard computational problems, *Science*, 268 (April 1995), 542 – 545.

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

[12] Gh. Păun, On the splicing operation, *Discrete Appl. Math.*, to appear.

[13] Gh. Păun, Regular extended H systems are computationally universal, *J. Inform. Process. Cybern., EIK*, to appear.

[14] Gh. Păun, G. Rozenberg, A. Salomaa, Computing by splicing, submitted, 1995.

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

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

[17] 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.

[18] A. Salomaa, *Formal Languages*, Academic Press, New York, 1973.