Extended Time-Varying Distributed H Systems - Universality Result

Maurice MARGENSTERN
LITA, Université de Metz
Metz, France

and

Yurii ROGOZHIN
Institute of Mathematics and Computer Science
Chişinău, Moldova

ABSTRACT

The extended time-varying distributed H systems (ETVDH systems shortly) of degree n were introduced by both authors recently as another theoretical model of molecular computing, based on splicing operations. They are as well known time-varying distributed H systems (TVDH systems shortly) of degree n, but the components of ETVDH systems work as H systems, so the behaviour of them is more complicated and more natural than TVDH systems. In the paper of both authors presented at DNA6 conference in Leiden, it was proved that ETVDH systems with one component can generate only regular languages. Now we prove that ETVDH systems with 2 components can generate any recursively enumerable languages. So the border between universality and nonuniversality for this type of computing device is settled.

Keywords: molecular computing, splicing systems, Turing machines, formal languages, universal computations.

1. INTRODUCTION

Starting from [1] a lot of studies were devoted to splicing computations. They constitute various extensions of H systems originating from [2]. In particular, many studies point at their possible universality power ([11], [12], [16]).

Time-varying distributed H systems were recently introduced in [12], [13, 14]. These models introduce *components*, later see the formal definition, which cannot all be used at the same time but one after another, periodically.

This aims at giving an account of real biochemical reactions where the work of enzymes essentially depends on the environment conditions.

In [13], it is proved that 7 different *components* of TVDH systems are enough in order to generate any recursively enumerable language. Recently, see [5], both authors proved that two components are enough to construct a *universal* time-varying distributed H system, *i.e.* a time-varying distributed H system, capable of simulating the computation of any Turing machine. Universality of computation and generating any recursively enumerable language are equivalent properties, but it is *a priori* not necessarily true, that the universality of some time-varying distributed H system with two components entails that time-varying distributed H systems generate all recursively enumerable languages, with only two components. We proved in [6] that this is the case: 2 different components are enough in order to generate any recursively enumerable language.

The question of the status of time-varying distributed H system of degree one is solved now. Recently both authors proved that there is a universal time-varying distributed H system of degree 1 [8].

We consider also G.Paun's original definition of TVDH systems and suggest a slightly different definition of such systems based on the definition of H systems (so-called extended TVDH systems or ETVDH systems). It seems that ETVDH systems are more "natural" than TVDH systems but they have a more complex behaviour, as the strings can pass unchanged from a step to the next one, while also multiple splicings are allowed. So them study is a more complex task than for TVDH systems. For ETVDH systems we proved [6, 7] that these systems with one component generate exactly the set of all regular languages.

Now we prove that ETVDH systems with 2 components can generate any recursively enumerable languages (so, it settles a frontier between decidability and undecidability for extended time-varying distributed H systems in terms of the number of their components: two is the smallest required number of components in order to generate all recursively enumerable sets).

2. BASIC DEFINITIONS

We recall some notions. An *alphabet* V is a finite, non-empty set whose elements are called *letters*. A *word* (over some alphabet V) is a finite (possibly empty) concatenation of letters (from V). The empty concatenations of letters is also called the *empty word* and is denoted by ε. The set of all words over V is denoted by V^*. A *language* (over V) is a set of words (over V).

A *formal grammar* (of *type* 0) G is a tuple $G = (N, T, P, S)$ of an alphabet N of so-called *non-terminal* letters, an alphabet T of so-called *terminal* letters, with $N \cap T = \emptyset$, an *initial letter* S from N, and a finite set P of *rules* of the form $u \to v$ with $u, v \in (N \cup T)^*$. Note that u contains at least one symbol from N. Any rule $u \to v \in P$ is a substitution rule allowing to substitute any occurrence of u in some word w by v.

Formally, we write $w \Rightarrow_G w'$ if there is a rule $u \to v$ in P and words $w_1, w_2 \in (N \cup T)^*$ with $w = w_1 u w_2$ and $w' = w_1 v w_2$. We denote by $\Rightarrow_G^*$ the reflexive and transitive closure of $\Rightarrow$. I.e., $w \Rightarrow_G^* w'$ means that there is an integer n and words $w_1, \cdots, w_n$ with $w = w_1, w' = w_n$ and $w_i \Rightarrow_G w_{i+1}$ for all $i, 1 \leq i < n$.

The sequence $w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n$ is also called a computation (from w_1 to w_n of length $n-1$). A *terminal word* is a word in T^*; all terminal words computable from the initial letter S form the language $L(G)$ generated by G. More formally, $L(G) \stackrel{\text{def}}{=} \{w \in T^*;\, S \Rightarrow^* w\}$.

An *(abstract) molecule* is simply a word over some alphabet. *A splicing rule* (over alphabet V), is a quadruple (u_1, u_2, u_1', u_2') of words $u_1, u_2, u_1', u_2' \in V^*$, which is often written in a two dimensional way as follows: $\dfrac{u_1 \mid u_2}{u_1' \mid u_2'}$.

A splicing rule $r = (u_1, u_2, u_1', u_2')$ is applicable to two molecules m_1, m_2 if there are words $w_1, w_2, w_1', w_2' \in V^*$ with $m_1 = w_1 u_1 u_2 w_2$ and $m_2 = w_1' u_1' u_2' w_2'$, and produces two new molecules $m_1' = w_1 u_1 u_2' w_2'$ and $m_2' = w_1' u_1' u_2 w_2$. In this case, we also write $(m_1, m_2) \vdash_r (m_1', m_2')$.

A pair $h = (V, R)$, where V is an alphabet and R is a finite set of splicing rules, is called a *splicing scheme* or an H *scheme*.

For an H scheme $h = (V, R)$ and a language $L \subseteq V^*$ we define:
$\sigma_h(L) = \sigma_{(V,R)}(L) \stackrel{\text{def}}{=} \{w, w' \in V^* | \exists w_1, w_2 \in L : \exists r \in R : (w_1, w_2) \vdash_r (w, w')\}$.

A *Head-splicing-system* [2], or H *system*, is a construct:

$$H = (h, A) = ((V, R), A),$$

which consists of an alphabet V, a set $A \subseteq V^*$ of initial molecules over V, the *axioms*, and a set $R \subseteq V^* \times V^* \times V^* \times V^*$ of splicing rules. H is called finite if A and R are finite sets.

For any H scheme h and any language $L \in V^*$ we define:

$$\sigma_h^0(L) = L,$$
$$\sigma_h^{i+1}(L) = \sigma_h^i(L) \cup \sigma_h(\sigma_h^i(L)),$$
$$\sigma_h^*(L) = \cup_{i \geq 0} \sigma_h^i(L).$$

The language generated by H system H is defined as $L(H) \stackrel{\text{def}}{=} \sigma_h^*(A)$.

Thus, the language generated by H system H is the set of all molecules that can be generated in H starting with A as initial molecules by iteratively applying splicing rules to copies of the molecules already generated.

A time-varying distributed H *system* (of degree n, $n \geq 1$), (TVDH system) is a construct:

$$D = (V, T, A, R_1, R_2, \ldots, R_n),$$

where V is *an alphabet*, $T \subseteq V$ is *a terminal alphabet*, $A \subseteq V^*$ is a finite set of *axioms*, and *components* R_i are finite sets of splicing rules over V, $1 \leq i \leq n$.

At each moment $k = n \cdot j + i$, for $j \geq 0$, $1 \leq i \leq n$, only component R_i is used for splicing the currently available strings. Specifically, we define
$L_1 = A$,
$L_{k+1} = \sigma_{h_i}(L_k)$, for $i \equiv k (mod\ n)$, $k \geq 1, 1 \leq i \leq n$, $h_i = (V, R_i)$.

Therefore, from a step k to the next step, $k+1$, one passes only the result of splicing the strings in L_k according to the rules in R_i for $i \equiv k (mod\ n)$; the strings in L_k that cannot enter a splicing are removed.

The language generated by D is, by definition:

$$L(D) \stackrel{\text{def}}{=} (\cup_{k \geq 1} L_k) \cap T^*.$$

As for ETVDH systems we extend the definition of H system to components of TVDH system, so the components $R_i, 1 \leq i \leq n$ of TVDH system of degree n will work as corresponding H systems.

Let us define operation $\tilde{\sigma}_h(L)$ as follows:
$\tilde{\sigma}_h(L) = \tilde{\sigma}_h(L' \cup L'') \stackrel{\text{def}}{=} \sigma_h^*(L')$, where
$L' = \{w_1 \in L | \exists w_2 \in L : \exists w, w' \in V^* : \exists r \in R : (w_1, w_2) \vdash_r (w, w') \text{ or } (w_2, w_1) \vdash_r (w, w')\}$, $L'' = L \setminus L'$, and $h = (V, R)$.

(We note that every language $L \subseteq V^*$ for every H scheme h can be split recursively into two subsets L' and L'').

So, $\tilde{\sigma}_h(L)$ works as H system $(h, L') = ((V, R), L')$.

An extended time-varying distributed H *system* (of degree $n,\ n \geq 1$), (ETVDH system) is a construct:

$$E = (V, T, A, R_1, R_2, \ldots, R_n),$$

where V is an alphabet, $T \subseteq V$ is the *terminal alphabet*, $A \subseteq V^*$ is the finite set of *axioms*, and R_i are *components*, i.e. finite sets of splicing rules over $V,\ 1 \leq i \leq n$.

At each moment $k = n \cdot j + i$, for $j \geq 0,\ 1 \leq i \leq n$, only component R_i is used for splicing the currently available strings.

$L_1 = A$,

$L_{k+1} = \tilde{\sigma}_{h_i}(L_k)$, for $i \equiv k (mod\ n),\ k \geq 1, 1 \leq i \leq n, h_i = (V, R_i)$.

The language generated by E is:

$$L(E) = (\cup_{k \geq 1} L_k) \cap T^*.$$

We denote by REG the set of all regular languages, by RE the set of all recursively enumerable languages, by $EVDH_n,\ n \geq 1$, the family of languages generated by extended time-varying distributed H systems of degree at most n, and by $EVDH_*$ the family of all languages of this type.

3. FORMAL DEFINITIONS

We shall consider recursively enumerable sets of natural numbers instead of recursively enumerable languages. There are many well known methods to encode any formal languages into subsets of natural numbers. It will make our proof simpler.

It is a well known fact that from theoretical computer science that for every recursively enumerable set $\mathcal{L}$ of natural numbers, there is a Turing machine $T_{\mathcal{L}}$ which generates $\mathcal{L}$ as follows. Let $f_{\mathcal{L}}$ be a total recursive function enumerating the elements of $\mathcal{L}$, *i.e.* $\mathcal{L} = \{f_{\mathcal{L}}(x)\ ;\ x \in I\!N\}$. It is well known, see [3], chap.XIII, § 68, Theorem XXVIII, how to construct a Turing machine $T_{\mathcal{L}}$ from the definition of $f_{\mathcal{L}}$ which would compute what f does, *i.e.* for every natural number x, machine $T_{\mathcal{L}}$ transforms any initial configuration $q_1 0 1^{x+1}$ into the final configuration $q_0 0 1^{f_{\mathcal{L}}(x)+1} 0 \ldots 0$. And more, machine $T_{\mathcal{L}}$ never visits cells on the left at the cell with 0 of the initial configuration (i.e. machine $T_{\mathcal{L}}$ has a tape that is semiinfinite to the right) and no instruction of machine $T_{\mathcal{L}}$ is stationary, i.e. the head moves on every steps of computations.

Let us assume for a while that we succeeded to simulate machine $T_{\mathcal{L}}$. It is not yet enough to prove the theorem. Indeed, starting from one word, say $q_1 0 1^{x+1}$ for some x, we would obtain an encoding of $f_{\mathcal{L}}(x)$ as $q_0 0 1^{f_{\mathcal{L}}(x)+1}$. But this is not what we need, because this gives us a single word.

It is not difficult to see that starting from machine $T_{\mathcal{L}}$, there is a Turing machine $T'_{\mathcal{L}}$ which computes $0 1^{x+2} q_0 0 1^{f_{\mathcal{L}}(x)+1} 0 \ldots 0$, starting from $q_1 0 1^{x+1}$. As we shall see, it will be possible for us to device an extended time-varying distributed H system Γ which, arriving to that stage of the computation, will split the obtained word into two parts: $0 1^{f_{\mathcal{L}}(x)+1}$ as a *generated* word and $q_1 0 1^{x+2}$ as a new starting configuration. This will allow us to obtain exactly $\mathcal{L}$ as $\mathcal{L}(\Gamma)$.

In what follows, in order to facilitate the reading of the formulas, we shall write $f(x)$ instead of $f_{\mathcal{L}}(x)$ when it will be clear from the context that we mean that latter function.

Definition of Γ

And so, we assume that $\{0, 1\}$ is the tape alphabet of Turing machine $T'_{\mathcal{L}}$ and let us number its states by $q_0, q_1, \ldots, q_n$, where q_1 is the initial state and q_0 the final one. From now on, we shall use latin small letters to denote tape symbols, greek letters for words on the alphabet of the tape symbols, bold latin small letters for states of the Turing machine.

Theorem. $EVDH_2 = EVDH_* = RE.$

Define ETVDH system Γ as follows:

Its terminal alphabet is $T = \{0, 1\}$, the tape alphabet of $T'_{\mathcal{L}}$. Let us introduce variables for $q_0, q_1, \ldots, q_n$, namely $\mathbf{a}$ and $\mathbf{b}$ with that additional assumption that $\mathbf{b}$ ranges over all q_i's but $\mathbf{a}$ ranges over all of them except q_0. Moreover, we shall introduce one copy of the set of q_i's, namely $\{q'_1, \ldots, q'_n\}$. Accordingly, if $\mathbf{a}$ denotes one of the q_i's, say q_j, then $\mathbf{a}'$ denotes q'_j.

Denoting by $\mathcal{I}$ the set of instructions of $T'_{\mathcal{L}}$, the expressions $\mathbf{a},\ x \in \mathcal{I}$ as well as $\mathbf{a},\ x, \mathbf{b},\ y \in \mathcal{I}$ will be used for indicating an instruction by its constituting elements. In particular, we will use quantifiers with such expressions. Moreover, we shall specialise $\mathcal{I}$ by denoting by $\mathcal{I}_R$ the set of all instructions with a move to right, by $\mathcal{I}_L$ the set of all instructions with a move to left.

Let also $x, y, z \in \{0, 1\}$.

Define alphabet V by setting

$V = \{0, 1\} \cup \{X, Y, Y', \mathbf{b}, \mathbf{a}', t_{e1}, t_{e2}, t_R, t_L, t_{q0}, t_{r1}, t_{r2}, t_{l1}, t_{l2}, R, L, Q, Z, \rightarrow, \leftarrow\}$, with $\mathbf{b}, \mathbf{a}'$ taking all their possible values.

Axioms A are given by

$A = \{Xq_1 01Y,\ t_{e1}\mathbf{a}'0Y,\ t_{e2}\mathbf{a}0Y,\ Ry\mathbf{b}t_R,\ L\mathbf{b}zyt_L,\ RQ \rightarrow t_{q0},\ t_{r1}1 \rightarrow t_{r2},\ Z \leftarrow 1,\ t_{l1} \leftarrow 1t_{l2},\ 0Y',\ Xq_1 0Z\ \}$, with $\mathbf{b}, \mathbf{a}'$ taking all their possible values.

Component R_1.

1.1 : $\begin{array}{r|l} s & \mathbf{a}Y \\ \hline t_{e1} & \mathbf{a}'0Y \end{array}$ $\quad \forall s \in \{0,1,X\}; \forall \mathbf{a};$

1.2 : $\begin{array}{r|l} s & \mathbf{a}\,xpr \\ \hline R & y\,\mathbf{b}\,t_R \end{array}$ $\quad \forall\, s \in \{0,1,X\};$

$\forall p \in \{0,1\};$
$\forall r \in \{0,1,Y\};$
$\forall\, \mathbf{a}, x, y, \mathbf{b} \in \mathcal{I}_R;$

1.3 : $\begin{array}{r|l} s & z\,\mathbf{a}\,xpw \\ \hline L & \mathbf{b}\,z\,yt_L \end{array}$ $\quad \forall\, s \in \{0,1,X\}; \forall\, z;$

$\forall\, p \in \{0,1,\};$
$\forall\, w \in \{0,1,Y\};$
$\forall\, \mathbf{a}, x, y, \mathbf{b} \in \mathcal{I}_L;$

1.3′ : $\begin{array}{r|l} s & z\,\mathbf{a}\,xY \\ \hline L & \mathbf{b}\,z\,yt_L \end{array}$ $\quad \forall\, s \in \{0,1,X\}; \forall\, z;$

$\forall\, \mathbf{a}, x, y, \mathbf{b} \in \mathcal{I}_L;$

1.4 : $\begin{array}{r|l} 1 & q_0\,0\,1p \\ \hline R & Q \rightarrow t_{q0} \end{array}$ $\quad \forall\, p \in \{0,1,Y\};$

1.5 : $\begin{array}{r|l} s & \rightarrow 1 \\ \hline t_{r1} & 1 \rightarrow t_{r2} \end{array}$ $\quad \forall s \in \{1,Q\};$

1.6 : $\begin{array}{r|l} s & 1 \rightarrow p \\ \hline Z & \leftarrow 1 \end{array}$ $\quad \forall s \in \{1,Q\}; \forall p \in \{0,Y\};$

1.7 : $\begin{array}{r|l} & 1 \leftarrow t_{l2} \\ \hline t_{l1} & \leftarrow 1p \end{array}$ $\quad \forall p \in \{1,Y\};$

1.8 : $\begin{array}{r|l} & Q \leftarrow Y' \\ \hline & \leftarrow Y \end{array}$

Component R_2.

2.1 : $\begin{array}{r|l} s & \mathbf{a}'\,0Y \\ \hline t_{e2} & \mathbf{a}\,0Y \end{array}$ $\quad \forall s \in \{0,1,X\}; \forall \mathbf{a};$

2.2 : $\begin{array}{r|l} s\,y\,\mathbf{b} & t_R \\ \hline R\,\mathbf{a}\,x & \end{array}$ $\quad \forall\, \mathbf{a}, x, y, \mathbf{b} \in \mathcal{I}_R;$

$\forall\, s \in \{0,1,X\};$

2.3 : $\begin{array}{r|l} s\,\mathbf{b}\,z\,y & t_L \\ \hline L\,z\,\mathbf{a}\,x & \end{array}$ $\quad \forall\, s \in \{0,1,X\}; \forall\, z;$

$\forall\, \mathbf{a}, x, y, \mathbf{b} \in \mathcal{I}_L;$

2.4 : $\begin{array}{r|l} 1\,Q \rightarrow & t_{q0} \\ \hline R\,q_0 0 & \end{array}$

2.5 : $\begin{array}{r|l} s1 \rightarrow & t_{r2} \\ \hline t_{r1} \rightarrow 1 & \end{array}$ $\quad \forall\, s \in \{1,Q\};$

2.6 : $\begin{array}{r|l} 1 \leftarrow & p \\ \hline t_{l1} \leftarrow 1 & t_{l2} \end{array}$ $\quad \forall p \in \{1,Y\};$

2.7 : $\begin{array}{r|l} Q \leftarrow & 1 \\ \hline 0 & Y' \end{array}$

2.8 : $\begin{array}{r|l} X\,0 \leftarrow & 1 \\ \hline X\,q_1\,0 & Z \end{array}$

Both components R_1 and R_2 also contain the following rules:

$\begin{array}{c|l} \alpha & \\ \hline \alpha & \end{array}$ $\quad$ for each axiom $\alpha \in A$, except $Xq_1 01Y$.

As it can be seen from the computations, axioms must always be present for making it possible to apply most of the rules.

4. CHECKING THE SIMULATION

4.1 One step in a Turing computation

The first thing we have now to check is that starting from a word representing the current configuration of machine $T'_{\mathcal{L}}$, the application of the rules of Γ leads to a word representing the next configuration. In other words, given any instruction of the program of $T'_{\mathcal{L}}$, we have to show that there are rules in R_1 and R_2 and axioms allowing to mimic the application of that instruction to a suitable configuration.

There are several cases according to the move of the instruction and of the position of the scanned cell in the configuration: according to whether the move is to right or to left, according to whether the scanned cell is an interior one or a cell being at the right end of the configuration.

Case of an interior cell

Let ω_ℓ, ω_r be finite words of alphabet $\{0,1\}$ and $\omega_\ell\,\mathbf{a}\,x\,\omega_r$ be the current configuration of machine $T'_{\mathcal{L}}$ with $\omega_\ell = \nu_\ell\, z$. Assume that Γ has obtained word $X\,\nu_\ell\, z\,\mathbf{a}\,x\,\omega_r\,Y$ at a time, say t, when the rule to be applied must be taken from R_1. Assume that instruction $\boxed{\mathbf{a}\,x\,L\,y\,\mathbf{b}}$ applies to the current configuration. Possibly after several steps, Γ should produce word $X\,\nu_\ell\,\mathbf{b}\,z\,y\,\omega_r\,Y$. The reader can get convinced that in two steps Γ produces the expected word and no undesirable one: indeed, rule 1.3 (1.3′) applies as far as the axiom $L\,\mathbf{b}\,z\,y\ t_L$ is present, leading to words $X\,\nu_\ell\,\mathbf{b}\,z\,y\;t_L$ and $L\,z\,\mathbf{a}\,x\,w_r\,Y$ that can enter in a rule of R_2 and, therefore, are kept for the further processing. Now, rule 2.3 straightforward applies to these words and provides $X\,\nu_\ell\,\mathbf{b}\,z\,y\,\omega_r\,Y$ and $L\,z\,\mathbf{a}\,x\;t_L$. Observe now that $L\,z\,\mathbf{a}\,x\;t_L$ cannot enter a rule of R_1 and so, it is ruled out. On another hand, word $X\,\nu_\ell\,\mathbf{b}\,z\,y\,\omega_r\,Y$ is the representation of the next configuration and, as it is now the turn to apply a rule namely from R_1, we see that the previous conditions are again observed, which proves the simulation of one step of Turing computation by Γ, at least when a instruction with a move to left is applied.

The reader can now check that in the case of an instruction $\mathbf{a}\,x\,R\,y\,\mathbf{b}$ with a move to right, rules 1.2 and 2.2 successively apply, providing the expected next configuration, namely $X\,\omega_\ell\,y\,\mathbf{b}\,\omega_r\,Y$.

Case of the right end of the tape

We have now to check what happens when the head has to go to the right of the right end of the configuration. As in our representation of a configuration the ends are realised by special markers, X for the left end, Y for the right one, such cases are easily characterised: $X\,\mathbf{a}\,x\,\nu\,Y$ for the left end and $X\,\omega\,\mathbf{a}\,Y$ for the right one.

In the case of the left end, the case is dealt with by the already seen rules 1.2, 2.2 if the instruction determined by $\mathbf{a}\,x$ has a move to right.

In the case of the right end there is a uniform way for solving that case: it is enough to transform representation $X\,\omega\,\mathbf{a}\,Y$ into the following one: $X\,\omega\,\mathbf{a}\,0\,Y$, which can be dealt with as the case of an interior cell. It is easy to check that rule 1.1 and then rule 2.1 allow to obtain the just described transformation.

4.2 Checking how to resume the computation on 01^{x+2}

We now arrive at an important point of our simulation. From what we already seen, we now prove that Γ transforms $X\,q_1\,0\,1^{x+1}\,Y$ into $X\,0\,1^{x+2}\,q_0\,0\,1^{f(x)+1}\,0\,\ldots\,0\,Y$. Let us write $\omega_m = 01^{f(m)+1}$, where f enumerates $\mathcal{L}$, and inductively define $\mathcal{L}_m$ by

$$\mathcal{L}_0 = \{\omega_0\}$$
$$\mathcal{L}_{k+1} = \mathcal{L}_k \cup \{\omega_{k+1}\}.$$

It will be enough to check that if at some time t_k we have obtained $\mathcal{L}_k$ and $X\,q_1\,0\,1^{k+2}\,Y$, then at a later time t_{k+1} we shall obtain $\mathcal{L}_{k+1}$ and $X\,q_1\,0\,1^{k+3}\,Y$. And so, starting from $X\,0\,1^{k+3}\,q_0\,0\,1^{f(k+1)+1}\,0\,\ldots\,0\,Y$, we leave a marker, Q at the place of $q_0 0$ and, at the same time, send a signal searching the first 0 following the 0 scanned by the head at that time. When the wanted 0 is found, it cuts off the superfluous 0's and Y and the signal goes back to the marker. When it reaches it, it splits the word in two parts: the right one is the expected ω_{k+1}; in the left one, the marker is changed into Y and the signal goes on to left until it meets 0. At that moment it transforms what is on the left of that $X0$ into $Xq_1 0$, and the new expected starting configuration is reached.

So, we explain that in details. Starting from word $\boxed{X\,0\,1^{k+3}\,q_0\,0\,1^{f(k+1)+1}\,0\,\ldots\,0\,Y}$, the application of rules 1.4 and 2.4 allows to put marker Q on the place of q_0, to erase $q_0 0$ and to insert in front of $1^{f(k+1)+1}$ signal $\rightarrow$: indeed, the application of rule 1.4 yields $X\,0\,1^{k+3}\,Q \rightarrow t_{q0}$ and $Rq_0\,0\,1^{f(k+1)+1}\,0\,\ldots\,0\,Y$. Both words are kept as far as they enter rule 2.4 which in its turn yields word $\boxed{X\,0\,1^{k+3}\,Q \rightarrow 1^{f(k+1)+1}\,0\,\ldots\,0\,Y}$, as well as $R\,q_0\,0t_{q0}$ which enter no rule of R_1 and, consequently, is ruled out. Then, rules 1.5 and 2.5 allow to cross over the block of 1's constituting the expected result ω_{k+1}. Indeed, assuming that the computation starts from word $X\,0\,1^{k+3}\,Q\,1^u \rightarrow 11^v\,0\,\ldots\,0\,Y$, rule 1.5 yields $X\,0\,1^{k+3}\,Q\,1^u 1 \rightarrow t_{r2}$ and $t_{r1} \rightarrow 1\,1^v\,0\,\ldots\,0\,Y$ which enter rule 2.5 yielding $X\,0\,1^{k+3}\,Q\,1^u 1 \rightarrow\; 1^v\,0\,\ldots\,0\,Y$ and $t_{r1} \rightarrow 1\;t_{r2}$ which is ruled out, as far as it enters no rule of R_1. And so, by induction on the number of 1's to be crossed over, we see that the appropriate iteration of rules 1.5 and 2.5 lead to word $\boxed{X\,0\,1^{k+3}\,Q\,1^{f(k+1)+1} \rightarrow 0\,\ldots\,0\,Y}$. But now, rule 1.6 allows to throw away the superfluous substring $0\,\ldots\,0\,Y$ and to reverse the signal, leading to word $\boxed{X\,0\,1^{k+3}\,Q\,1^{f(k+1)} \leftarrow 1}$. Notice that now, the starting set of rules becomes R_2. Indeed, repeated applications of rule 2.6 and 1.7 allow to arrive to word $\boxed{X\,0\,1^{k+3}\,Q \leftarrow 1^{f(k+1)+1}}$. Indeed, starting from $X\,0\,1^{k+3}\,Q\,1^u\,1 \leftarrow\; 1^v$, rule 2.6 yields $X\,0\,1^{k+3}\,Q\,1^u\,1 \leftarrow t_{r2}$ and $t_{r1} \leftarrow 1\,1^v$, both word entering rule 1.7, yielding $X\,0\,1^{k+3}\,Q\,1^u \leftarrow 1\,1^v$ and $t_{r1}1 \leftarrow t_{r2}$, the latter being ruled out as far as it enters no rule of R_2.

Now, whatever is the value of $f(k+1)$, rule 2.7 and 1.8 apply and allow to separate the result as a word in the terminal alphabet, namely ω_{k+1} from the other part, transformed into word $\boxed{X\,0\,1^{k+3} \leftarrow Y}$. Indeed, applied to word $X\,0\,1^{k+3}\,Q \leftarrow 1^{f(k+1)+1}$ rule 2.7 yields $X\,0\,1^{k+3}\,Q \leftarrow Y'$ and $\boxed{01^{f(k+1)+1}}$.

The latter word is the expected ω_{k+1} and the computation goes on with the former word, and it enters rule 1.8 allowing to obtain $\boxed{X\,0\,1^{k+3}\,\leftarrow\,Y}$ and also $Q \leftarrow Y'$ which is ruled out, as entering no rule of R_2.

We are now reaching the last stage of the computation leading to $\boxed{X\,q_1\,0\,1^{k+3}\,Y}$. Starting from word $X\,0\,1^{k+3}\,\leftarrow\,Y$, rules 2.6 and 1.7 again apply, allowing after an appropriate iteration to reach word $X\,0\,\leftarrow\,1^{k+3}\,Y$, to which rule 2.8 applies, yielding the needed configuration $\boxed{X\,q_1\,0\,1^{k+3}\,Y}$ together with $X\,0 \leftarrow Z$ which is ruled out as entering no rule of R_1.

And so we proved our claim about turning from $\mathcal{L}_k$ to $\mathcal{L}_{k+1}$.

ACKNOWLEDGEMENT

The authors acknowledge the very helpful contribution of *INTAS project* 97-1259 and *NATO project* PST.CLG.976912 for enhancing their cooperation, giving the best conditions for producing the present result. For the same reasons they also acknowledge the help of the University of Metz (France), INRIA Lorraine and the French Ministry of Education and Research.

REFERENCES

[1] E. Csuhaj-Varjù, L. Kari, G. Păun, "Test Tube distributed system based on splicing", Computer and AI., Vol. 2–3, 1996, pp. 211–232.

[2] T. Head, "Formal language theory and DNA: an analysis of the generative capacity of recombinant behaviours", Bulletin of Mathematical Biology, Vol. 49, 1987, pp. 737–759.

[3] S. Kleene, Introduction to Metamathematics: Van Nostrand Comp. Inc., New-York, 1952.

[4] M. Margenstern, "Frontier between decidability and undecidability: a survey", TCS, Vol. 231-2, 2000, pp. 217–251.

[5] M. Margenstern, Yu. Rogozhin, "A universal time-varying distributed H-system of degree 2", Biosystems, Vol. 52, 1999, pp. 73–80.

[6] M. Margenstern, Yu. Rogozhin, "Generating all recursively enumerable languages with a time-varying distributed H-system of degree 2", Publications du GIFM, no. 99-102, I.U.T., Metz (France), 1999, ISBN 2-9511539-5-3.

[7] M. Margenstern, Yu. Rogozhin, "About Time-Varying Distributed H systems", DNA Computing, 6th International Workshop on DNA-Based Computers, DNA 2000, Leiden, The Netherlands, June 13-17, 2000, LNCS, Vol. 2054, pp. 53–62.

[8] M. Margenstern, Yu.Rogozhin, "A universal time-varying distributed H system of degree 1", Accepted for DNA7 conference, 2001.

[9] A. Păun, "On Time-Varying H Systems", Bulletin of EATCS, Vol. 67, 1999, pp. 157–164.

[10] G. Păun, "Regular extended H systems are computationally universal", Journal of Automata, Languages and Combinatorics, Vol. 1, No.1, 1996, pp. 27–36.

[11] G. Păun, G. Rozenberg, A. Salomaa, "Computing by splicing", TCS, Vol. 168-2, 1996, pp. 321–336.

[12] G. Păun, "DNA computing: distributed splicing systems", In *Structures in Logic and Computer Science. A Selection of Essays in honor of A. Ehrenfeucht*, LNCS, Vol. 1261, 1997, pp. 353–370.

[13] G. Păun, "DNA Computing Based on Splicing: Universality Results", TCS, Vol. 231-2, 2000, pp. 275–296.

[14] G. Păun, G. Rozenberg, A. Salomaa, DNA Computing: New Computing Paradigms: Springer, Heidelberg, 1998.

[15] D. Pixton, "Regularity of splicing languages", Discrete Applied Mathematics, Vol. 69, 1996, pp. 101–124.

[16] L. Priese, Yu. Rogozhin, M. Margenstern, "Finite H-Systems with 3 Test Tubes are not Predictable", In Proceedings of Pacific Symposium on Biocomputing, Kapalua, Maui, January 1998 (R.B.Altman, A.K.Dunker, L.Hunter, T.E.Klein, eds): World Sci. Publ., Singapure, 1998, pp. 545–556.

[17] Yu. Rogozhin, "Small universal Turing machines", TCS, Vol. 168-2, 1996, pp. 215–240.

[18] S. Verlan, "On extended time-varying distributed H systems", In Proceedings, Sixth International Meeting on DNA Based Computers, June 13-17, 2000, Leiden, The Netherlands, 2000, p. 281.