DNA Computing Based on Splicing: Universality Results

Erzsébet CSUHAJ-VARJÚ[1]
Computer and Automation Institute, Hungarian Academy of Sciences
Kende u. 13-17, 1111-Budapest, Hungary

Rudolf FREUND[2]
Institute for Computer Languages, Technical University of Vienna
Resselgasse 3, 1040 Wien, Austria

Lila KARI[3]
Department of Mathematics and Computer Science, University of Western Ontario
London, Ontario, N6A 5B7, Canada

Gheorghe PĂUN[4]
Institute of Mathematics of the Romanian Academy
PO Box 1 - 764, 70700 Bucureşti, Romania

Abstract. The paper extends some of the most recently obtained results on the computational universality of extended H systems (with regular sets of rules respectively with finite sets of rules used with simple additional mechanisms) and shows the possibility to obtain universal systems based on these extended H systems, i. e. the theoretical possibility to design programmable universal DNA computers based on the splicing operation. The additional mechanisms considered here are: multisets (counting the numbers of copies of each available string), checking the presence/absence of certain symbols in the spliced strings, and organizing the work of the system in a distributed way (like in a parallel communicating grammar system). In the case of multisets we also consider the way of simulating a Turing machine (computing a partial recursive function) by an equivalent H system (computing the same function), in the other cases we consider the interpretation of algorithms as language generating devices, hence the aim is to reach the power of Chomsky type-0 grammars, the standard model for representing algorithms being equivalent with Turing machines taken as language generators.

Keywords: DNA splicing, grammar systems, H systems, Turing machines, universal computing

[1]Research supported by the Hungarian Scientific Research Fund "OTKA" T 017105.

[2]All correspondence to this author.

[3]Research supported by grants OGP0007877 and OGP0000243 of the National Science and Engineering Research Council of Canada.

[4]Research supported by the Academy of Finland, project 11281.

1 Introduction

One of the recently introduced paradigms which promises to have a tremendous influence on the (theoretical and practical) progress of computer science is *DNA computing.* The main step in making it so interesting was the announcement of solving (a small instance of) the Hamiltonian path problem in a test tube just by handling DNA sequences [1], but the event has somewhat been prepared by the intensive efforts aiming to draw mappings of the human genome, and by related developments not only in biology, but also in computer science - genetic algorithms [8], neural computation [15], etc.

Adleman's approach raised some exciting problems emerging in the new framework, concerning the new kind of inputs, the questions about what is a *computation*, what is an *algorithm* designed to *compute,* etc.; his "algorithm" is based on properties of the so-called Watson-Crick complements and in some sense able to simulate a Post Correspondence Problem (PCP): given a set of DNA single stranded strings, $x_1, x_2, ..., x_n$, Watson-Crick complements $y_1, y_2, ..., y_m$ are considered which are able to match specified suffixes and prefixes of the strings $x_1, ..., x_n$ (e. g., if $x_1 = x_1' x_1''$, $x_2 = x_2' x_2''$, a string y_i will be the complement of $x_1'' x_2'$, thus matching it and forcing x_1, x_2 to be bound, hence to be concatenated). In this way, all possible desired concatenations of strings $x_1, ..., x_n$ can be produced, paired with strings $y_1, ..., y_m$, which is similar to finding the solution of a PCP for the two lists of strings. As PCP is "computationally universal", every recursively enumerable language is the morphic image of an "equality set" (the set of all solutions of a Post correspondence problem). Of course, the remarks above are only a metaphorical "proof" of the fact that Adleman's way to compute using DNA is powerful. Actually the universality of this way for computing still seems to be not yet settled theoretically in a satisfactory way.

"Universal systems require the ability to store and retrieve information, and DNA is certainly up to the task if one could design appropriate molecular mechanisms to interpret and update the information in DNA. This ultimate goal remains elusive, but once solved, it will revolutionize the way we think about both computer science and molecular biology. A great hope is that as we begin to understand how biological systems compute, we will identify a naturally occurring universal computational system," [12].

Another trend in DNA computing is based on the recombinant behaviour of DNA (double stranded) sequences under the influence of restriction enzymes and lygases.

This approach starts with [13], where the operation of splicing has been introduced as a model for this phenomenon.

As formalized in [13], given two strings of symbols x and y, the splicing operation consists of cutting x and y at certain positions (determined by the splicing rule) and pasting the resulting prefix of x with the suffix of y, respectively pasting the resulting prefix of y with the suffix of x. Formally, if the applied splicing rule is $(u_1, u_2; u_3, u_4)$, then the results of splicing x and y are z and w if and only if $x = x_1 u_1 u_2 x_2$, $y = y_1 u_3 u_4 y_2$ and $z = x_1 u_1 u_4 y_2$, $w = y_1 u_3 u_2 x_2$; all the strings $u_1, u_2, u_3, u_4, x_1, x_2, y_1, y_2$ are strings over a given alphabet V. (In the case of real DNA sequences, the alphabet consists of four letters, i. e. *a, c, g, t*, representing the four bases adenine, cytosine, guanine and thymine, the cutting is realized by restriction enzymes, and the concatenation by lygases. Pairs of the form

$(u_1, u_2), (u_3, u_4)$ as above are intended to specify the places where cutting and pasting operations are possible.) In [14], [17], and [19] a more general definition for the result of a splicing operation is considered, i. e. only the string $z = x_1u_1u_4y_2$ is taken as the result of splicing $x = x_1u_1u_2x_2$ and $y = y_1u_3u_4y_2$, but we will be able to prove the results of this paper within the framework of the restricted definition given above. (From a practical point of view, it is important to consider constructions which are as close as possible to the test tube reality.)

The splicing operation can be used as a basic tool for building a generative mechanism, called a *splicing system* or *H system*, in the following way. Given a set of strings (axioms) and a set of splicing rules, the generated language will consist of the strings obtained in an iterative way by applying the rules to the axioms and/or to the strings obtained in preceeding splicing steps. If we add the restriction that only strings over a designed subset of the alphabet are accepted in the language, we obtain an extended H system in the sense of [19].

The power of H systems, extended or not, turned out to be very large, and their behaviour to be very interesting. For instance, one of the important results in this area states that H systems with finite sets of axioms and finite sets of splicing rules can generate only regular languages. The long proof in [6] is based on dominoes techniques, a shorter one is provided in [21], in terms of formal language theory. However, if we use a regular set of splicing rules of a very particular type, surprisingly enough a maximal increase of the power is obtained: such H systems characterize the family of recursively enumerable languages. We recall here the construction in the proof of this result in [18], both for the sake of completeness, as well as because we start from it in order to obtain improved versions of this specific result. More precisely, working with *regular sets* of splicing rules is natural from a mathematical point of view, but unrealistic from a practical point of view (merely for the reason that the test tubes are finite...) *How to obtain H systems with both the set of axioms and the set of splicing rules being finite, but still being able to equalize the full power of Turing machines?* In view of the results in [6], [21], we have to pay the reduction of the sets of rules from being regular to being finite, and the way to do this is well-known in formal language theory [7]: we regulate the use of the splicing rules by suitable control mechanisms. This idea has been explored in [11], where computationally universal classes of finite H systems are obtained by associating *permitting* or *forbidding* context conditions to the splicing rules: a splicing rule can be used only when a certain favourizing symbol (a "catalyst", a "promotor") is present in the strings to be spliced, respectively when no "inhibitor" from a specified finite set of symbols is present.

Another recently developed branch of formal language theory aiming, among others, to increase the power of grammars, is the theory of *grammar systems* (see [4]: the main idea is to put several grammars to work together, according to a specified cooperation protocol, in order to generate a common language). The H systems mentioned above with permitting/forbidding contexts can be viewed as "cooperating distributed test tube systems", similar to the cooperating distributed grammar systems with dynamic start conditions (checking the presence/absence of the context symbols) in the sense of [4].

A new idea is to consider a "parallel communicating" architecture as introduced in [20]: several test tubes work in parallel (splicing their contents), communicating by redistributing their contents in a way similar to the operation of *separating* the contents of a tube [2], [16]: the contents of a tube is redistributed to other tubes according to certain specified "separation conditions". The result is, on the one hand expected - the increase in power again leads to equalizing the power of Turing machines - on the other hand quite impressing - systems with *seven* components are sufficient, the hierarchy on the number of used test tubes collapses. (We do not know whether seven tubes are really necessary or whether only our proof requests this "magic" number of tubes.)

Another powerful idea able to increase the power of H systems with finite sets of axioms and finite sets of splicing rules is to count the number of copies of each used string. This has already been used in [9] and also appears in [11], where it is proved that extended H systems working in the multiset style are able to characterize the recursively enumerable languages. Here we extend this theorem and its proof in a way closer to the computing framework, i. e. we consider Turing machines as devices for computing partial recursive functions: starting from a tape $Z_0wq_0B^\omega$ the Turing machine halts with $Z_0f(w)q_fB^\omega$ if and only if w is in the domain of f, where f is the function to be computed, Z_0 is the left marker, B is the blank symbol, q_0 is the initial state, and q_f is the final state (moreover, no further action is possible after having introduced q_f).

The work of such a Turing machine can be simulated in a natural way by an H system, for which the string originally written on the Turing machine tape is supposed to appear in only one copy, whereas all the other strings are available in arbitrary many copies. (Moreover, in this way the obtained H system need not to be extended, due to the working styles of the Turing machines we consider.) Hence, we here find interesting details important from practical points of view.

¿From the proof of all the previously mentioned results and from the existence of universal Turing machines [22] and correspondingly, universal Chomsky type-0 grammars, we obtain ways to construct universal H systems. This can be interpreted as a proof for the (theoretical) possibility to construct universal and programmable DNA computers based on the splicing operation.

2 Definitions for H systems

We use the following notations: V^* is the free monoid generated by the alphabet V, λ is the empty string, $V^+ = V^* - \{\lambda\}$, $|x|$ is the length of $x \in V^*$, 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 regulated rewriting to [7], and for grammar systems to [4].

Definition 1. An *extended H system* is a quadruple

$$\gamma = (V, T, A, R),$$

where V is an alphabet, $T \subseteq V$, $A \subseteq V^*$, and $R \subseteq V^*\#V^*\$V^*\#V^*$; $\#, \$$ are special symbols not in V. (V is the *alphabet* of γ, T is the *terminal* alphabet, A is the set of *axioms*, and R is the set of *splicing rules*; the symbols in T are called *terminals* and those in $V - T$ are called *nonterminals*.)

For $x, y, z, w \in V^*$ and $r = u_1\#u_2\$u_3\#u_4$ in R we define

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

□

The strings x, y are called the *terms* of the splicing; u_1u_2 and u_3u_4 are called the *sites* of the splicing.

Definition 2. For an H system $\gamma = (V, T, A, R)$ and for any language $L \subseteq V^*$, we write

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

and we define

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

where

$$\begin{array}{lcl} \sigma^0(L) & = & L, \\ \sigma^{i+1}(L) & = & \sigma^i(L) \cup \sigma(\sigma^i(L)) \quad \text{for } i \geq 0. \end{array}$$

The *language generated* by the H system γ is defined by

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

Then, for two families of languages, F_1, F_2, we denote

$$EH(F_1, F_2) = \{L(\gamma) \mid \gamma = (V, T, A, R), A \in F_1, R \in F_2\}.$$

(An H system $\gamma = (V, T, A, R)$ with $A \in F_1, R \in F_2$, is said to be *of type* F_1, F_2.)

□

In the definition above, the rule to be used and the positions where the terms of the splicing shall be cut are not prescribed, they are chosen in a nondeterministic way. Moreover, after splicing two strings x, y and obtaining two strings z and w, we may use again x or y (they are not "consumed" by splicing) as a term of a splicing, possibly the second one being z or w, but also the new strings are supposed to appear in infinitely many copies. Probably more realistic is the assumption that at least part of the strings are available in a limited number of copies. This leads to consider *multisets*, i. e. sets with multiplicities associated to their elements.

In the style of [10], a multiset over V^* is a function $M : V^* \longrightarrow \mathbf{N} \cup \{\infty\}$; $M(x)$ is the number of copies of $x \in V^*$ in the multiset M. All the multisets we consider are supposed to be defined by recursive mappings M. The set $\{w \in V^* \mid M(w) > 0\}$ is called the support of M and it is denoted by *supp*(M). A usual set $S \subseteq V^*$ is interpreted as the multiset defined by $S(x) = 1$ for $x \in S$, and $S(x) = 0$ for $x \notin S$.

For two multisets M_1, M_2 we define their *union* by $(M_1 \cup M_2)(x) = M_1(x) + M_2(x)$, and their *difference* by $(M_1 - M_2)(x) = M_1(x) - M_2(x)$, $x \in V^*$, provided $M_1(x) \geq M_2(x)$ for all $x \in V^*$. Usually, a multiset with finite support, M, is presented as a set of pairs $(x, M(x))$, for $x \in$ *supp*(M).

Definition 3. An *extended mH system* is a quadruple $\gamma = (V, T, A, R)$, where V, T, R are as in an extended H system (Definition 1) and A is a multiset over V^*.

For such an mH system and two multisets M_1, M_2 over V^* we define

$$\begin{array}{lll} M_1 \Longrightarrow_\gamma M_2 & \text{iff} & \text{there are } x, y, z, w \in V^* \text{ such that} \\ & \text{(i)} & M_1(x) \geq 1,\ M_1(y) \geq 1, \text{ and if } x = y, \text{ then } M_1(x) \geq 2, \\ & \text{(ii)} & x = x_1u_1u_2x_2,\ y = y_1u_3u_4y_2, \\ & & z = x_1u_1u_4y_2,\ w = y_1u_3u_2x_2, \\ & & \text{for } x_1, x_2, y_1, y_2 \in V^*,\ u_1\#u_2\$u_3\#u_4 \in R, \\ & \text{(iii)} & M_2 = (((M_1 - \{(x,1)\}) - \{(y,1)\}) \cup \{(z,1)\}) \cup \{(w,1)\} \end{array}$$

(At point (iii) we have operations with multisets.)

The *language generated* by an extended mH system γ is

$$L(\gamma) = \{w \in T^* \mid w \in \mathit{supp}(M) \text{ for some } M \text{ such that } A \Longrightarrow_\gamma^* M\},$$

where $\Longrightarrow_\gamma^*$ is the reflexive and transitive closure of $\Longrightarrow_\gamma$.

For two families of languages, F_1, F_2, we denote

$$EH(mF_1, F_2) = \{L(\gamma) \mid \gamma = (V, T, A, R) \text{ is an mH system with } \mathit{supp}(A) \in F_1,\ R \in F_2\}.$$

□

An H system as in Definition 1 can be interpreted as an mH system working with multisets of the form $M(x) = \infty$ for all x such that $M(x) \neq 0$. Such multisets are called ω-multisets and the corresponding H systems are called ωH systems. The corresponding families will be also denoted by $EH(\omega F_1, F_2)$.

An H system $\gamma = (V, T, A, R)$ with $V = T$ is called *non-extended*; the families of languages generated by such systems, corresponding to $EH(mF_1, F_2)$ and $EH(\omega F_1, F_2)$ are denoted by $H(mF_1, F_2)$ and $H(\omega F_1, F_2)$, respectively.

3 Computational completeness of H systems with multisets

In this section we elaborate how the use of multisets in connection with extended H systems allows us to achieve the generative power of type-0 Chomsky grammars; moreover we show how (even non-extended) H systems with multisets together with suitable strategies for selecting the final strings can simulate arbitrary computations with Turing machines.

Theorem 1. *$EH(mFIN, FIN) = EH(mF_1, F_2) = RE$, for all families F_1, F_2 such that $FIN \subseteq F_1 \subseteq RE$, $FIN \subseteq F_2 \subseteq RE$.*

Proof. We will only prove the inclusion $RE \subseteq EH(mFIN, FIN)$, because the other inclusions are obvious.

Consider a type-0 Chomsky grammar $G = (N, T, S, P)$, with the rules in P of the form $u \to v$ with $1 \leq |u| \leq 2$, $0 \leq |v| \leq 2$, $u \neq v$ (for instance, we can take G in Kuroda normal form). Also assume that the rules in P are labelled in an one-to-one manner with elements of a set L; we write $r : u \rightarrow v$, for r being the label of $u \rightarrow v$. By U we denote the set $N \cup T$ and we construct the extended mH system $\gamma = (V, T, A, R)$, where

$$V = N \cup T \cup \{X_1, X_2, Y, Z_1, Z_2\} \cup \{(r), [r] \mid r \in L\},$$

the multiset A contains the string $w_0 = X_1^2 Y S X_2^2$, with the multiplicity $A(w_0) = 1$, and the following strings with infinite multiplicity:

$$\begin{array}{lcll} w_r & = & (r)\, v\, [r]\,, & \text{for } r : u \rightarrow v \in P, \\ w_\alpha & = & Z_1 \alpha Y Z_2, & \text{for } \alpha \in U, \\ w'_\alpha & = & Z_1 Y \alpha Z_2, & \text{for } \alpha \in U, \\ w_t & = & YY. & \end{array}$$

The set R contains the following splicing rules:

$$\begin{array}{lll}
1. & \delta_1\delta_2 Y u \# \beta_1\beta_2 \$ \,(r)\, v \# \,[r]\,, & \text{for } r : u \rightarrow v \in P, \\
 & & \beta_1, \beta_2 \in U \cup \{X_2\}\,,\ \delta_1, \delta_2 \in U \cup \{X_1\}\,, \\
2. & Y \# u\,[r]\,\$\,(r)\,\# v\alpha, & \text{for } r : u \rightarrow v \in P,\ \alpha \in U \cup \{X_2\}\,, \\
3. & \delta_1\delta_2 Y \alpha \# \beta_1\beta_2 \$ Z_1 \alpha Y \# Z_2, & \text{for } \alpha \in U,\ \beta_1, \beta_2 \in U \cup \{X_2\}\,, \\
 & & \delta_1, \delta_2 \in U \cup \{X_1\}\,, \\
4. & \delta \# Y \alpha Z_2 \$ Z_1 \# \alpha Y \beta, & \text{for } \alpha \in U,\ \delta \in U \cup \{X_1\}\,, \\
 & & \beta \in U \cup \{X_2\}\,, \\
5. & \delta \alpha Y \# \beta_1\beta_2\beta_3 \$ Z_1 Y \alpha \# Z_2, & \text{for } \alpha \in U,\ \beta_1 \in U,\ \beta_2, \beta_3 \in U \cup \{X_2\}\,, \\
 & & \delta \in U \cup \{X_1\}\,, \\
6. & \delta \# \alpha Y Z_2 \$ Z_1 \# Y \alpha \beta, & \text{for } \alpha \in U,\ \delta \in U \cup \{X_1\}\,, \\
 & & \beta \in U \cup \{X_2\}\,, \\
7. & \# YY \$ X_1^2 Y \# w, & \text{for } w \in \{X_2^2\} \cup T\,\{X_2^2\} \cup T^2\,\{X_2\} \cup T^3, \\
8. & \# X_2^2 \$ Y^3 \#. &
\end{array}$$

The idea behind this construction is the following. The rules in the groups 1 and 2 simulate rules in P, but only in the presence of the symbol Y. The rules in the groups 3 and 4 move the symbol Y to the right, the rules in the groups 5and 6 move the symbol Y to the left. The "main axiom" is w_0. All rules in the groups 1 – 6 involve a string derived from w_0 and containing such a symbol Y introduced by this axiom, in the sense that they can use only one axiom different from w_0. In any moment, we have two occurrences of X_1 at the beginning of a string and two occurrences of X_2 at the end of a string (maybe the same string). The rules in groups 1, 3, and 5 separate strings of the form $X_1^2 z X_2^2$ into two strings $X_1^2 z_1$, $z_2 X_2^2$, each one with multiplicity one; the rules in groups 2 and 4, 6 bring together these strings, leading to a string of the form $X_1^2 z' X_2^2$. The rules in the groups 7 and 8 remove the auxiliary symbols X_1, X_2, Y. If the remaining string is terminal, then it is an element of $L(G)$. The symbols $(r), [r]$ are associated with labels in L, Z_1 and Z_2 are associated with *moving* operations.

Using these explanations, the reader can easily verify that each derivation in G can be simulated in γ, hence we have $L(G) \subseteq L(\gamma)$ (an induction argument on the length of the derivation can be used, but the details are straightforward and tedious; we shall avoid such a strategy here).

Let us consider in some detail the opposite inclusion. We claim that if $A \Longrightarrow_\gamma^* M$ and $w \in T^*, M(w) > 0$, then $w \in L(G)$.

As we have pointed out above, by a direct check we can see that we cannot splice two of the axioms $w_r, w_\alpha, w'_\alpha, w_t$ (for instance, the symbols δ, β in rules in the group 4 and 6 prevent the splicing of w'_α and w_α). In the first step, we have to start with w_0, $w_0 = X_1^2 Y S X_2^2$, $A(w_0) = 1$. Now assume that we have a string $X_1^2 w_1 Y w_2 X_2^2$ with multiplicity 1. If w_2 starts with the left hand member of a rule in P, then we can apply to it a rule of type 1. Assume that this is the case, the string is $X_1^2 w_1 Y u w_3 X_2^2$ for some $r : u \to v \in P$. Using the axiom $(r)v[r]$ from A we obtain the strings

$$X_1^2 w_1 Y u[r],\ (r)v w_3 X_2^2.$$

No rule from the groups 1 and 3 – 8 can be applied to these strings, because so far no string containing Y^3 has been derived. From group 2, the rule $Y\#u[r]\$(r)\#v\alpha$ can be applied involving both these strings, which leads to

$$X_1^2 w_1 Y v w_3 X_2^2,\ (r)\,u\,[r]\,,$$

where the string $(r)\,u\,[r]$ can never enter a new splicing, because in the rule $r : u \to v$ from P we have assumed $u \neq v$. The multiplicity of $X_1^2 w_1 Y u[r]$ and $(r)v w_3 X_1^2$ has been reduced to 0 again (hence these strings are no more available), the multiplicity of $X_1^2 w_1 Y v w_3 X_2^2$ is one. In this way, we have passed from $X_1^2 w_1 Y u w_3 X_2^2$ to $X_1^2 w_1 Y v w_3 X_2^2$, which corresponds to using the rule $r : u \to v$ in P. Moreover we see that at each moment there is only one string containing X_1^2 and only one string (maybe the same) containing X_2^2 in the current multiset.

If to a string $X_1^2 w_1 Y \alpha w_3 X_2^2$ we apply a rule of type 3, then we get

$$X_1^2 w_1 Y \alpha Z_2,\ Z_1 \alpha Y w_3 X_2^2.$$

No rule form the groups 1 – 3 and 5 – 8 can be applied to these strings. By using a rule from group 4 we obtain

$$X_1^2 w_1 \alpha Y w_3 X_2^2,\ Z_1 Y \alpha Z_2.$$

The first string has replaced $X_1^2 w_1 Y \alpha w_3 X_2^2$ (hence we have interchanged Y with α), the second one is an axiom.

In the same way, one can see that using a rule from group 5 must be followed by using the corresponding rule of type 6, which results in interchanging Y with its left hand neighbour.

Consequently, in each moment we have a multiset with either one word $X_1^2 w_1 Y w_2 X_2^2$ or two words $X_1^2 z_1,\ z_2 X_2^2$, each one with multiplicity 1. Only in the first case, provided $w_1 = \lambda$, we can remove $X_1^2 Y$ by using a rule from group 7; then we can also remove X_2^2 by using the rule in group 8. This is the only way to remove these nonterminal symbols. If the obtained string is not terminal, then it cannot be further processed any more, because it does not contain the symbol Y. In conclusion, we can only simulate derivations in G and move Y freely in the string of multiplicity one, hence $L(\gamma) \subseteq L(G)$. $\square$

In the second part of this section we take another look on H systems of the form $\Gamma = (V, A, R)$, i.e. we do not specify a terminal alphabet in advance, and look at such systems in a slightly different way compared with the notations introduced for generating devices like grammars we have considered in the first part of this section: In the following we will assume that the H system "really" starts to work only if one additional single string is added. In the sense of multisets, we take exactly one copy of this starting string, whereas all the other strings in A are assumed to be available unboundedly (hence it is sufficient to specify the strings to appear as axioms without their common multiplicity ∞).

In this model we now can consider different possibilities for selecting the result of the computation:

1. We take every string w that is contained in a special regular language (i. e. we use intersection with regular languages, e. g. T^* for some $T \subseteq V$ as before).
2. We take every string w that cannot be processed any more (we call such strings *terminating*).
3. We take every string w not in A that has reached a "steady state", i. e. there exist rules in R that can be applied to w, but still yield w again.

We will use the following model of a deterministic Turing machine, which is equivalent to all the other models appearing in literature as *the* model of a mechanism defining computability:

A deterministic Turing machine M is an 8-tuple $(Q, q_0, q_f, V, V_T, Z_0, B, \delta)$, where Q is the (finite) set of states, q_0 is the initial state, q_f is the finite state, V is the (finite) alphabet of tape symbols, $V_T \subseteq V$ is the set of terminal symbols, $Z_0 \in V$ is the left boundary symbol, $V_0 := V - \{Z_0\}$, $B \in V_0$ is the blank symbol, $\delta : Q \times V \rightarrow Q \times V \times \{L, R\}$ is the transition function with the following restrictions (the fact $(p, Y, D) \in \delta(q, X)$ will be expressed by the relation $(q, X, p, Y, D) \in \delta$):

- $(q, X, p, Y, L) \in \delta$ implies $X \in V_0$, i. e. $X \neq Z_0$ (Z_0 marks the left boundary of the semi-infinite tape);
- $(q, Z_0, p, Y, D) \in \delta$, $D \in \{L, R\}$, implies $Y = Z_0$ and $D = R$, i. e. Z_0 cannot be rewritten, and reading Z_0 the Turing machine can only go to the right;
- $(q, X, p, Z_0, R) \in \delta$ implies $X = Z_0$, i. e. Z_0 cannot be written except at the beginning of the tape;
- $(q, X, p, Y, D) \in \delta$, $D \in \{L, R\}$, implies $q \neq q_f$, i. e. there is no transition from the final state q_f, whereas
- for all $q \in Q - \{q_f\}$ and all $X \in V$ there is some transition (q, X, p, Y, D) in δ.

The Turing machine M works on a semi-infinite tape with left boundary marker Z_0. An instantaneous description of that tape during a computation of M is of the form Z_0uqvB^ω with $u, v \in V^*$ and $q \in Q$, which describes the situation that M is in state q, the head of the Turing machine looks at the rightmost symbol of Z_0u, and the contents of the tape is Z_0uvB^ω, where the notation B^ω just describes the fact that to the right we find an infinite number of blank symbols B.

The effect of a transition specified by δ on such a *configuration* Z_0uqvB^ω is defined as follows:

- Applying $(q, X, p, Y, L) \in \delta$ (remember that we have the restriction $X \neq Z_0$) yields $Z_0 u' p Y v B^\omega$ from $Z_0 u' X q v B^\omega$.
- Applying $(q, X, p, Y, R) \in \delta$ yields $Z_0 u' Y U p v' B^\omega$ from $Z_0 u' X q U v' B^\omega$.

It is well-known that such a model for (deterministic) Turing machines as defined above is one of the general models for *computability*, i. e.

- for every computable (partial) function $f : T^* \to T^*$ there exists a Turing machine M_f such that
  1. if $f(w)$, $w \in T^*$, is defined, M_f started with the initial configuration $Z_0 w q_0 B^\omega$ computes $f(w)$ by ending up in the final configuration $Z_0 f(w) q_f B^\omega$, and
  2. if $w \in T^*$ is not contained in *dom* (f), the domain of the function f, then M_f started with the initial configuration $Z_0 w q_0 B^\omega$ never halts (i. e. M_f never enters the final state q_f);
- for each alphabet T there exists a *universal* Turing machine $M_{U,T}$, i. e. every Turing machine M with terminal alphabet T can be encoded as a string $C(M) \in \{c_1, c_2\}^+$ in such a way that $M_{U,T}$ starting with the initial configuration $Z_0 C(M) c_3 w q_0 B^\omega$, $w \in T^*$, halts in the final state q_f if and only if $f_M(w)$ is defined (where c_1, c_2, c_3 are new symbols and f_M is the partial function induced by the Turing machine M) and moreover the final configuratoin is $Z_0 f_M(w) q_f B^\omega$.

In the following theorem we show how H systems (with multisets) can simulate the actions of Turing machines; hence we can also construct universal H systems, i. e. H systems Γ, $\Gamma = (V, A, R)$, which from a representation of a given partial function f, $f : T^* \to T^*$, and a representation of the input data w compute a representation of $f(w)$ provided $w \in$ *dom*(f). The final result $f(w)$ can be filtered out by several techniques as they were considered before, e. g. by taking the intersection with T^*.

For the proof of the following theorem we choose to select exactly those strings that cannot be processed any more.

Theorem 2. *Let $f : T^* \to T^*$ be a partial recursive function, and let*

$$M_f = (Q, q_0, q_f, V, T, Z_0, B, \delta)$$

be a deterministic Turing machine computing f (i. e. for every $w \in T^*$, starting with the initial configuration $Z_0 w q_0 B^\omega$, M_f halts with the final configuration $Z_0 f(w) q_f B^\omega$ if $w \in dom(f)$, and never halts otherwise). Then we can effectively construct an H system $\Gamma_f = (V', A, R)$ which also computes f in the following way:

For an arbitrary $w \in T^*$, the computation with Γ_f on w is initialized by the string $Z_0 w q_0 Z_1$; if $w \in dom(f)$, then finally the string $Z_0 f(w) q_f Z_1$ will be derivable and also be terminating (no other string can be obtained from this string any more); if $w \notin dom(f)$, no terminating string is derivable.

Proof. From M_f we construct Γ_f in the following way: $\Gamma = (V', A, R)$ with

$$V' = V \cup \{Z_1, Z_2, Z_3\} \cup \{(r), [r] \mid r \in \delta\}$$

and $A = A_1 \cup A_2$, where

$$\begin{array}{rcl}
A_1 & = & \{Z_0UpZ_2 \mid \text{for some } q \in Q \;\; (q,Z_0,p,Z_0,R) \in \delta,\; U \in V_0,\; p \in Q\} \cup \\
 & & \{(r)\,YUp\,[r] \mid r=(q,X,p,Y,R) \in \delta,\; U,X,Y \in V_0,\; p,q \in Q\} \cup \\
 & & \{(r)\,pY\,[r] \mid r=(q,X,p,Y,L) \in \delta,\; X,Y \in V_0,\; p,q \in Q\} \cup \\
 & & \{Z_3qBZ_1 \mid q \in Q - \{q_f\}\} \cup \{Z_3q_fZ_2\}\,, \\
A_2 & = & \{Z_0qUZ_2 \mid \text{for some } q \in Q \;\; (q,Z_0,p,Z_0,R) \in \delta,\; U \in V_0,\; p \in Q\} \cup \\
 & & \{(r)\,XqU\,[r] \mid r=(q,X,p,Y,R) \in \delta,\; U,X,Y \in V_0,\; p,q \in Q\} \cup \\
 & & \{(r)\,Xq\,[r] \mid r=(q,X,p,Y,L) \in \delta,\; X,Y \in V_0,\; p,q \in Q\} \cup \\
 & & \{Z_3qZ_1 \mid q \in Q - \{q_f\}\} \cup \{Z_3q_fBZ_2\}\,.
\end{array}$$

The set R contains the following splicing rules:

1.	$Z_0qU\#C\$Z_0Up\#Z_2$	for $(q,Z_0,p,Z_0,R) \in \delta,\; U \in V_0,\; p,q \in Q,$ $C \in V_0 \cup \{Z_1\}$;
2.	$DXqU\#C\$(r)YUp\#\,[r]$	for $r=(q,X,p,Y,R) \in \delta,\; U,X,Y \in V_0,\; p,q \in Q,$ $C \in V_0 \cup \{Z_1\},\; D \in V$;
3.	$D\#XqU\,[r]\,\$(r)\#YUpC$	for $r=(q,X,p,Y,R) \in \delta,\; U,X,Y \in V_0,\; p,q \in Q,$ $C \in V_0 \cup \{Z_1\},\; D \in V$;
4.	$DXq\#C\$(r)pY\#\,[r]$	for $r=(q,X,p,Y,L) \in \delta,\; X,Y \in V_0,\; p,q \in Q,$ $C \in V_0 \cup \{Z_1\},\; D \in V$;
5.	$D\#Xq\,[r]\,\$(r)\#pYC$	for $r=(q,X,p,Y,L) \in \delta,\; X,Y \in V_0,\; p,q \in Q;$ $C \in V_0 \cup \{Z_1\},\; D \in V$;
6.	$U\#qZ_1\$Z_3\#qBZ_1$	for $q \in Q - \{q_f\}$ and $U \in V$;
7.	$D\#q_fB\$Z_3\#q_fZ_2$	for $D \in V$;
8.	$Dq_f\#Z_2\$Z_3q_fB\#C$	for $D \in V,\; C \in \{B, Z_1\}$;
9.	$w\#\$\#w$	for all $w \in A$.

Any configuration Z_0uqvB^{ω} in M is represented by some finite string $Z_0uqvB^mZ_1$, for some $m \geq 0$, in the H system Γ. By using the rules in group 6 we can add a new blank symbol at the right-hand side of the string just to the left of the right marker Z_1. If some string $Z_0f(w)q_fB^mZ_1$ representing the final configuration $Z_0f(w)q_fB^{\omega}$ has been derived, the desired terminal finite string $Z_0f(w)q_fZ_1$ that cannot be derived any more can be obtained by using the rules in the groups 7 and 8. The rules in the groups 1, 2 and 3 as well as 4 and 5 allow us to simulate the transitions from δ with the head moving to the right on Z_0, moving to the right on a symbol $\neq Z_0$ and rewriting the symbol, as well as moving to the left and rewriting the symbol.

The only terminating string not in A obtained by using the rules from R from an initial string $Z_0wq_0Z_1$ therefore is the final string $Z_0f(w)q_fZ_1$ provided $w \in dom(f)$; because of the rules in group 9 this final string even is the only string that cannot be derived any more. For $w \notin dom(f)$ no terminating string is derivable from the initial string $Z_0wq_0Z_1$. □

If we want to select the final strings via the "steady state" condition, in the proof of the preceding theorem we only have to replace group 9 by group 9′:

$$9'. \quad q_f\#Z_1\$q_f\#Z_1$$

Moreover, we have to add $q_f Z_1$ to A. Then we obtain another method for selecting the final string (configuration), i. e. by taking exactly those strings not in A that reach a steady state; obviously, $q_f Z_1$ will be the only string in A also fulfilling the steady state condition.

Another way to select the final strings (configurations) is to apply the "filter" $\{Z_0\}\, T^*\, \{q_f Z_1\}$.

Finally we have to mention that the proof of the preceding theorem could be extended in such a way that from the final configuration $Z_0 f(w) q_f Z_1$ we could obtain $f(w)$ itself as the final string that cannot be processed any more (in fact, similar constructions like in the proof of Theorem 1 can be used) which also corresponds to using a "filter" T^*.

The following result proving the universality of H systems (with multisets) with respect to computability can be proved by using similar techniques as in the proof if the previous theorem:

Corollary 1. *Let T be an arbitrary alphabet. Then we can effectively construct an H system Γ, $\Gamma = (V, A, R)$, with $T \subseteq V$ such that Γ can compute every partial recursive function f in the following way: Started with the initial string $Z_0 C(M_f) c_3 w q_0 Z_1$ where $C(M_f)$ is the code of a deterministic Turing machine M_f realizing f, $f : T \to T^*$, Γ for $w \in dom(f)$ computes the terminating string $Z_0 f(w) q_f Z_1$, whereas for $w \notin dom(f)$ no terminating string (not in A) is derivable.*

The result above can again be obtained for other selection strategies, too, as already elaborated after the proof of Theorem 2.

In the following sections we will restrict ourselves to the selection of the terminal strings by specifying a terminal alphabet for extended H systems.

4 Regular extended H systems

In [6] it is proved that $H(FIN, FIN) \subseteq REG$ (in fact, also $H(REG, FIN) \subseteq REG$); the proof has been simplified in [21]. As REG is closed under intersection, it follows that $EH(REG, FIN) \subseteq REG$, too. The converse inclusion is proved in [18], hence we can state

Theorem 3. *$EH(FIN, FIN) = EH(F, FIN) = REG$ for all $FIN \subseteq F \subseteq REG$.*

Surprisingly enough, the extended H systems of the next complexity level after those with finite sets of axioms are already powerful enough to equalize the power of Turing machines (and of any other language describing class of algorithms). We here recall the construction in the proof of this result in [18], because we shall use its main ideas in subsequent sections of this paper.

Theorem 4. *$EH(FIN, REG) = EH(F_1, F_2) = RE$ for all $FIN \subseteq F_1 \subseteq RE$, $REG \subseteq F_2 \subseteq RE$.*

Proof. The inclusions $EH(FIN, REG) \subseteq EH(F_1, F_2)$ for F_1, F_2 as above are obvious, and because of the Turing/Church thesis $EH(F_1, F_2) \subseteq RE$ is obvious. Hence it is sufficient to prove that $RE \subseteq EH(FIN, REG)$:

Consider a type-0 Chomsky grammar $G = (N, T, S, P)$ and construct the extended H system $\gamma = (V, T, A, R)$, where

$$\begin{array}{rcl} V & = & N \cup T \cup \{X, X', B, Y, Z\} \cup \{Y_\alpha \mid \alpha \in N \cup T \cup \{B\}\}\,, \\ A & = & \{XBSY, ZY, XZ\} \cup \{ZvY \mid u \to v \in P\} \cup \{ZY_\alpha, X'\alpha Z \mid \alpha \in N \cup T \cup \{B\}\}\,, \end{array}$$

and R contains the following groups of rules:

1. $Xw\#uY\$Z\#vY$ for $u \to v \in P,\ w \in (N \cup T \cup \{B\})^*$;
2. $Xw\#\alpha Y\$Z\#Y_\alpha$ for $\alpha \in N \cup T \cup \{B\},\ w \in (N \cup T \cup \{B\})^*$;
3. $X\#wY_\alpha\$X'\alpha\#Z$ for $\alpha \in N \cup T \cup \{B\},\ w \in (N \cup T \cup \{B\})^*$;
4. $X'w\#Y_\alpha\$Z\#Y$ for $\alpha \in N \cup T \cup \{B\},\ w \in (N \cup T \cup \{B\})^*$;
5. $X'\#wY\$X\#Z$ for $w \in (N \cup T \cup \{B\})^*$;
6. $XB\#wY\$\#ZY$ for $w \in T^*$;
7. $\#Y\$XZ\#$.

We can show that we obtain $L(\gamma) = L(G)$. □

5 H systems with context conditions

A natural way to regulate the application of the splicing rules is to use context conditions as in random context grammars: associate sets of symbols/strings to rules and use a rule only when the associated symbols/strings are present in the currently spliced strings (permitting contexts) or they are not present (forbidden contexts). Formally, we consider

Definition 4. An extended ωH system *with permitting contexts* is a quadruple $\gamma = (V, T, A, R)$, where V, T, A are as in Definition 1 and R is a set of triples (we call them rules with permitting contexts) of the form

$$p = (r;\ C_1, C_2) \text{ with } r = u_1\#u_2\$u_3\#u_4,$$

where $u_1\#u_2\$u_3\#u_4$ is a splicing rule over V and C_1, C_2 are finite subsets of V^+.

For $x, y, z, w \in V^*$ and $p \in R$ as above, we define $(x, y) \vdash_p (z, w)$ if and only if $(x, y) \vdash_r (z, w)$, every string contained in C_1 appears as a substring in x and every string contained in C_2 appears as a substring in y (of course, when $C_1 = \emptyset$ or $C_2 = \emptyset$, then this imposes no restriction on the use of the rule p). □

The language generated by γ is defined in the natural way, and the family of languages $L(\gamma)$, for $\gamma = (V, T, A, R)$ as above, with $A \in F_1$ and R having the set of strings $u_1\#u_2\$u_3\#u_4C_1C_2$ in the rules with permitting contexts in a family F_2, is denoted by $EH(\omega F_1, cF_2)$.

Theorem 5. *$EH(\omega FIN, cFIN) = EH(\omega F_1, cF_2) = RE$ for all families F_1, F_2 such that $FIN \subseteq F_1 \subseteq RE$, $FIN \subseteq F_2 \subseteq RE$.*

Proof. The inclusions $EH(\omega FIN, cFIN) \subseteq EH(\omega F_1, cF_2) \subseteq RE$ are obvious, hence it is sufficient to prove the inclusion $RE \subseteq EH(\omega FIN, cFIN)$.

Consider a type-0 Chomsky grammar $G = (N, T, S, P)$ like in the proof of Theorem 1; let L denote the set of labels of the rules in P and denote $U' = U \cup \{B\}$, $U = N \cup T$. We now construct the extended ωH system with permitting contexts $\gamma = (V, T, A, R)$, where

$$\begin{aligned} V &= U \cup \{B, E, E', F, F', X, X', Y, Z\} \cup \{Y_\alpha \mid \alpha \in U' \cup L\}\,, \\ A &= \{F'Z, XBSY, XZ, ZE, ZE', ZF, ZY\} \cup \\ &\quad\ \{ZY_\alpha, X'\alpha Z \mid \alpha \in U'\} \cup \{ZY_r, X'vZ \mid r : u \to v \in P\} \end{aligned}$$

and R contains the following rules with permitting contexts:

$$\begin{array}{rll}
1. & (\#uY\$Z\#Y_r; \{X\}, \emptyset)\,, & \text{for } r : u \to v \in P, \\
2. & (X\#\$X'v\#Z; \{Y_r\}, \emptyset)\,, & \text{for } r : u \to v \in P, \\
3. & (\#Y_r\$Z\#Y; \{X'\}, \emptyset)\,, & \text{for } r : u \to v \in P, \\
4. & (X'\#\$X\#Z; \{Y\}, \emptyset)\,, & \\
5. & (\#\alpha Y\$Z\#Y_\alpha; \{X\}, \emptyset)\,, & \text{for } \alpha \in U', \\
6. & (X\#\$X'\alpha\#Z; \{Y_\alpha\}, \emptyset)\,, & \text{for } \alpha \in U', \\
7. & (\#Y_\alpha\$Z\#Y_\alpha; \{X'\}, \emptyset)\,, & \text{for } \alpha \in U', \\
8. & (\#Y\$Z\#F; \{X\}, \emptyset)\,, & \\
9. & (XB\#\$F'\#Z; \{F\}, \emptyset)\,, & \\
10. & (\#F\$ZE\#; \{F'\}, \emptyset)\,, & \\
11. & (F'\#\$\#ZE'; \{F'\}, \emptyset)\,. &
\end{array}$$

The idea behind this construction is the following. The rules from the groups 1, 2, 3, and 4 allow us to simulate rules from P on a suffix of the first term of the splicing. A rule in group 1 cuts the left-hand side u of the production $r : u \to v \in P$ from the right-hand end of the string and the associated symbol Y_r memorizes the label of this rule; in the presence of Y_r a rule from group 2 will introduce the right-hand side v of the rule with label r on the left-hand end of the string together with X' instead of X; then Y_r is again replaced by Y (by using the appropriate rule from group 3), and X' is again replaced by X (by using the rule from group 4).

However, we must be able to simulate the application of a rule from P at an arbitrary position of the underlying sentential form, not only at the right-hand end of the string. To this aim, the rules in the groups 5, 6, 7, and 4 allow us to "rotate" the string: A rule in group 5 cuts a symbol α from the right-hand end of the string, Y_α memorizes this symbol, in its presence a rule from group 6 will introduce α in the left hand end (together with X'), then Y_α is again replaced by Y (by using the appropriate rule from group 7), and X' is again replaced by X (by using the rule from group 4). Any circular permutation can be obtained in this way.

In a quite similar way, the rules from the groups 8, 9, 10, and 11 finally allow us to remove the markers X and Y by first replacing Y by F and X by F' and then by removing F and F'.

We obtain $L(\gamma) = L(G)$. The detailed proof of this equality can be found in [11]. □

Instead of controlling the applicability of a splicing rule by using permitting contexts, i. e. by checking the occurrence of specific substrings (symbols) in the underlying strings, we can also control the applicability of a splicing rule by using forbidden contexts, i. e. by forbidding the occurrence of specific substrings (symbols) in the underlying strings.

These forbidding contexts can be interpreted as *inhibitors* of the associated rules (and they can be checked, manually, as in [1]).

Definition 5. An extended ωH system *with forbidden contexts* is a quadruple $\gamma = (V, T, A, R)$, where V, T, A are as in Definition 1 and R is a set of triples (we call them rules with forbidden contexts) of the form

$$p = (r;\ D_1, D_2) \text{ with } r = u_1\#u_2\$u_3\#u_4,$$

where $u_1\#u_2\$u_3\#u_4$ is a splicing rule over V and D_1, D_2 are finite subsets of V^+.

For $x, y, z, w \in V^*$ and $p \in R$ as above, we define $(x, y) \vdash_p (z, w)$ if and only if $(x, y) \vdash_r (z, w)$, no string contained in D_1 appears as a substring in x and no string contained in D_2 appears as a substring in y (of course, when $D_1 = \emptyset$ or $D_2 = \emptyset$, then this imposes no restriction on the use of the rule p). □

The language generated by γ is defined in the natural way, and the family of languages $L(\gamma)$, for $\gamma = (V, T, A, R)$ as above, with $A \in F_1$ and R having the set of strings $u_1\#u_2\$u_3\#u_4D_1D_2$ in the rules with forbidden contexts in a family F_2 is denoted by $EH(\omega F_1, fF_2)$.

Theorem 6. *$EH(\omega FIN, fFIN) = EH(\omega F_1, fF_2) = RE$, for all families F_1, F_2 such that $FIN \subseteq F_1 \subseteq RE$, $FIN \subseteq F_2 \subseteq RE$.*

Proof. Again, it is sufficient to prove that $RE \subseteq EH(\omega FIN, FIN)$.

Consider a type-0 grammar $G = (N, T, S, P)$ like in the proof of Theorem 1, let L be the set of labels of the rules in P, and denote $U' = N \cup T \cup \{B\}$. We now construct the ωH system with forbidden contexts $\gamma = (V, T, A, R)$, where V, T, and A are as in the proof of the preceding theorem, and R contains the following rules with forbidden contexts:

1. $(\#uY\$Z\#Y_r; V - (U' \cup \{X, Y\}), \emptyset)$, for $r : u \to v \in P$,
2. $(X\#\$X'v\#Z; V - (U' \cup \{X, Y_r\}), \emptyset)$, for $r : u \to v \in P$,
3. $(\#Y_r\$Z\#Y; V - (U' \cup \{X', Y_r\}), \emptyset)$, for $r : u \to v \in P$,
4. $(X'\#\$X\#Z; V - (U' \cup \{X', Y\}), \emptyset)$,
5. $(\#\alpha Y\$Z\#Y_\alpha; V - (U' \cup \{X, Y\}), \emptyset)$, for $\alpha \in U'$,
6. $(X\#\$X'\alpha\#Z; V - (U' \cup \{X, Y_\alpha\}), \emptyset)$, for $\alpha \in U'$,
7. $(\#Y_\alpha\$Z\#Y_r; V - (U' \cup \{X', Y_\alpha\}), \emptyset)$, for $\alpha \in U'$,
8. $(\#Y\$Z\#F; V - (T \cup \{B, X, Y\}), \emptyset)$,
9. $(XB\#\$F'\#Z; V - (T \cup \{B, F, X\}), \emptyset)$,
10. $(\#F\$ZE\#; V - (T \cup \{F, F'\}), \emptyset)$,
11. $(F'\#\$\#ZE'; V - (T \cup \{F'\}), \emptyset)$.

In an even more restrictive way than the permitting contexts in the rules of the ωH system with permitting contexts constructed in the proof of Theorem 5, the forbidden contexts in the ωH system with forbidden contexts constructed above control the derivation sequences possible in γ. Hence, again we conclude $L(\gamma) = L(G)$. □

6 H systems as universal generating mechanisms

The results in the previous sections prove that *finite* H systems of the considered types are computationally complete, but this does not mean that *programmable* computers based on splicing can be constructed. To this aim, it is necessary to find *universal H systems*, i. e. systems with all components but one (the set of axioms) fixed, able to behave as any given H system γ, when a code of γ is introduced in the set of axioms of the universal system.

Definition 6. Given an alphabet T and two families of languages, F_1, F_2, a construct

$$\gamma_U = (V_U, T, A_U, R_U),$$

where V_U is an alphabet, $A_U \in F_1$, and $R_U \subseteq V_U^* \# V_U^* \$ V_U^* \# V_U^*$, $R_U \in F_2$, is said to be a *universal* H system of type (F_1, F_2), if for every $\gamma = (V, T, A, R)$ of any type (F_1', F_2') there is a language A_γ such that $A_U \cup A_\gamma \in F_1$ and $L(\gamma) = L(\gamma_U')$, where $\gamma_U' = (V_U, T, A_U \cup A_\gamma, R_U)$. $\square$

The particularizations of this definition to mH systems or to ωH systems with permitting respectively forbidden contexts are obvious.

Note that the type (F_1, F_2) of the universal system is fixed, but the universal system is able to simulate systems *of any type* (F_1', F_2').

The restriction to a given terminal alphabet cannot be avoided, but this is anyway imposed by the fact that the DNA alphabet has only four letters. It is perhaps no surprise why this alphabet has been chosen: it is the smallest one by which we can codify two disjoint arbitrarily large alphabets (terminal and nonterminal symbols in our terminology), using two disjoint subsets of it. This is known in language and information theory in general, but this works also in the H systems area.

Theorem 7. *For every given alphabet T there exists an mH system of type $(mFIN, FIN)$ which is universal for the class of mH systems with the terminal alphabet T.*

Proof. Consider an alphabet T and two different symbols c_1, c_2 not in T.

For the class of type-0 Chomsky grammars with given terminal alphabet T, there are universal grammars, i. e. constructs $G_U = (N_U, T, -, P_U)$ such that for any given grammar $G = (N, T, S, P)$ there is a string $w(G) \in (N_U \cup T)^*$ (the "code" of G) such that $L(G_U') = L(G)$ for $G_U' = (N_U, T, w(G), P_U)$. (In fact, the set of nonterminals N and the set of productions P from G can easily be encoded in the alphabet $\{c_1, c_2\}$. The language $L(G_U')$ then consists of all terminal strings z such that $w(G) \Longrightarrow^* z$ using the rules in P_U.) This follows from the existence of universal Turing machines [22] and the way of passing from Turing machines to type-0 grammars and conversely, or it can be proved directly (an effective construction of a universal type-0 grammar can be found in [3]).

For a given universal type-0 grammar $G_U = (N_U, T, -, P_U)$, we follow the construction in the proof of Theorem 1, obtaining an mH system $\hat{\gamma}_U = (\hat{V}_U, T, \hat{A}_U, \hat{R}_U)$, where the axiom (with multiplicity 1) $X_1^2 Y S X_2^2$ is not considered. Remark that all other axioms in $\hat{A}_U$ (all having infinite multiplicity) and the rules in $\hat{R}_U$ depend on N_U, T and P_U only, hence they are fixed.

All symbols in $\hat{V}_U - T$ now can be codified by strings over $\{c_1, c_2\}$; the obtained system,

$$\gamma_U = (\{c_1, c_2\} \cup T, T, A_U, R_U)$$

is the universal mH system we are looking for.

Indeed, take an arbitrary mH system $\gamma_0 = (V, T, A, R)$ and construct a type-0 grammar $G_0 = (N_0, T, S_0, P_0)$ such that $L(\gamma_0) = L(G_0)$ (the grammar G_0 can be constructed directly and in an effective way; because of the Turing/Church thesis we have left this obvious construction to the reader). Construct the code of G_0, $w(G_0)$, as imposed by the definition of universal type-0 grammars, consider the string $X_1^2 Y w(G_0) X_2^2$ corresponding to the axiom $X_1^2 Y S X_2^2$ in the proof of Theorem 1, then codify $X_1^2 Y w(G_0) X_2^2$ over $\{c_1, c_2\} \cup T$, and denote the obtained string by $w(\gamma_0)$. Then $L(\gamma_U') = L(\gamma_0)$, for $\gamma_U' = (\{c_1, c_2\} \cup T, T, \{(w(\gamma_0), 1)\} \cup A_U, R_U)$. □

Theorem 8. *For every given alphabet T, there is an ωH system of type $(\omega FIN, FIN)$ with permitting respectively forbidden contexts that is universal for the class of ωH systems with permitting respectively forbidden contexts and with the terminal alphabet T.*

Proof. This result can be proved in the same way as in Theorem 7 above. □

Remark that the universal H systems γ_U furnished by the previous proofs are enabled to simulate any given H system γ by adding one more axiom to γ_U (of multiplicity one in the case of mH systems). The computers based on γ_U seem to be quite economical as for as the way to program them is concerned.

7 Test tube systems

In this section we investigate a new model for biological computers that incorporates basic ideas of parallel communicating grammar systems.

Definition 7. A *test tube* (TT for short) *system of degree* n, $n \geq 1$, is a construct

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

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

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

For two configurations $(L_1, ..., L_n)$, $(L'_1, ..., L'_n)$ we define

$$\begin{array}{rcl}(L_1, ..., L_n) & \Longrightarrow & (L'_1, ..., L'_n) \text{ if and only if for each } i,\ 1 \leq i \leq n, \\ & & L'_i = \bigcup_{j=1}^{n} \left(\sigma_j^* (L_j) \cap V_i^*\right) \cup \left(\sigma_i^* (L_i) \cap B\right).\end{array}$$

In words, the contents of each tube is spliced according to the associated set of rules (we pass from L_i to $\sigma^* (L_i)$, $1 \leq i \leq n$), and the result is redistributed among the n tubes according to the selectors $V_1, ..., V_n$; the part which cannot be redistributed, because it does not belong to some V_i^*, $1 \leq i \leq n$, remains in the tube. When a string belongs to several languages V_i^*, then copies of it will be distributed to all tubes i with this property.

A *computation* of length k, $k \geq 1$, with respect to Γ is a sequence of configurations

$$\left(L_1^{(0)}, ..., L_n^{(0)}\right) \left(L_1^{(1)}, ..., L_n^{(1)}\right) ... \left(L_1^{(k)}, ..., L_n^{(k)}\right)$$

such that

1. $\left(L_1^{(0)}, ..., L_n^{(0)}\right) = (A_1, ..., A_n)$.
2. $\left(L_1^{(t)}, ..., L_n^{(t)}\right) \Longrightarrow \left(L_1^{(t+1)}, ..., L_n^{(t+1)}\right)$, with respect to Γ for each t, $0\leq t \leq k-1$.

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

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

where $C_0 (\Gamma) = \{(A_1, ..., A_n)\}$.

The i-th *result* of a computation $C = \left(L_1^{(0)}, ..., L_n^{(0)}\right) \left(L_1^{(1)}, ..., L_n^{(1)}\right) ... \left(L_1^{(k)}, ..., L_n^{(k)}\right)$ is the set of all strings which were present in the tube i, that is

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

By convention, the *language generated* by a TT system Γ is the result of all computations in the tube 1, i. e.

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

More compactly, we write

$$\begin{array}{rcl}L (\Gamma) & = & \Big\{w \in V^* \mid w \in L_1^{(+)} \text{ for some } \left(L_1^{(t)}, ..., L_n^{(t)}\right),\ t \geq 0, \\ & & \text{such that } (A_1, ..., A_n) \Longrightarrow^* \left(L_1^{(t)}, ..., L_n^{(t)}\right)\Big\} ,\end{array}$$

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

Definition 8. Given two families of languages, F_1, F_2, by $TT_n (F_1, F_2)$ we denote the family of languages $L (\Gamma)$, for $\Gamma = (V, (A_1, R_1, V_1) , ..., (A_m, R_m, V_m))$, with $m \leq n$, $A_i \in F_1$,

$R_i \in F_2$, for all i, $1 \leq i \leq m$. (We say that Γ is of type (F_1, F_2).) When n is not specified, we replace it by $*$, that is we write

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

$\square$

A TT system as above has a structure very similar to that of a parallel communicating (PC for short) grammar system. The rewriting steps in a PC grammar system here correspond to the splicing phases, that is to passing from $L_i^{(t)}$ to $\sigma^*\left(L_i^{(t)}\right)$, whereas the communication steps correspond to the redistribution of $\sigma^*\left(L_i^{(t)}\right)$ to the n tubes according to the selectors $V_1, ..., V_n$. 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 performed automatically after every splicing step. Moreover, communication here means a *separate* operation in the sense of [16], [2].

Theorem 9. *$TT_7(FIN, FIN) = TT_*(FIN, FIN) = TT_*(F_1, F_2) = RE$, for all families F_1, F_2 such that $FIN \subseteq F_i \subseteq RE$, $i = 1, 2$.*

Proof. The inclusions $TT_7(FIN, FIN) \subseteq TT_*(FIN, FIN) \subseteq TT_*(F_1, F_2)$ are obvious, $TT_*(F_1, F_2) \subseteq RE$ is obvious from the Turing/Church thesis, hence it is sufficient to prove that $RE \subseteq TT_7(FIN, FIN)$.

Take a type-0 Chomsky grammar $G = (N, T, S, P)$, denote $U = N \cup T$ and construct the TT system

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

with

$$V = N \cup T \cup \{X, X', Y, Z, Z', B\} \cup \{Y_\alpha \mid \alpha \in U \cup \{B\}\}$$

and

- $A_1 = \emptyset$,

  $R_1 = \emptyset$,

  $V_1 = T$,

- $$\begin{aligned} A_2 &= \{XBSY, Z'Z\} \cup \\ &\quad \{ZvY \mid u \to v \in P\} \cup \\ &\quad \{ZY_\alpha \mid \alpha \in U \cup \{B\}\}, \\ R_2 &= \{\#uY\$Z\#vY \mid u \to v \in P\} \cup \\ &\quad \{\#\alpha Y\$Z\#Y_\alpha \mid \alpha \in U \cup \{B\}\} \cup \\ &\quad \{Z'\#Z\$XB\#\}, \\ V_2 &= U \cup \{B, X, Y\}, \end{aligned}$$

- $$\begin{aligned} A_3 &= \{X'\alpha Z\}, \\ R_3 &= \{X'\alpha\#\$X\# \mid \alpha \in U \cup \{B\}\}, \\ V_3 &= U \cup \{B, X\} \cup \{Y_\alpha \mid \alpha \in U \cup \{B\}\}, \end{aligned}$$

- $A_4 = \{ZY\}$,
  $R_4 = \{\#Y_\alpha\$Z\#Y \mid \alpha \in U \cup \{B\}\}$,
  $V_4 = U \cup \{B, X'\} \cup \{Y_\alpha \mid \alpha \in U \cup \{B\}\}$,

- $A_5 = \{XZ\}$,
  $R_5 = \{X\#Z\$X'\#\}$,
  $V_5 = U \cup \{B, X', Y\}$,

- $A_6 = \{ZZ\}$,
  $R_6 = \{\#Y\$ZZ\#\}$,
  $V_6 = T \cup \{Y, Z'\}$,

- $A_7 = \{ZZ\}$,
  $R_7 = \{\#ZZ\$Z'\#\}$,
  $V_7 = T \cup \{Z'\}$.

Let us examine the work of Γ:

The first component only selects the strings produced by the other components that are terminal according to G. No such terminal string can enter a splicing, because all rules in $R_2 - R_7$ involve symbols X, X', Y, Z, Y_α, for $\alpha \in U \cup \{B\}$. Tubes 2, 3, 4 and 5 simulate derivations of sentential forms of G, while tubes 6 and 7 are for testing if the derivation has successfully been terminated by yielding a terminal word. In tube 2 applications of productions of the form $u \to v \in P$ to sentential forms Xw_1Bw_2uY are simulated, where w_2uw_1 is a sentential form of G, and X, B, Y are special symbols, X and Y indicating the left respectively the right end of the sentential form in Γ and B specifying the beginning of the rotated string representing the corresponding sentential form in G. Moreover, tubes 2, 3, 4 and 5, by passing the strings to each other in this order and again to tube 2, by using a "rewind technique" explained below, guarantee that the applications of productions of G are simulated at the correct place in the string. The construction works as follows:

In the initial configuration $(A_1, ..., A_7)$, only the second component can execute a splicing. There are three possibilities: we can use a rule of the form $\#uY\$Z\#vY$, for $u \to v \in P$ (we say that this is a splicing of type 1), a rule of the form $\#\alpha Y\$Z\#Y_\alpha$ for $\alpha \in U \cup \{B\}$ (a splicing of type 2) or the rule $Z'\#Z\$XB\#$ (a splicing of type 3).

Consider the general case, of having in tube 2 a string XwY, with $w \in U^*BU^*$; initially, $w = BS$. We have three possibilities for splicings (in order to elucidate the effect of the application of a splicing rule, in the following we will indicate the position where the underlying strings are cut, by vertical strokes):

1. $(Xw_1 \mid uY, Z \mid vY) \vdash_1 (Xw_1vY, ZuY)$ for $u \to v \in P$ and $w = w_1u$,

2. $(Xw_1 \mid \alpha Y, Z \mid Y_\alpha) \vdash_2 (Xw_1Y_\alpha, Z\alpha Y)$ for $\alpha \in U \cup \{B\}$ and $w = w_1\alpha$,

3. $(Z' \mid Z, XB \mid w_1Y) \vdash_3 (Z'w_1Y, XBZ)$ for $w = Bw_1$.

The string Xw_1vY is of the same form as the string Xw_1uY and therefore 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:

1. if ZuY is an axiom, then nothing new appears;
2. ZuY is used as the first term of a splicing of the form $(Zu_1 \mid u'Y, Z \mid v'Y) \vdash_1 (Zu_1v'Y, Zu'Y)$, for $u = u_1u'$ and $u' \to v' \in P$; we obtain two strings of the same form, ZxY, which will remain in tube 2;
3. ZuY is used as the first term of a splicing of the form $(Zu_1 \mid \alpha Y, Z \mid Y_\alpha) \vdash_2 (Zu_1Y_\alpha, Z\alpha Y)$, for $u = u_1\alpha$, $\alpha \in U \cup \{B\}$; the string Zu_1Y_α cannot enter new splicings and cannot be transmitted to another tube.

After any sequence of such splicings, the obtained strings still will be of the form ZxY, hence they will remain in tube 2 and will enter other "legal" splicings, when they are axioms, or they will enter splicings producing "useless" strings ZyY.

Therefore, after a series of splicings of type 1, eventually in tube 2 a splicing of type 2 will be performed, producing strings of the form $X_1w_1Y_\alpha$ and $Z\alpha Y$.

The second string behaves exactly as we discussed above for the string ZuY. If a string XwY_α enters a new splicing in tube 2, this can only be a splicing of type 3, i. e.

$$(Z' \mid Z, XB \mid w_2Y_\alpha) \vdash_3 (Z'w_2Y_\alpha, XBZ),$$

for $w_1 = Bw_2$. The string $Z'w_2Y_\alpha$ cannot enter new splicings in tube 2 and cannot be transmitted to another tube. The case of XBZ will be discussed below.

Any string Xw_1Y_α is moved from tube 2 to tube 3, where we have to perform

$$(X'\alpha \mid Z, X \mid w_1Y_\alpha) \vdash (X'\alpha w_1Y_\alpha, XZ).$$

The second string, XZ, remains in tube 3 and it well enter only splicings of the form

$$(X'\beta \mid Z, X \mid Z) \vdash (X'\beta Z, XZ),$$

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

$$(X'\alpha w_1 \mid Y_\alpha, Z \mid Y) \vdash (X'\alpha w_1Y, ZY_\alpha).$$

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

$$(X \mid Z, X' \mid \alpha w_1Y) \vdash (X\alpha w_1Y, X'Z).$$

The second string remains in tube 5, and it produces nothing new, the first one has to be communicated to tube 2. Having started with the string $Xw_1\alpha Y$ in tube 2 we have returned to tube 2 with the string $X\alpha w_1Y$. 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 enter circular permutations as long as we want them to do that. This allows us to pass from a string Xw_1Bw_2Y to any string $Xw_1'Bw_2'Y$ such that $w_2w_1 = w_2'w_1'$. In this way, we can "rewind" the string until its suffix is the left-hand member of any rule in P we want to simulate by a rule in R_2 of the form $\#uY\$Z\#vY$. As the symbol B is always present (and exactly one copy of it is present as long as we do not use the rule $Z'\#Z\$XB$ in R_2), in every moment we know where the "actual beginning" of the string is placed. Consequently, using splicings of type 1 and the rewind technique made possible by the passing through the tubes 2, 3, 4, 5 as described above we can simulate every derivation in G. Conversely, exactly strings of the form Xw_1Bw_2Y can be obtained in this way, they correspond to strings w_2w_1 that are sentential forms of the grammar G.

Now consider the splicing of type 3 in tube 2. Let us return to the case of XBZ being in tube 2. If the string XBZ is used in further splicings, these are of the form

$$(Z' \mid Z, XB \mid Z) \vdash (Z'Z, XBZ)\,,$$

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, and here we have only one possibility, i. e.

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

If ZZY will enter new splicings, these are of the forms

$$\begin{aligned}(Z'x \mid Y, ZZ \mid Y) &\vdash (Z'xY, ZZY)\,,\\ (ZZ \mid Y, ZZ \mid Y) &\vdash (ZZY, ZZY)\,,\end{aligned}$$

hence no new string is obtained.

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

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

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:

$$\begin{aligned}&(Z'w_2 \mid uY, Z \mid vY) \vdash_1 (Z'w_2vY, ZuY)\,, \text{ for } u \to v \in P,\ w_1 = w_2u,\\ &(Z'w_2 \mid \alpha Y, Z \mid Y_\alpha) \vdash_2 (Z'w_2Y_\alpha, Z\alpha Y)\,, \text{ for } \alpha \in U,\ w_1 = w_2\alpha.\end{aligned}$$

The behaviour of ZuY, $Z\alpha Y$, $Z'w_2Y_\alpha$ is known, similar strings appeared in the previous discussion. The string $Z'w_2vY$ can be obtained by performing first

$$(XBw_2 \mid uY, Z \mid vY) \vdash_1 (XBw_2vY, ZuY)$$

and then

$$(Z' \mid Z, XB \mid w_2 v Y) \vdash_3 (Z' w_2 v Y, XBZ),$$

hence also this string is a "legal" one.

No parasitic string can reach the first tube, consequently $L(\Gamma) = L(G)$. $\square$

Examining the construction of the TT system Γ in the proof above, 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 Γ will be a universal TT system.* This suggests the following definition:

Definition 9. A universal TT system for a given alphabet T is a construct

$$\Gamma_U = (V_U, (A_{1,U}, R_{1,U}, V_{1,U}), ..., (A_{n,U}, R_{n,U}, V_{n,U})),$$

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 Γ, then there is a set $A_\Gamma \subseteq V^*$ such that the system

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

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

Stated in another way, encoding Γ as new axioms to be added to the i-th component of Γ_U, we obtain a system equivalent with Γ.

Theorem 10. *For every given alphabet T, there are universal TT systems of degree 7 and of type (FIN, FIN).*

Proof. Start the construction of the system Γ in the proof of Theorem 9 above from a universal type-0 grammar (for instance, as constructed in [3]). This grammar has the form $G_U = (\{X_1, X_2\}, T, -, P_U)$, hence it contains only two nonterminals (and a fixed set of productions). Therefore, for T being given, the alphabet V of Γ is fixed:

$$V \;=\; T \cup \{X_1, X_2, X, X', Y, Z, B\} \cup \{Y_\alpha \mid \alpha \in \{X_1, X_2, B\} \cup T\}.$$

In a similar way, all other components of Γ are fixed. Denote the obtained system by Γ_U. As G_U has no axiom, the axiom $XBSY$ of the second component of Γ_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 Γ_0, in view of the Turing/Church thesis 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 $A_{\Gamma_0} = \{XBw(G_0)Y\}$. We obtain a system Γ'_U such that $L(\Gamma'_U) = L(G_0)$. From the construction in the previous proof we have $L(G'_U) = L(\Gamma'_U)$. As G_0 is equivalent with the arbitrarily given TT system Γ, we have $L(\Gamma'_U) = L(\Gamma)$. This proves that Γ_U is universal, indeed. $\square$

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

8 Concluding remarks

The fact that the splicing operation is very powerful (as a formal operation on strings and languages) has been proved in various places. The usual way to do this is to characterize the family of recursively enumerable languages using the splicing operation and other "weak" prerequisites (other operations, special forms of splicing rules [17], [19], or additional languages such as Dyck languages, palindrom languages etc. [24]). Our results in Sections 3, 5, and 7 are the strongest possible of this type, because we only use the splicing operation, the intersection with a language of the form T^* (which according to [19], cannot be avoided), and systems with finite sets of axioms and finite sets of splicing rules. While it is true that we use the additional control mechanism of multiplicity counting or permitting respectively forbidden contexts, respectively the distributed mode of working in TT systems, such features are essential and cannot be removed; indeed, in view of [6] and [21], ordinary finite splicing systems can produce regular languages only.

However, as we have already pointed out, the most significant of the results we obtained is the existence of universal H systems of various types. This theoretically proves the feasibility of designing universal and programmable DNA computers, where a program consists of a single string to be added to the axiom set of the universal computer. In the particular case of mH systems, these program axioms have multiplicity one, while an unbounded number of copies of all the other axioms is available. The "fixed" axioms of the computer can be interpreted as a sort of non-erasable stored information available for free (i. e. a read-only memory).

As a closing remark, note that the proofs of Theorems 7, 8, and 10 rely on one hand on the specified constructions and on the other hand on the existence of universal type-0 grammars respectively of universal Turing machines and on the possibility of commuting from an H system to a type-0 grammar respectively to a Turing machine and conversely. This reduces the problem of the existence of universal H systems to the existence of universal type-0 grammars respectively of universal Turing machines. However, this quite indirect way, while theoretically useful, is inconvenient from a practical point of view. The open problem that remains is the effective construction of an universal H system that is as simple as possible. As the task seems to be a difficult one, it is perhaps better to look for a construction which meets at the same time the practical requirements raised by a possible implementation of such an universal H system. In short, we leave this task to a joint team of language theorists and practitioners of DNA computing.

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

[5] E. Csuhaj-Varju, L. Kari, Gh. Păun, Test tube distributed systems based on splicing, manuscript, 1995.

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

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

[8] L. Davis, *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, 1991.

[9] K. L. Denninghoff, R. W. Gatterdam, On the undecidability of splicing systems, *Intern. J. Computer Math.*, 27 (1989), 133 – 145.

[10] S. Eilenberg, *Automata, Languages and Machines, A*, Academic Press, New York, 1974.

[11] R. Freund, L. Kari, Gh. Păun, DNA computing based on splicing: The existence of universal computers. T. Report 185-2/FR-2/95, TU Wien, Institute for Computer Languages, 1995.

[12] D. K. Gifford, On the path to computation with DNA, *Science*, 226 (Nov. 1994), 993 – 994.

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

[14] 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.

[15] J. Hertz, A. Krogh, R. G. Palmer, *Introduction to the Theory of Neural Computation*, Addison-Wesley, Reading, Mass., 1991.

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

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

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

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

[20] Gh. Păun, L. Sântean, Parallel communicating grammar systems: the regular case. Ann. Buch. Univ., Series in Mathem. Inform. 38 (1989), 55 - 63.

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

[22] A. M. Turing, On computable numbers, with an application to the Entscheidungsproblem, *Proc. London Math. Soc.*, Ser. 2, 42 (1936), 230 – 265.

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

[24] T. Yokomori, S. Kobayashi, DNA evolutionary linguistics and RNA structure modelling: a computational approach, *Proc. 1st Intern. Symp. on Intelligence in Neural and Biological Systems*, IEEE, Herndon, 1995, 38 – 45.