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 Wien
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 specific variants of H systems (e. g. with regular sets of rules) and proves that we can construct universal computers based on various types of H systems with a finite set of splicing rules as well as a finite set of axioms, i. e. we show the theoretical possibility to design programmable universal DNA computers based on the splicing operation. For H systems working in the multiset style (where the numbers of copies of all available strings are counted) we elaborate how a Turing machine computing a partial recursive function can be simulated by an equivalent H system computing the same function; in that way, from a universal Turing machine we obtain a universal H system.

Considering H systems as language generating devices we have to add various simple control mechanisms (checking the presence/absence of certain symbols in the spliced strings) to systems with a finite set of splicing rules as well as with a finite set of axioms in order to obtain the full computational power, i. e. to get a characterization of the family of recursively enumerable languages. We also introduce test tube systems, where several H systems work in parallel

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

[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.

in their tubes and from time to time the contents of each tube are redistributed to all tubes according to certain separation conditions. By the construction of universal test tube systems we show that also such systems could serve as the theoretical basis for the development of biological (DNA) computers.

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

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.* One 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], yet actually the universality of Adleman's way for computing using DNA still seems to be not settled theoretically in a satisfactory way.

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 started with [8], where the operation of splicing was introduced as a model for this phenomenon. One of the important results in the area of H systems states that H systems with finite sets of axioms and finite sets of splicing rules can generate only regular languages [11]. However, if we use a regular set of splicing rules of a very particular type, a maximal increase of the generative power is obtained: such H systems characterize the family of recursively enumerable languages [10]. Working with (infinite) *regular sets* of splicing rules is natural from a mathematical point of view, but unrealistic from a practical point of view. *How to obtain H systems with both the set of axioms and the set of splicing rules being finite, but still being able to reach the full power of Turing machines?* In view of the results in [11], we have to pay the reduction of the sets of rules from being regular to being finite: One way (which is well-known from formal language theory [5]) to do this is to regulate the use of the splicing rules by suitable control mechanisms. This idea has been explored in [7], 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 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. In [7] 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. 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 arbitrarily 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.

A new idea is to consider a "parallel communicating" architecture as in grammar system [3]: 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], [9]: the contents of a tube are redistributed to all tubes according to certain specified "separation conditions". Again we obtain a characterization of the recursively enumerable languages.

From the existence of universal Turing machines [12] and from the proofs of all the results mentioned above for different types of H systems and for each of these types we obtain a way to construct a universal H system of the corresponding type. This can be interpreted as a proof for the (theoretical) possibility to construct universal 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, $V^+ = V^* - \{\lambda\}$, FIN, REG, RE are the families of finite, regular, and recursively enumerable languages, respectively. For general formal language theory prerequisites we refer to [13], for regulated rewriting to [5], and for grammar systems to [3].

Definition 1. A *splicing scheme* (or *H scheme*) is a pair $\sigma = (V, R)$, where V is an alphabet and $R \subseteq V^* \{\#\} V^* \{\$\} V^* \{\#\} V^*$; $\#, \$$ are special symbols not in V. (V is the *alphabet* of σ and R is the set of *splicing rules*). For $x, y, z, w \in V^*$ and $r = u_1\#u_2\$u_3\#u_4$ in R we define $(x, y) \vdash_r (z, w)$ if and only if $x = x_1u_1u_2x_2,\ y = y_1u_3u_4y_2$, and $z = x_1u_1u_4y_2,\ w = y_1u_3u_2x_2$ for some $x_1, x_2, y_1, y_2 \in V^*$. For a splicing scheme $\sigma = (V, R)$ 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 $\sigma^0(L) = L$ and $\sigma^{i+1}(L) = \sigma^i(L) \cup \sigma(\sigma^i(L))$ for all $i \geq 0$. $\square$

Definition 2. An *H system* is a pair $\gamma = (\sigma, A)$, where $\sigma = (V, R)$ is a splicing scheme and $A \subseteq V^*$ is the set of axioms. An *extended H system* is a quadruple $\rho = (V, T, R, A)$, where $\gamma = (V, R, A)$ is the underlying H system and $T \subseteq V$ is the terminal alphabet. The *language generated* by the extended H system ρ is defined by $L(\rho) = \sigma^*(A) \cap T^*$.

For two families of languages F_1, F_2, an extended H system $\rho = (V, T, R, A)$ with $A \in F_1$ and $R \in F_2$ is said to be of *type* (F_1, F_2), and we denote the family of languages generated by extended H systems of that type by $EH(F_1, F_2)$. $\square$

In the definitions above, after splicing two strings x, y and obtaining two strings z and w, we may again use x or y (they are not "consumed" by splicing) as a term of a splicing, possibly the second one being z or w; moreover, also the new strings are supposed to appear in infinitely many copies. Probably more realistic is the assumption that at least a part of the strings is available in a limited number of copies. This leads us to consider *multisets*, i. e. sets with multiplicities associated to their elements.

In the style of [6], 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 $\rho = (V, T, R, A)$, where V, T, R are as in an extended H system (Definition 2) and A is a multiset over V^*.

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

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

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

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

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

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

For two families of languages F_1, F_2, an extended mH system $\rho = (V, T, R, A)$ is said to be of type (mF_1, F_2) if $supp(A) \in F_1$ and $R \in F_2$; we denote the family of languages generated by such extended mH systems of type (mF_1, F_2) by $EH(mF_1, F_2)$. □

An (extended) H system as in Definition 2 can be interpreted as an mH system working with multisets of the form $M(x) = \infty$ for all x such that $M(x) \neq 0$.

3 Universal computing with mH systems

In this section 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. We will assume that the H system $\Gamma = (V, R, A)$ "really" starts to work only if one additional single string is added (to A). 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^*).
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 they all 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 $(Q, q_0, q_f, V, T, Z_0, B, \delta)$, where Q is the (finite) set of states, q_0 is the initial state, q_f is the final state, V is the (finite) alphabet of tape symbols, $T \subseteq V$ 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, $\delta : Q \times V \to Q \times V \times \{L, R\}$ 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. $X \neq Z_0$;
- $(q, Z_0, p, Y, D) \in \delta$, $D \in \{L, R\}$, implies $Y = Z_0$ and $D = R$;
- $(q, X, p, Z_0, R) \in \delta$ implies $X = Z_0$;
- $(q, X, p, Y, D) \in \delta$, $D \in \{L, R\}$, implies $q \neq q_f$, whereas
- for all $q \in Q - \{q_f\}$ and all $X \in V$ there is some (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 $(q, X, p, Y, L) \in \delta$ yields $Z_0u'pYvB^\omega$ from $Z_0u'XqvB^\omega$.
- Applying $(q, X, p, Y, R) \in \delta$ yields $Z_0u'YUpv'B^\omega$ from $Z_0u'XqUv'B^\omega$.

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 $Z_0wq_0B^\omega$ computes $f(w)$ by ending up in the final configuration $Z_0f(w)q_fB^\omega$, 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 $Z_0wq_0B^\omega$ 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 $Z_0C(M)c_3wq_0B^\omega$, $w \in T^*$, 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 configuration is $Z_0f_M(w)q_fB^\omega$.

Theorem 1. *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: $\Gamma_f = (V', A, R)$ 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

$$\begin{aligned}
A_1 \; = \; & \{Z_0 U p Z_2 \mid \text{for some } q \in Q \;\; (q, Z_0, p, Z_0, R) \in \delta,\; U \in V_0,\; p \in Q\} \cup \\
& \{(r)\, Y U p\, [r] \mid r = (q, X, p, Y, R) \in \delta,\; U, X, Y \in V_0,\; p, q \in Q\} \cup \\
& \{(r)\, p Y\, [r] \mid r = (q, X, p, Y, L) \in \delta,\; X, Y \in V_0,\; p, q \in Q\} \cup \\
& \{Z_3 q B Z_1 \mid q \in Q - \{q_f\}\} \cup \{Z_3 q_f Z_2\}, \\
A_2 \; = \; & \{Z_0 q U Z_2 \mid \text{for some } q \in Q \;\; (q, Z_0, p, Z_0, R) \in \delta,\; U \in V_0,\; p \in Q\} \cup \\
& \{(r)\, X q U\, [r] \mid r = (q, X, p, Y, R) \in \delta,\; U, X, Y \in V_0,\; p, q \in Q\} \cup \\
& \{(r)\, X q\, [r] \mid r = (q, X, p, Y, L) \in \delta,\; X, Y \in V_0,\; p, q \in Q\} \cup \\
& \{Z_3 q Z_1 \mid q \in Q - \{q_f\}\} \cup \{Z_3 q_f B Z_2\}.
\end{aligned}$$

The set R contains the following splicing rules:

1. $Z_0 q U \# C \$ Z_0 U p \# Z_2$ 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. $D X q U \# C \$ (r) Y U p \# [r]$ 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. $D \# X q U [r] \$ (r) \# Y U p C$ 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. $D X q \# C \$ (r) p Y \# [r]$ 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. $D \# X q [r] \$ (r) \# p Y C$ 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. $U \# q Z_1 \$ Z_3 \# q B Z_1$ for $q \in Q - \{q_f\}$ and $U \in V$;
7. $D \# q_f B \$ Z_3 \# q_f Z_2$ for $D \in V$;
8. $D q_f \# Z_2 \$ Z_3 q_f B \# C$ for $D \in V,\; C \in \{B, Z_1\}$;
9. $w \# \$ \# w$ for all $w \in A$.

Any configuration $Z_0 u q v B^\omega$ in M is represented by some finite string $Z_0 u q v B^m Z_1$, for some $m \geq 0$, in the H system Γ_f. The only terminating string not in A obtained by using the rules from R from an initial string $Z_0wq_0Z_1$ 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_fZ_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_fZ_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 regular "filter" $\{Z_0\}T^*\{q_fZ_1\}$.

The proof of the preceding theorem could also be extended in such a way that from the final configuration $Z_0f(w)q_fZ_1$ we could obtain $f(w)$ itself as the final string that cannot be processed any more.

The following result proving the universality of H systems (with multisets) with respect to computability is based on the existence of universal Turing machines [12] and can be proved by using similar techniques as in the proof of the previous theorem:

Theorem 2. *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 $Z_0C(M_f)c_3wq_0Z_1$ 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 $Z_0f(w)q_fZ_1$, 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 1.

4 The generative power of H systems

From the results proved in [11] and in [10] we can deduce

Theorem 3.

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

   for all F with $FIN \subseteq F \subseteq REG$;

2. $EH(FIN, REG) = EH(F_1, F_2) = RE$

for all F_1, F_2 *with* $FIN \subseteq F_1 \subseteq RE$, $REG \subseteq F_2 \subseteq RE$.

Theorem 4. [7] $EH(mFIN, FIN) = EH(mF_1, F_2) = RE$, *for all families* F_1 and F_2 *such that* $FIN \subseteq F_1 \subseteq RE$ *and* $FIN \subseteq F_2 \subseteq RE$.

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) respectively when they are not present (forbidden contexts). These forbidden contexts can be interpreted as *inhibitors* of the associated rules (and they can be checked, manually, as in [1]).

Definition 4. An extended H system *with permitting contexts* is a quadruple $\rho = (V, T, R, A)$, where V, T, A are as in Definition 2 and R is a set of triples (we call them rules with permitting contexts) of the form $p = (r;\ C_1, C_2)$ 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 $p \in R$ 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 ρ is defined in the natural way, and the family of languages $L(\rho)$, for $\rho = (V, T, R, A)$ 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 $EH(F_1, cF_2)$. □

Definition 5. An extended H system *with forbidden contexts* is a quadruple $\rho = (V, T, R, A)$, where V, T, A are as in Definition 2 and R is a set of triples (we call them rules with forbidden contexts) of the form $p = (r;\ D_1, D_2)$ 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 ρ is defined in the natural way, and the family of languages $L(\rho)$, for $\rho = (V, T, R, A)$ 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 $EH(F_1, fF_2)$. □

Theorem 5. [7] $EH(FIN, rFIN) = EH(F_1, rF_2) = RE$, *for all families* F_1, F_2 *such that* $FIN \subseteq F_1 \subseteq RE$, $FIN \subseteq F_2 \subseteq RE$, and for each $r \in \{c, f\}$.

The results stated above in this section prove that the considered types of H systems with a finite number of rules as well as a finite number of axioms are already computationally complete. Yet in order to prove that *programmable* computers based on splicing can be constructed, it is necessary to find *universal H systems*:

Definition 6. Given an alphabet T and two families of languages F_1, F_2, a construct $\rho_U = (V_U, T, R_U, A_U)$, 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 $\rho = (V, T, R, A)$ of type (F_1, F_2) there is a language A_ρ such that $A_U \cup A_\rho \in F_1$ and $L(\rho) = L(\rho'_U)$, where $\rho'_U = (V_U, T, R_U, A_U \cup A_\rho)$. The particularizations of this definition to mH systems or to H systems with permitting respectively forbidden contexts are obvious. □

Theorem 6. [7] *For every given alphabet T there exist mH systems of type $(mFIN, FIN)$, extended H systems of type (FIN, FIN) with permitting respectively forbidden contexts that are universal for the class of mH systems respectively the class of extended H systems with permitting respectively forbidden contexts with the terminal alphabet T.*

5 Test tube systems

In this section we investigate a new model for biological computers that incorporates basic ideas of parallel communicating grammar systems [4].

Definition 7. A *test tube* (TT for short) *scheme of degree n*, $n \geq 1$, is a construct $\Sigma = (V, (R_1, F_1), ..., (R_n, F_n))$, where V is an alphabet, $F_i \subseteq V^*$ and $R_i \subseteq V^* \{\#\} V^* \{\$\} V^* \{\#\} V^*$, for each i, $1 \leq i \leq n$.

Each pair (R_i, F_i) is called a *component* of the scheme, or a *tube*; R_i is the set of splicing rules of the tube i, F_i is the *selector* of the tube i. The pair $\sigma_i = (V, R_i)$ is the *underlying H scheme* associated with the component i of the system. For arbitrary n-tuples $(L_1, ..., L_n)$, $(L'_1, ..., L'_n)$, $L_i, L'_i \subseteq V^*$, $1 \leq i \leq n$, we define $\Sigma((L_1, ..., L_n)) = (L'_1, ..., L'_n)$ if and only if $L'_i = \bigcup_{j=1}^{n} \left(\sigma_j^* (L_j) \cap F_i\right)$ for all i, $1 \leq i \leq n$. □

In words, the contents of each tube are 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 $F_1, ..., F_n$. When a string belongs to several languages F_i, then copies of it will be distributed to all tubes i with this property.

Definition 8. A *TT system of degree n*, $n \geq 1$, is a construct

$$\Gamma = (V, (R_1, F_1), ..., (R_n, F_n), (A_1, ..., A_n)),$$

where $\Sigma = (V, (R_1, F_1), ..., (R_n, F_n))$ is the underlying TT scheme and A_i, $1 \leq i \leq n$, is the set of axioms of the i-th component (i. e. of tube i). Considering the iterated application of the scheme Σ to the initial configuration$(A_1, ..., A_n)$, i. e. $\Sigma^k((A_1, ..., A_n))$, by convention we obtain the *language generated* by a TT system Γ as the result of all computations in tube 1, i. e.

$$L(\Gamma) = \{w \in V^* \mid w \in L_1 \text{ for some } (L_1, ..., L_n), \text{ with } (L_1, ..., L_n) = \Sigma^k((A_1, ..., A_n)) \text{ for some } k \geq 0\Big\}.$$

$\square$

In the following we will restrict ourselves to very special selectors F_i, i. e. of the form V_i^*, where the $V_i \subseteq V$, $1 \leq i \leq n$. In this sense, selecting the results in the first tube corresponds to the intersection with a terminal alphabet T in extended H systems by choosing $V_1 = T$.

Definition 9. Given two families of languages, F_1 and F_2, a TT system $\Gamma = (V, (R_1, V_1^*), ..., (R_m, V_m^*), (A_1, ..., A_n))$ with $A_i \in F_1$ and $R_i \in F_2$, for all i, $1 \leq i \leq m$, is said to be of type (F_1, F_2). The family of all languages generated by such TT systems is denoted by $TT_*(F_1, F_2)$. $\square$

Theorem 7. [4] $TT_*(FIN, FIN) = TT_*(F_1, F_2) = RE$ *for all families* F_1 and F_2 *such that* $FIN \subseteq F_1 \subseteq RE$ *and* $FIN \subseteq F_2 \subseteq RE$.

Definition 10. A universal TT system for a given alphabet T is a construct

$$\Gamma_U = \left(V_U, (R_{1,U}, T^*), \left(R_{2,U}, V_{2,U}^*\right), ..., \left(R_{n,U}, V_{n,U}^*\right), (A_{1,U}, A_{2,U}, ..., A_{n,U})\right)$$

with the following property: If we take an arbitrary TT system Γ, then there is a set $A_\Gamma \subseteq V_{2,U}^*$ such that $L(\Gamma_U') = L(\Gamma)$ for the TT system

$$\Gamma_U' = (V_U, (R_{1,U}, T^*), (R_{2,U}, V_{2,U}), ..., (R_{n,U}, V_{n,U}), (A_{1,U}, A_{2,U} \cup A_\Gamma, ..., A_{n,U})).$$

$\square$

Theorem 8. [4] *For every given alphabet* T*, there exists a universal TT system of type* (FIN, FIN).

6 Concluding remarks

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 programmable universal 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.

Although in this paper we have shown the (theoretical) possibility how to obtain universal biological computers based on the splicing operation, many questions remain open. For instance, from a theoretical point of view, an interesting problem is to determine the minimal number of test tubes needed for a universal test tube system with respect to a given alphabet. From a practical point of view, the real implementation of such test tube systems or other universal H systems introduced in the preceding sections is a most challenging task that has to be done in a joint team of theorists, biologists and practitioners of DNA computing; the presentation of this paper should also be seen as a starting point for discussions on such topics between formal language theorists and collegues working in other fields.

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

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

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

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

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

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

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

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

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

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

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