DNA Computing Based on Splicing: The Existence of Universal Computers[1]

Rudolf FREUND
Technical University Wien, Institute for Computer Languages
Resselgasse 3, 1040 Wien, Austria

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

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

Abstract. Splicing systems are generative mechanisms based on the splicing operation introduced by Tom Head as a model of DNA recombination. We prove that the generative power of finite extended splicing systems equals that of Turing machines, provided we consider multisets or provided a control mechanism is added. We also show that there exist universal splicing systems with the properties above, i.e. there exists a universal splicing system with fixed components which can simulate the behaviour of any given splicing system, when an encoding of the particular splicing system is added to its set of axioms. In this way the possibility of designing programmable DNA computers based on the splicing operation is proved.

Categories and Subject Descriptors: F 1.1 [**Computations by Abstract Devices**]: Models of Computation, F 4.2 [**Mathematical Logic and Formal Languages**]: Grammars and Other Rewriting Systems

General Terms: Theory, DNA Computing

Additional Key Words and Phrases: DNA recombination, splicing operation, H systems, Chomsky grammars, Turing machines, random context grammars, universal grammars, universal Turing machines

---

[1]Research supported by the Academy of Finland, project 11281, and grants OGP0007877 and OGP0000243 of the National Science and Engineering Research Council of Canada.

[2]All correspondence to Gheorghe Păun. Current address: Turku Centre for Computer Science TUCS, Data City, Lemminkäisenkatu 14 A, 4th floor, 20520 Turku, Finland; email: paun@sara.utu.fi

1. Introduction

The past years have witnessed an increased interest in developing new types of algorithms and designing new types of computers which differ from the classical Turing/von Neumann notions of algorithms and computers in a fundamental way. Special attention has been paid to biological-like computing, as illustrated by the well developed area of neural computation and that of genetic algorithms ([Davis, 1991], [Hertz et al., 1991], [Koza, Rice, 1992], etc.)

A rather new area is that of DNA computing, which is based on the observation that the incredibly complex structure a living being is results from applying a few simple operations (local mutations, copying and splicing/recombination/crossing over) to an initial DNA sequence. The complexity of the output implies that these operations are powerful and of a very fundamental nature.

These observations, together with the feasibility of DNA synthesis and of manipulating it in lab conditions, suggest the idea that any complex computation can be carried out by starting from an initial information, encoded in a DNA-like sequence, by performing the operations mentioned above.

And, indeed, at the end of 1994, this idea started to be implemented: [Adleman, 1994] reports the way of solving (a small instance of) the Hamiltonian path problem in a test tube, just by handling DNA strings. A series of enthusiastic (yet also less enthusiastic) reactions followed, see [Gifford, 1994], [Lipton, 1994], [Lipton, 1995], etc. ([Lipton, 1995] describes the way how to solve the satisfiability problem by DNA computing). At the beginning of 1995, [Adleman, 1995] has developed the idea, proposing a formal framework (a sort of test tube programming language) for encoding molecular algorithms.

Yet still two fundamental problems have remained open (or at least as not answered in a satisfying manner): (1) Are the classes of DNA algorithms *computationally complete*, in the sense that *every* algorithm (hence every Turing machine or every equivalent mechanism) can be encoded as a DNA algorithm in a specified class ? (2) Are there, at least theoretically, *universal and programmable DNA computers*, i.e. given "test tube computers" that are able to run any arbitrarily given program/algorithm ?

These problems are already formulated in the papers quoted above: "It seems likely that a single molecule of DNA can be used to encode the instantaneous description of a Turing machine and that currently available protocols and enzymes could (at least under idealized conditions) be used to induce successive sequence modifications, which would correspond to the execution of the machine", [Adleman, 1994]. "If we were able to construct a universal machine out of biological macromolecular components, then we could perform any computation by means of biological techniques. There are certainly powerful practical motivations for this approach, including the information-encoding density offered by macromolecules and the high energy efficiency of enzyme systems. At present, there is no known way of creating a synthetic universal system based on macromolecules. 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", [Gifford, 1994].

It is somewhat intriguing why the problems mentioned above were not investigated for the "programming language" in [Adleman, 1995] (this is also true for genetic algorithms, [Goldberg, 1989]).

In this paper we investigate a class of (generative) mechanisms based on an operation specific to DNA recombination, i.e. the operation of *splicing*. For various types of such mechanisms we affirmatively solve both the problems mentioned above: (1) these mechanisms are computationally complete, and (2) there are universal mechanisms for each class we consider.

The fundamental notion we investigate here is that of splicing. (A word of warning: the biologists use this term also for a quite different operation on RNA sequences. The operation we consider here is, in fact, a less restrictive cross-overing as used in genetic algorithms: we allow the splicing of strings of different length at arbitrary, possibly different, positions.)

As formalized in [Head, 1987], 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 $x = x_1u_1u_2x_2$, $y = y_1u_3u_4y_2$ and $z = x_1u_1u_4y_2$, $w = y_1u_3u_2x_2$; 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 [Păun, **a**], [Păun, **b**], [Păun, **c**], and [Păun et al.] 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 [Păun et al.].

The H systems, extended or not, turned out to be very powerful generative mechanisms. Several characterizations of recursively enumerable languages in terms of various types of H systems were obtained in [Păun, **b**] and [Păun et al.]. Related results appear in [Yokomori, Kobayaski, 1995], while [Denninghoff, Gatterdam, 1989] encodes the range of Turing machines using iterated splicing on multisets (sets with multiplicities associated to their elements).

We improve these results here by obtaining the strongest characterization of recursively enumerable languages possible in this framework, i. e. the family of recursively enumerable languages is generated by H systems using finite sets of axioms and finite sets of splicing rules, supplemented with a certain control mechanism. Such a control mechanism consists of keeping track of the multiplicities of strings (the number of available copies of a string at each moment), respectively of checking the presence of certain substrings in the strings to be spliced.

This is the strongest result that can be obtained since, as shown in [Culik, Harju, 1991], [Pixton, 1995], if no control is used, the H systems using finite sets of axioms and finite sets of rules generate only the family of regular languages. Our results can be reformulated by saying that the computational power of H systems of the mentioned types equals the computational power of Turing machines. This is rather unexpected, taking into account that the nature of the splicing operation is quite different from the nature of the operations involved in the work of a Turing machine.

Still more suprising is the fact that not only are the H systems computationally complete, but, as shown in this paper, they can also be executed as programs of a *universal H system.* By a universal H system we mean a system with fixed sets of auxiliary symbols, of axioms and of splicing rules that can behave like any given H system γ if we add an *encoding* of γ as an additional axiom of the universal system. We will prove the existence of such an universal H system with multiplicities, respectively with checking the occurrence of certain substrings.

The interpretation of the results we obtain, from the point of view of genetic/molecular computing is that, theoretically, there exist *universally programmable DNA computers* which are based on the splicing operation. The only operations used in these computers are the iterated splicing (the splicing known from DNA recombination) and the squeezing operation (which in formal language terms amounts to the intersection with a regular set of the form T^*, with T a given alphabet). Therefore, for the case of using the splicing operation as a basic operation on DNA sequences, we (affirmatively) answer the problems formulated in [Adleman, 1994] and [Gifford, 1994].

In an optimistic way, one can think of an analogy between these results and the existence of universal Turing machines proved in [Turing, 1936], which has laid the theoretical foundation for the design of electronic computers. In a similar fashion, we claim that DNA computers, programmable and universal, are possible. Will they become "personal computers", in say, 50 years, as it happened with electronic computers ? Maybe yes, maybe no. Anyway, here we do not discuss the feasibility of actually building such computers, as this problem goes far beyond mathematics, stepping into practical DNA engineering.

2. Extended 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. A Chomsky grammar will be written as $G = (N, T, S, P)$, where N is the nonterminal alphabet, T is the terminal alphabet, $S \in N$ is the axiom, and P is the set of rewriting rules, presented in the form $u \rightarrow v$.

For general formal language theory prerequisites we refer to [Harrison, 1978], [Salomaa, 1973], and for regulated rewriting to [Dassow, Păun, 1989].

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

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

□

The strings x, y are called the *terms* of the splicing. When r is understood, we write $\vdash$ insted of $\vdash_r$.

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

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

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.)
$\square$

Examples. We shall illustrate the definitions above with two examples. The first one is simpler (it is, in fact, a reformulation of an example in [Mateescu et al., 1995]); take

$$\gamma_1 = (\{a, b, c\}, \{a, b, c\}, \{abaca, acaba\}, R),$$

with R containing two splicing rules,

$$r_b = b\#\$b\#, \quad r_c = c\#\$c\#.$$

For instance, we can perform

$$(abac|a, ac|aba) \vdash_{r_c} (abacaba, aca),$$
$$(abacab|a, ab|aca) \vdash_{r_b} (abacabaca, aba),$$

or, in general, for $n \geq 1$,

$$((abac)^n|a, ac|aba) \vdash_{r_c} ((abac)^n aba, aca),$$
$$((abac)^n ab|a, ab|aca) \vdash_{r_b} ((abac)^{n+1} a, aba).$$

In each term of these four splicings, we have indicated by a vertical bar, $|$, the position where the cutting of terms takes place. For an easier understanding of the splicing operations, we shall use this notation in all constructions below.

Similarly,

$$((acab)^n|a, ab|aca) \vdash_{r_b} ((acab)^n aca, aba),$$
$$((acab)^n ac|a, ac|aba) \vdash_{r_c} ((acab)^{n+1} a, aca).$$

Therefore,

$$(abac)^+ a \cup (abac)^* aba \cup (acab)^+ a \cup (acab)^* aca \subseteq L(\gamma_1).$$

(The extension plays no role in this case, γ_1 is of the form (T, T, A, R).)

The reader can verify that also the converse inclusion holds (hint: when splicing two strings of one of the forms $(abac)^n a, (acab)^n a$ – initially we have $n = 1$ – or of the forms $(abac)^n aba, (acab)^n aca$, we identify either a substring

aba or a substring aca of them, hence the obtained strings are of the same forms).

Consider now a more complex example, namely an extended H system of type (FIN, REG) generating the language $\{a^{2^n} \mid n \geq 0\}$. Besides the aim of familiarizing the reader with the way the splicing operation and the H systems work, this example also involves an idea used in the constructions in Section 4.

Consider the system

$$\gamma_2 = (V, \{a\}, A, R),$$

with

$$V = \{X, X', Y, Y', Y'', B, a\},$$
$$A = \{XBaY,\ X'aY',\ X'Y''\},$$

and R containing the following groups of rules:

$$\begin{array}{lll}
1. & Xa^nBa^m\#aY\$X'a\#Y', & \text{for } n, m \geq 0, \\
2. & X'a^2\#Y\$X\#a^nBa^mY', & \text{for } n, m \geq 0, \\
3. & X'a^nBa^m\#Y'\$X\#Y, & \text{for } n, m \geq 0, \\
4. & X\#Y\$X'\#a^nBa^mY, & \text{for } n, m \geq 0, \\
5. & Xa^n\#BY\$X'\#Y'', & \text{for } n \geq 1, \\
6. & X'B\#Y\$X\#a^nY'', & \text{for } n \geq 1, \\
7. & X'Ba^n\#Y''\$X\#Y, & \text{for } n \geq 1, \\
8. & \#XY\$XB\#a^nY, & \text{for } n \geq 1, \\
9. & a^n\#Y\$XY\#, & \text{for } n \geq 1.
\end{array}$$

Here is the way of producing the string a^4:

$$\begin{array}{l}
(XB|aY, X'a|Y') \vdash_1 (XBY', X'a^2Y), \\
(X'a^2|Y, X|BY') \vdash_2 (X'a^2BY', XY), \\
(X'a^2B|Y', X|Y) \vdash (X'a^2BY, XY'), \\
(X|Y, X'|a^2BY) \vdash_4 (Xa^2BY, X'Y), \\
(Xa^2|BY, X'|Y'') \vdash_5 (Xa^2Y'', X'BY), \\
(X'B|Y, X|a^2Y'') \vdash_6 (X'Ba^2Y'', XY), \\
(X'Ba^2|Y'', X|Y) \vdash_7 (X'Ba^2Y, XY''), \\
(X|Y, X'|Ba^2Y) \vdash_4 (XBa^2Y, X'Y), \\
(XBa|aY, X'a|Y') \vdash_1 (XBaY', X'a^2Y), \\
(X'a^2|Y, X|BaY') \vdash_2 (X'a^2BaY', XY), \\
(X'a^2Ba|Y', X|Y) \vdash_3 (X'a^2BaY, XY'), \\
(X|Y, X'|a^2BaY) \vdash_4 (Xa^2BaY, X'Y), \\
(Xa^2B|aY, X'a|Y') \vdash_1 (Xa^2BY', X'a^2Y), \\
(X'a^2|Y, X|a^2BY') \vdash_2 (X'a^4BY', XY),
\end{array}$$

$$(X'a^4B|Y', X|Y) \vdash_3 (X'a^4BY, XY'),$$
$$(X|Y, X'|a^4BY) \vdash_4 (Xa^4BY, X'Y),$$
$$(Xa^4|BY, X'|Y'') \vdash_5 (Xa^4Y'', X'BY),$$
$$(X'B|Y, X|a^4Y'') \vdash_6 (X'Ba^4Y'', XY),$$
$$(X'Ba^4|Y'', X|Y) \vdash_7 (X'Ba^4Y, XY''),$$
$$(X|Y, X'|Ba^4Y) \vdash_4 (XBa^4Y, X'Y),$$
$$(|XY, XB|a^4Y) \vdash_8 (a^4Y, XBXY),$$
$$(a^4|Y, XY|) \vdash_9 (a^4, XYY).$$

One sees how rules in groups 1 – 4 cut a symbol a from the right-hand end of a string of the form $Xa^nBa^mY, n \geq 0, m \geq 1$, and add a^2 to the left-hand string a^n. The pairs of nonterminals $(X, Y), (X, Y'), (X', Y), (X', Y')$ precisely control these operations. Similarly, rules in groups 5 – 7, 4 cut a symbol B from the strings of the form Xa^nBY and move it to the left-hand end, producing the string XBa^nY. The process can be iterated. Between a moving of B from the right-hand end to the left-hand end and another moving of B, all occurrences of a are doubled. The symbol B can be removed only in the presence of both X and Y, namely, from strings of the form XBa^nY. This ensures the fact that n is of the form 2^i. Then also Y can be removed. (If Y is removed before removing B, then B can never be removed.) All rules in groups 1 – 7 can be used only in the presence of both X, Y, or of their primed variants.

Using these remarks, the reader can prove not only the rather easy inclusion $\{a^{2^n} \mid n \geq 0\} \subseteq L(\gamma_2)$, but also the converse one. $\square$

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 as a term of a splicing, possibly the second one being z or w, they are not "consumed" by splicing; 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 [Eilenberg, 1971], 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

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

At point (iii) we have operations with multisets. The writing above is meant to cover also the case when $x = y$ (then we must have $M_1(x) \geq 2$ and we must subtract 2 from $M_1(x)$), or $z = w$ (then we must add 2 to $M_2(z)$).

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

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

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

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

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

$\square$

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. The family of languages generated by mH systems $\gamma = (V, T, A, R)$ with $card(supp(A)) \leq k$, for given k, and $R \in F_2$ is denoted by $EH(m[k], F_2)$. In a similar way, by $EH(\omega[k], F_2)$ we denote the family of languages generated by ωH systems with at most k axioms.

Remark 1. As we have pointed out in the introduction, from a mathematical point of view, for a splicing rule $r = u_1 \# u_2 \$ u_3 \# u_4$ it is enough to define the ternary relation $(x, y) \vdash_r z$, for $x = x_1 u_1 u_2 x_2$, $y = y_1 u_3 u_4 y_2$, $z = x_1 u_1 u_4 y_2$. Considering also the string $w = y_1 u_3 u_2 x_2$ amounts to considering the *symmetric rule* $u_3 \# u_4 \$ u_1 \# u_2$, too. This is the style of many papers on splicing. We here work in the restricted set-up of Definition 1 also from "practical" reasons, in order not to introduce non-necessary assumptions which cannot be met in practice.

On the other hand, also a *reflexivity assumption* should be considered: together with $r = u_1\#u_2\$u_3\#u_4$, also the rules $u_1\#u_2\$u_1\#u_2$ and $u_3\#u_4\$u_3\#u_4$ are implicitly present (the restriction enzyme defining the cut at $u_1\#u_2$ acts on this site irrespective whether or not some other string containing the site $u_3\#u_4$ is present; moreover there might also be a lygase recombining the two ends u_1 and u_2). We shall not consider these additional rules in our constructions, but in remarks placed after the main proofs we shall check whether they would modify the generated languages.

Anyway, in order to implement constructions as those in the proofs in the subsequent sections, one has to face many other (perhaps vital) assumptions and practical restrictions. For instance, in the case of multisets, we here assume that one can precisely count the number of copies of each string, that we can distinguish between having n copies and having $n+1$ copies (in particular, for $n = 1$). With the techniques we know today, this seems to be a very difficult task. We have to face these problems and other related ones when attempting to actually build a "test tube"-computer. (Or, maybe, the notion of a multiset can be replaced with a less restrictive one, of a statistical nature, or like in fuzzy sets theory.)

On the other hand, it is worth noting that the intersection with T^*, which is essential for defining the language generated by an extended H system, can already be practically realized. This is exactly the way how the description of the Hamiltonian path is selected in [Adleman, 1994]. (In fact, the practical procedure consists of removing from the test tube all the sequences containing a given subword; after the removing of all sequences containing an auxiliary symbol, codified in a specific way, the set of strings over T is exactly what remains in the test-tube.) □

3. Computational completeness of mH systems

In [Păun, **b**] it is proved that every recursively enumerable language L can be written as

$$L = h(L_1 \cap L_2),$$

where h is a restricted morphism, $L_1 \in H(mFIN, FIN)$ and $L_2 \in REG$.

We improve this result to the following much stronger form:

Theorem 1. *$REG = EH(m[1], FIN) \subset EH(m[2], FIN) = EH(mF_1, F_2) = RE$, for all families F_1, F_2 such that $FIN \subseteq F_1 \subseteq RE$, $FIN \subseteq F_2 \subseteq RE$.*

To stress the significance of this result, let us reformulate (part of) it in different terms: every Turing machine can be simulated by an extended mH system with a finite set of rules; two axioms are enough (and necessary: systems with one axiom only generate regular languages).

We separate the proof of this theorem into a series of lemmas. The first one establishes the most important (and most unexpected) of the relations stated above.

Lemma 1. $RE \subseteq EH(mFIN, FIN)$.

Proof. Consider a type-0 Chomsky grammar $G = (N, T, S, P)$, with the rules in P of the form $u \rightarrow v$ 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 Lab; 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

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

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 5 and 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 $X_1^2zX_2^2$ into two strings $X_1^2z_1$, $z_2X_2^2$, each one with multiplicity one; the rules in groups 2 and 4, 6 bring together these strings, leading to a string of the form $X_1^2z'X_2^2$. 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 Lab, 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. Moreover, the discussion below implicitly shows how to simulate a terminal derivation in G by splicing operations in γ.)

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 $w_\alpha, w'_\alpha, \alpha \in U$). In the first step, we have to start with w_0, $w_0 = X_1^2YSX_2^2$, $A(w_0) = 1$. Now assume that we have a string $X_1^2w_1Yw_2X_2^2$ with multiplicity 1 (w_0 is of this form). 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 $X_1^2w_1Yuw_3X_2^2$ for some $r : u \rightarrow v \in P$. Using the axiom $(r)v[r]$ from A we obtain

$$(X_1^2w_1Yu|w_3X_2^2, (r)v|[r]) \vdash (X_1^2w_1Yu[r],\ (r)vw_3X_2^2).$$

No rule from the groups 1 and 3 – 8 can be applied to the obtained 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

$$(X_1^2w_1Y|u[r], (r)|vw_3X_2^2) \vdash (X_1^2w_1Yvw_3X_2^2,\ (r)u\,[r]),$$

where the string $(r)u\,[r]$ can never enter a new splicing, because in the rule $r : u \rightarrow v$ from P we have assumed $u \neq v$. The multiplicity of $X_1^2w_1Yu[r]$ and $(r)vw_3X_1^2$ has been reduced to 0 again (hence these strings are no more available), the multiplicity of $X_1^2w_1Yvw_3X_2^2$ is one. In this way, we have passed from $X_1^2w_1Yuw_3X_2^2$ to $X_1^2w_1Yvw_3X_2^2$, both having the multiplicity one, which corresponds to using the rule $r : u \rightarrow v$ 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

$$(X_1^2 w_1 Y \alpha | w_3 X_2^2, Z_1 \alpha Y | Z_2) \vdash (X_1^2 w_1 Y \alpha Z_2,\ Z_1 \alpha Y w_3 X_2^2).$$

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

$$(X_1^2 w_1 | Y \alpha Z_2, Z_1 | \alpha Y w_3 X_2^2) \vdash (X_1^2 w_1 \alpha Y w_3 X_2^2,\ Z_1 Y \alpha Z_2).$$

The first of the obtained strings has replaced $X_1^2 w_1 Y \alpha w_3 X_2^2$, which now have the multiplicity 0 (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 $L(\gamma) \subseteq L(G)$. □

Remark 2. The proof above also remains valid when the reflexivity assumption in Remark 1 is considered, which can be seen as follows: The rules of the type $u_1 \# u_2 \$ u_1 \# u_2$ for rules from the groups 1, 2, 3, 4, 5, 6, 8 and the rules of the type $u_3 \# u_4 \$ u_3 \# u_4$ for rules from the groups 2, 4, 6, 7, 8 involve strings whose multiplicities are at most one, hence these rules cannot be applied. The remaining rules of these types are applied to axioms of infinite multiplicity and what we obtain are always new copies of the axioms we used, because there is a unique position where we can splice. □

Lemma 2. $EH(mFIN, FIN) \subseteq EH(m[2], FIN)$.

Proof. Take an mH system $\gamma = (V, T, A, R)$, with finite $supp(A)$. Let $w_1, w_2, \ldots, w_n$ be the strings of $supp(A)$ such that $A(w_i) < \infty, 0 \leq i \leq n$, and let $z_1, \ldots, z_m$ be the strings in $supp(A)$ with $A(z_i) = \infty, 0 \leq i \leq m$. We construct the mH system

$$\gamma' = (V \cup \{c, d_1, d_2\}, T, A', R'),$$

where A' contains the string

$$w = (w_1 c)^{A(w_1)} (w_2 c)^{A(w_2)} \ldots (w_n c)^{A(w_n)},$$

with multiplicity 1, and the string

$$z = d_1 c z_1 c z_2 c \dots c z_m c d_2,$$

with infinite multiplicity. If $n = 0$, then w does not appear, if $m = 0$, then $z = d_1 c d_2$. Moreover

$$R' = R \cup \{\#c\$d_2\#,\ \#d_1\$c\#\}.$$

The string z can be used for cutting each w_i and each z_j from w and z, respectively. For instance, in order to obtain z_j we splice z with z using $\#d_1\$c\#$ for the occurrence of c to the left hand of z_j, that is

$$(d_1 c z_1 c \dots z_{j-1} c | z_j c \dots c z_m c d_2, |z) \vdash (d_1 c z_1 c \dots c z_{j-1} c z,\ z_j c \dots c z_m c d_2),$$

then we splice the second string with z again using $\#c\$d_2\#$, and we get

$$(z_j | c \dots c z_m c d_2, z|) \vdash (z_j,\ z c z_{j+1} \dots c z_m c d_2).$$

Arbitrarily many strings z_j can be produced, because $A'(z) = \infty$.

In order to produce the strings $w_i, 1 \leq i \leq n$, we start from the left hand end of w, applying $\#c\$d_2\#$ to w and z; we get w_1 and $zc(w_1 c)^{A(w_1)-1}(w_2 c)^{A(w_2)} \dots (w_n c)^{A(w_n)}$, both with multiplicity 1. Using the rule $\#d_1\$c\#$ for z and the second string, we obtain zcz and $(w_1 c)^{A(w_1)-1}(w_2 c)^{A(w_2)} \dots (w_n c)^{A(w_n)}$, again both with multiplicity 1. From the first string we can separate axioms $z_j, 1 \leq j \leq m$, but this is not important, because these axioms appear with infinite multiplicity in A. From the second string we can continue as above, cutting again a prefix w_1. In this way, exactly $A(w_1)$ copies of w_1 will be produced; in a similar way we can proceed for the other axioms $w_2, \dots, w_n$ in order to obtain exactly the $A(w_i)$ copies of w_i, $i = 2, \dots, n$.

The use of the nonterminals c, d_1 and d_2 guarantees that only the axioms of γ with multiplicity ∞ can be generated in an arbitrary number by the splicing rules in $R' - R$, whereas for each axiom w_i of γ with finite multiplicity $A(w_i)$ we can only obtain $A(w_i)$ copies of w_i. If a rule of R is used for splicing strings of the form $x_1 c x_2$, i.e. containing the nonterminal c, we finally will have to cut such a string by using the rules in $R' - R$ in order to obtain a terminal string. As we start from the axioms of γ, separated by c, and with the correct multiplicities (guaranteed by the mode of constructing the strings w and z), this also corresponds to a correct splicing in γ. Consequently, $L(\gamma') = L(\gamma)$. $\square$

Remark 3. If the reflexivity assumption does not change the language generated by γ, then this is also true for γ', because the additional rules are

$$\#c\$\#c,\ c\#\$c\#,\ d_2\#\$d_2\#,\ \#d_1\$\#d_1.$$

The last two rules can be used only on unique places, hence they reproduce the terms of the splicing. The first two rules can be used for splicing parts of

w, parts of z, or parts of w with parts of z, but this is always done without changing the subwords $w_i, 1 \leq i \leq n$, and $z_j, 1 \leq j \leq m$, and without modifying the number of copies of each $w_i, 1 \leq i \leq n$. $\square$

Lemma 3. $EH(m[1], FIN) \subseteq REG$.

Proof. Take $\gamma = (V, T, A, R)$ with $supp(A) = \{w\}$. If $A(w) < \infty$, then $L(\gamma)$ is, obviously, a finite language (every string in $L(\gamma)$ has a length not greater than $|w| \cdot A(w)$).

If $A(w) = \infty$, then $L(\gamma) \in EH(\omega[1], FIN) \subseteq EH(\omega FIN, FIN)$. In [Culik, Harju, 1991], [Pixton, 1995] it is proved that $H(\omega FIN, FIN) \subseteq REG$. As REG is closed under intersection and $L(\gamma) = L(\gamma') \cap T^*$, for $\gamma' = (V, V, \{w\}, R)$, we obtain $L(\gamma) \in REG$. Hence we conclude $EH(m[1], FIN) \subseteq REG$. $\square$

Lemma 4. $REG \subseteq EH(\omega[1], FIN)$.

Proof. Let $G = (N, T, S, P)$ be a Chomsky grammar, where all the rules in P are of the forms $X \to \lambda$ or $X \to aY$ for $X, Y \in N$ and $a \in T$. (Observe that every regular language can be generated by a grammar of that type; conversely the language generated by a grammar of that type is regular.) Construct the ωH system

$$\gamma = (N \cup T \cup \{E, F\}, T, A, R),$$

with

$$\begin{aligned} A &= \{(SE, \infty)\} \cup \\ &\cup \{(XaYE, \infty) \mid X \to aY \in P, X, Y \in N, a \in T\} \cup \\ &\cup \{(XF, \infty) \mid X \to \lambda \in P, X \in N\}, \end{aligned}$$

and R contains the following splicing rules:

$$\begin{array}{lll} 1. & \#XE\$X\#aYE, & \text{for } X \to aY \in P, \\ 2. & \#XE\$X\#F, & \text{for } X \to \lambda \in P, \\ 3. & \#F\$XXE\#. & \end{array}$$

Starting with the axiom SE, we can simulate the derivations in G by using rules in the first group (and axioms $XaYE$), then a rule in the second group (and the corresponding axiom XF), and finally applying the rule of type 3. Indeed, the following splicings are possible in γ for all $w \in T^*$

$$\begin{array}{lll} \text{(i)} & (w|XE, X|aYE) \vdash_1 (waYE, XXE), & \text{for } x \to aY \in P, \\ \text{(ii)} & (w|XE, X|F) \vdash_2 (wF, XXE), & \text{for } X \to \lambda \in P, \\ \text{(iii)} & (w|F, XXE|) \vdash_3 (w, XXEF). & \end{array}$$

Therefore, $L(G) \subseteq L(\gamma)$.

Conversely, each string in A is nonterminal, and each rule in R involves strings containing occurrences of E and/or of F. Let us examine the result of splicing operations different from the "legal" ones of the forms (i) – (iii) above. This means that $w \notin T^*$, hence the first term is one of $XXE, XXEF$ obtained as above, or $XaYE, XF$ in A, or other similar strings obtained by splicing. We have

$$\begin{aligned}
&(X|XE, X|aYE) \vdash_1 (XaYE, XXE), \text{ for } X \to aY \in P,\\
&(X|XE, X|F) \vdash_2 (XF, XXE), \text{ for } X \to \lambda \in P,\\
&(X|XE, XXE|) \vdash_3 (X, XXEXE),\\
&(X|XEF, X|aYE) \vdash_1 (XaYE, XXEF), \text{ for } X \to aY \in P,\\
&(X|XEF, X|F) \vdash_2 (XF, XXEF), \text{ for } X \to \lambda \in P,\\
&(X|XEF, XXE|) \vdash_3 (X, XXEXEF).
\end{aligned}$$

We obtain strings already considered, or axioms, or strings never entering a splicing (like X), or new strings composed of X, E, F. Moreover,

$$\begin{aligned}
&(Xa|YE, Y|bZE) \vdash_1 (XabZE, YYE), \text{ for } Y \to bZ \in P,\\
&(Xa|YE, Y|F) \vdash_2 (XaF, YYE), \text{ for } Y \to \lambda \in P.
\end{aligned}$$

Strings $XxZE, |x| \geq 2, x \in T^*, X, Z \in N$, can be used only as the first term for a rule in group 1, but the leftmost X can never be removed, hence no terminal string can be produced in this way. The same with $XxF, x \in T^*, X \in N$: the symbol F can be removed by rule 3, but X cannot.

The remarks above apply to the new strings $XXEXE, XXEXEF$, too: in all cases we obtain "legal" strings (axioms), or old or new strings composed of X, E, F, or strings starting with X and containing also terminals, but never turning to terminal strings. Consequently, no string outside $L(G)$ can be produced, $L(\gamma) \subseteq L(G)$. Hence we conclude that $L(G) = L(\gamma)$.

Now, using the same construction as in the proof of Lemma 2, we can combine all axioms from A with multiplicity ∞ into only one axiom z, hence we obtain $REG \subseteq EH(\omega[1], FIN)$, which completes the proof. □

We now are able to give a complete proof for Theorem 1 stated at the beginning of this section, i.e.

$REG = EH(m[1], FIN) \subset EH(m[2], FIN) = EH(mF_1, F_2) = RE,$
for all families F_1, F_2 *such that* $FIN \subseteq F_1 \subseteq RE, FIN \subseteq F_2 \subseteq RE.$

Proof of Theorem 1. Lemma 3 and Lemma 4 show that $REG = EH(m[1], FIN)$ (note that the ωH systems are a particular case of mH systems). Lemma 1 and Lemma 2 imply that $RE \subseteq EH(m[2], FIN)$. The inclusion $EH(m[2], FIN) \subseteq EH(mF_1, F_2)$, for $FIN \subseteq F_1 \subseteq RE$, $FIN \subseteq F_2 \subseteq RE$, is obvious. From the Turing/Church thesis we also have $EH(mRE, RE) \subseteq RE$, hence $EH(m[2], FIN) = RE$, which completes the proof. □

4. Computational completeness of ωH systems

In [Păun, **c**] it is proved that

$$EH(\omega[1], REG) = RE.$$

(In fact, in [Păun, **c**] one works with the splicing operation defined for triples $(x, y) \vdash z$, not for quadruples $(x, y) \vdash (z, w)$ as here, but one can easily see that the used construction works also for our restricted case.)

In view of the results in [Culik, Harju, 1991], [Pixton, 1995], we cannot replace REG by FIN in the previous equality without diminishing the obtained family. From practical points of view, the generality of REG is inconvenient. Fortunately, the proofs in [Păun, **c**] use quite particular types of regular sets (strictly locally testable languages) and, moreover, they suggest that finite sets of splicing rules are enough for simulating arbitrary type-0 grammars, provided certain (rather weak) control mechanisms are added to the system. In the following we consider two such variants.

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^+. To a triple p we associate the string

$$\rho(p) = u_1\#u_2\$u_3\#u_4@w_1\&\ldots\&w_k@v_1\&\ldots\&v_m,$$

where $C_1 = \{w_1, \ldots, w_k\}$, $C_2 = \{v_1, \ldots, v_m\}$, $k, m \geq 0$ and @, & are new special symbols. We denote $\rho(R) = \{\rho(p) \mid p \in R\}$.

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). □

As we have pointed out at the end of Remark 1, it is possible to check in a lab whether or not a given word appears in a DNA sequence (see again [Adleman, 1994]). So far, however, this can only be done manually; what we need here is a way to automatically block the splicing by a rule r when the associated contexts C_1, C_2 are not present. How to do this is a practical (open) problem.

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 with $\rho(R) \in F_2$, is denoted by $EH(\omega F_1, cF_2)$; instead of F_1 we put $[k]$ when systems with at most k axioms are used. The strings over V in the sets C_1, C_2 control the use of splicing rules in the same way as permitting contexts in random context grammars are doing ([van derWalt, 1971]).

A result similar to Theorem 1 holds also for ωH systems with permitting contexts, with the difference that the hierarchy on the number of axioms now collapses to a hierarchy with one level only.

Theorem 2. *$EH(\omega[1], cFIN) = EH(\omega F_1, cF_2) = RE$, for all families F_1, F_2 such that $FIN \subseteq F_1 \subseteq RE$, $FIN \subseteq F_2 \subseteq RE$.*

The proof of this theorem will follow from combining the next two lemmas, the Turing/Church thesis, and the obvious inclusion $EH(\omega FIN, cFIN) \subseteq EH(\omega F_1, cF_2)$, for F_1, F_2 as above.

Lemma 5. *$RE \subseteq EH(\omega FIN, cFIN)$.*

Proof. Consider a type-0 Chomsky grammar $G = (N, T, S, P)$ like in the proof of Lemma 1; let Lab 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

$$\begin{array}{rcl} V & = & N \cup T \cup \{B, E, E', F, F', X, X', Y, Z\} \cup \{Y_\alpha \mid \alpha \in U' \cup Lab\}, \\ A & = & \{F'Z, XBSY, XZ, ZE, ZE', ZF, ZY\} \cup \\ & & \{ZY_\alpha, X'\alpha Z \mid \alpha \in U'\} \cup \{ZY_r, X'vZ \mid r : u \to v \in P\} \end{array}$$

and R contains the following rules with permitting contexts:

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

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'.

Let us look in some details to the way how the ideas mentioned above work:

When simulating derivation steps in G, we start from $XBSY$, and at every step the markers X and its variant X' as well as Y and its variants Y_β, $\beta \in U' \cup Lab$, are present to indicate the ends of the string. Moreover, in each moment the symbol B tells us where the beginning of the string is whose permutation we consider.

All the splicing rules with permitting contexts contained in R require an occurrence of the symbol Z in the second term of the splicing; in fact these strings are meant to be taken from A. If we start with rule 1 applied to $XBSY$ and ZvY, for some $S \to v \in P$, this starts the simulation of a derivation in G. In general, having a string Xx_1Bx_2uY and $r : u \to v \in P$, we can obtain Xvx_1Bx_2Y by using the associated rules in the groups 1, 2, 3, and 4. This corresponds to a derivation step $x_2ux_1 \Longrightarrow x_2vx_1$ in G. The corresponding steps in γ have the following form:

1. $(Xw|uY, Z|Y_r) \vdash_p (XwY_r, ZuY)$, for $p = (\#uY\$Z\#Y_r; \{X\}, \emptyset) \in R$, where $r : u \to v \in P$ and $w \in U^*\{B\}U^*$;
2. $(X|wY_r, X'v|Z) \vdash_p (XZ, X'vwY_r)$, for $p = (X\#\$X'v\#Z; \{Y_r\}, \emptyset)$, where $r : u \to v \in P$, $w \in U^*\{B\}U^*$;
3. $(X'vw|Y_r, Z|Y) \vdash_p (X'vwY, ZY_r)$, for $p = (\#Y_r\$Z\#Y; \{X'\}, \emptyset)$, where $r : u \to v \in P$, $w \in U^*\{B\}U^*$;
4. $(X'|vwY, X|Z) \vdash_p (X'Z, XvwY)$, for $p = (X'\#\$X\#Z; \{Y\}, \emptyset)$ (where $r : u \to v \in P$, $w \in U^*\{B\}U^*$).

As additional results (that cannot already be found in the set of axioms A) of the splicing derivations listed above we obtain the strings

$$ZuY, \quad \text{for } r : u \to v \in P$$

and the string $X'Z$ if $v \neq \lambda$ for all $r : u \to v \in P$.

To each string $Xw\alpha Y$, $\alpha \in U$, $w \in U^*\{B\}U^*$, respectively $Xw\alpha Y$, $\alpha = B$, $w \in U^*$, we can also apply the appropriate rule from group 5 and then proceed with applying the appropriate rules from the groups 6 and 7; finally, by using the rule in group 4 again, we obtain the string $X\alpha wY$. The symbol α has been moved from the right-hand end to the left-hand end of the string, exactly what we need for rotating the underlying sentential form:

1. $(Xw|\alpha Y, Z|Y_\alpha) \vdash_p (XwY_\alpha, Z\alpha Y)$, for $p = (\#\alpha Y\$Z\#Y_\alpha; \{X\}, \emptyset) \in R$, where $\alpha \in U$ and $w \in U^*\{B\}U^*$ or $\alpha = B$ and $w \in U^*$;
2. $(X|wY_\alpha, X'\alpha|Z) \vdash_p (XZ, X'\alpha wY_\alpha)$, for $p = (X\#\$X'\alpha\#Z; \{Y_\alpha\}, \emptyset)$, where $\alpha \in U$ and $w \in U^*\{B\}U^*$ or $\alpha = B$ and $w \in U^*$;
3. $(X'\alpha w|Y_\alpha, Z|Y) \vdash_p (X'\alpha wY, ZY_\alpha)$, for $p = (\#Y_\alpha\$Z\#Y; \{X'\}, \emptyset)$, where $\alpha \in U$ and $w \in U^*\{B\}U^*$ or $\alpha = B$ and $w \in U^*$;
4. $(X'|\alpha wY, X|Z) \vdash_p (X'Z, X\alpha wY)$, for $p = (X'\#\$X\#Z; \{Y\}, \emptyset)$ (where $\alpha \in U$ and $w \in U^*\{B\}U^*$ or $\alpha = B$ and $w \in U^*$).

As additional results (that cannot already be found in the set of axioms A) of the splicing derivations listed above we obtain the strings

$$Z\alpha Y, \quad \text{for } \alpha \in U'$$

and $X'Z$ if $v \neq \lambda$ for all $r : u \to v \in P$.

Notice again that every string obtained from $XBSY$ so far and not containing the symbol Z is of the form $\alpha_1 x_1 B x_2 \alpha_2$, with (α_1, α_2) being one of the pairs (X, Y), (X, Y_α), (X', Y_α), (X', Y), $\alpha \in U' \cup Lab$, hence these symbols appearing as permitting contexts in the splicing rules of R precisely control the work of γ.

In order to obtain a terminal string we have to use rules from the groups 8, 9, 10, and 11, in this order (Y must be present when using rule 8); it is easy to see that in fact we have to start with a string of the form $XBwY$ with $w \in U^*$:

1. $(Xw'|Y, Z|F) \vdash_p (Xw'F, ZY)$, for $p = (\#Y\$Z\#F; \{X\}, \emptyset) \in R$, where $w' \in U^*\{B\}U^*$;
2. $(XB|wF, F'|Z) \vdash_p (XBZ, F'wF)$, for $p = (XB\#\$F'\#Z; \{F\}, \emptyset)$, where $w \in U^*$;

3. $(F'w|F, ZE|) \vdash_p (F'w, ZEF)$, for $p = (\#F\$ZE\#; \{F'\}, \emptyset)$,
   where $w \in U^*$;

4. $(F'|w, |ZE') \vdash_p (F'ZE', w)$ for $p = (F'\#\$\#ZE'; \{F'\}, \emptyset)$,
   where $w \in U^*$.

As additional results (that cannot already be found in the set of axioms A) of the splicing derivations listed above we obtain the strings

$$XBZ,\ ZEF,\ F'ZE'.$$

In sum, we can produce in γ every terminal string that can be produced by G, i.e. $L(\gamma) \supseteq L(G)$.

Conversely, no parasitic terminal strings can be generated in γ, i.e. $L(\gamma) \subseteq L(G)$:

All the strings generated in addition to the axioms represent a sentential form of a derivation in G (maybe in a rotated version) or contain the symbol Z. Only the strings containing this symbol Z can be used as second terms of a splicing in γ, and except for the string $F'ZE'$, all strings that really can be used already appear in the set of axioms A; with $F'ZE'$ we can also do the following derivations involving strings of the forms F'^lZE' as well as F'^kw and F'^kwF, for $l, k \geq 0$: If $k = 0$, then we obtain w respectively wF, where the string wF cannot be processed any more, because the rule in group 10 needed for eliminating the symbol F requires the occurrence of the symbol F'; if the obtained string w is not terminal, then no further step can be done. If $k \geq 2$, then we can only obtain similar strings like from $F'wF$. The strings F'^lZE' for $l \geq 1$ can be used freely in the same way as the string ZE' itself.

1. $(F'^m|F'^kZE', F'^l|ZE') \vdash_p (F'^mZE', F'^lF'^kZE')$,
   for $p = (F'\#\$\#ZE'; \{F'\}, \emptyset)$, where $m, k, l \geq 0$ and $m + k \geq 1$;

2. $(F'^m|F'^kw, F'^l|ZE') \vdash_p (F'^mZE', F'^lF'^kw)$, for $p = (F'\#\$\#ZE'; \{F'\}, \emptyset)$,
   where $m, k, l \geq 0$ and $m + k \geq 1$ and $w \in U^*$;

3. $(F'^m|F'^kwF, F'^l|ZE') \vdash_p (F'^mZE', F'^lF'^kwF)$,
   for $p = (F'\#\$\#ZE'; \{F'\}, \emptyset)$, where $m, k, l \geq 0$ and $m + k \geq 1$ and $w \in U^*$;

4. $(F'^kw|F, ZE|) \vdash_p (F'^kw, ZEF)$, for $p = (\#F\$ZE\#; \{F'\}, \emptyset)$,
   where $k \geq 1$ and $w \in U^*$.

In sum, we obtain $L(\gamma) = L(G)$. □

Remark 4. Note that in the rules $(p_1; C_1, C_2)$ of the ωH system with permitting contexts constructed in the proof of Lemma 5, the pairs (C_1, C_2) of permitting contexts are of the special form $(\{X\}, \emptyset)$ for some nonterminal X, i.e. we only check the occurrence of one nonterminal in the first term of the splicing. Also observe that when considering the rules $(u_1\#u_2\$u_1\#u_2; C_1, C_1)$, $(u_3\#u_4\$u_3\#u_4; C_2, C_2)$ for each rule $(u_1\#u_2\$u_3\#u_4; C_1, C_2)$ in the previous proof, the generated language is the same: the place of using such rules is unique, hence we always have $(x, y) \vdash_p (x, y)$. $\square$

The construction in the previous proof is an extension of the idea used in the second example in Section 2, and it also reminds the way of simulating Turing machines by Post TAG systems in [Minsky, 1961].

Lemma 6. $EH(\omega FIN, cFIN) \subseteq EH(\omega[1], cFIN)$.

Proof. Consider an ωH system $\gamma = (V, T, A, R)$ with permitting contexts, and suppose that $A = \{w_1, w_2, \ldots, w_n\}, n \geq 2$. Take the new symbols $c_0, c_1, \ldots, c_n, c_{n+1}$ and construct

$$\gamma' = (V \cup \{c_0, c_1, \ldots, c_n, c_{n+1}\}, T, \{w\}, R'),$$

where

$$\begin{aligned} w &= c_0c_1w_1c_2w_2 \ldots c_n, c_{n+1}c_0, \\ R' &= R \cup R'' \end{aligned}$$

with R'' containing the following rules with permitting contexts:

1. $(c_0\#c_1\$c_i\#w_ic_{i+1}; \emptyset, \emptyset)$, for $1 \leq i \leq n$,
2. $(c_0w_i\#c_{i+1}\$c_{n+1}\#c_0; \emptyset, \emptyset)$, for $1 \leq i \leq n$,
3. $(\#c_0c_1\$c_0\#w_ic_0; \emptyset, \emptyset)$, for $1 \leq i \leq n$,
4. $(w_i\#c_0\$c_{n+1}c_0\#; \emptyset, \emptyset)$, for $1 \leq i \leq n$.

It is easy to see that by splicing w with itself using a rule of type 1 we obtain the string $c_0w_ic_{i+1}w_{i+1}c_{i+2} \ldots c_nw_nc_{n+1}c_0$ (and $c_0c_1w_1 \ldots c_ic_1w_1 \ldots c_nw_nc_{n+1}c_0$). By splicing this string with w using the rule in group 2 with the suitable value of i we get $c_0w_ic_0$ (and $c_0c_1w_1 \ldots c_{n+1}c_{i+1}w_{i+1} \ldots c_nw_nc_{n+1}c_0$). Using rules from the groups 3 and 4, we now can remove the front c_0 and then the back c_0. All the axioms of γ can be detached from w; now applying rules in R we can produce each string of $L(\gamma)$, hence $L(\gamma) \subseteq L(\gamma')$.

Conversely, if we apply a rule of R to two copies of w (possibly on substrings w_i, w_j which cannot be spliced correctly when they are separated, due to the restrictions imposed by the permitting contexts), producing strings of the form $z_1c_iz_2c_jz_3$, with z_2 in V^*, then z_2 can be separated from the neighboring c_i, c_j if and only if $j = i+1$ and $z_2 = w_i$, hence no new string can be produced in this way. The same holds if the string $z_1c_iz_2c_jz_3$ is obtained after several splicings, possibly also using strings of the form $c_0w_ic_{i+1} \ldots c_nw_nc_{n+1}c_0$ etc. or $c_0w_ic_0$, obtained after using rules in groups 1, 2. On the other hand, splicing copies of w or of $c_0 w_i c_{i+1} \ldots c_n w_n c_{n+1} c_0$ etc. and $c_0 w_i c_0$ as above using rules in R, we can obtain only strings starting and ending with c_0, hence we can separate substrings $c_i z c_j, z \in V^*$, out of them if and only if $j = i + 1$ and $z = w_i$.

If a rule in R is applied to two strings of the form $w_i c_0, w_j c_0$ obtained after using rules of type 3, then instead of v and w we obtain vc_0 and wc_0, where v and w are correct splicing results. If a rule in R is applied to a string $w_i c_0$ and to another string of a different form (hence also starting with a symbol c_0), then we either obtain a string both starting and ending with c_0 (when $w_i c_0$ is the second term of the splicing), and, as above, we can separate from it only substrings of the form $c_j w_j c_{j+1}$), or we obtain a string $z_1 c_j z_2, z_1 \in V^*$, with w_i contributing to z_1, and z_1 cannot be separated from c_j. Therefore, the rules in R can be used successfully only for splicing the result of using rules in groups 1, 2, 3 (and 4), and this corresponds to the work of γ, starting from the axioms in A. Therefore, also $L(\gamma') \subseteq L(\gamma)$ is true. $\square$

Remark 5. When considering rules associated to rules in R'' by the reflexivity assumption, the generated language is not modified, because these rules can be applied only on places defined by symbols $c_0, c_1, \ldots, c_{n+1}$, hence what we obtain always consists of complete strings $w_i, 1 \leq i \leq n$, and symbols $c_0, c_1, \ldots, c_{n+1}$. $\square$

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 forbidding 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 [Adleman, 1994]).

Definition 5. An extended ωH system *with forbidding 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 forbidding 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^+. As in Definition 4, the triple p can also be represented by the string

$$\rho(p) = u_1 \# u_2 \$ u_3 \# u_4 @ w_1 \& \ldots \& w_k @ v_1 \& \ldots \& v_m,$$

where $D_1 = \{w_1, \ldots, w_k\}$ and $D_2 = \{v_1, \ldots, v_m\}$, $k, m \geq 0$. We denote $\rho(R) = \{\rho(p) \mid p \in R\}$.

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 $\rho(R) \in F_2$, is denoted by $EH(\omega F_1, fF_2)$. Instead of F_1 we put $[k]$ when systems with at most k axioms are used.

Theorem 3. *$EH(\omega[1], fFIN) = EH(\omega F_1, fF_2) = RE$, for all families F_1, F_2 such that $FIN \subseteq F_1 \subseteq RE$, $FIN \subseteq F_2 \subseteq RE$.*

The relations above are consequences of the following two lemmas, of the Turing/Church thesis, and of the obvious inclusion $EH(\omega[1], fFIN) \subseteq EH(\omega F_1, fF_2)$, F_1, F_2 as in the theorem.

Lemma 7. *$RE \subseteq EH(\omega FIN, fFIN)$.*

Proof. Consider a type-0 grammar $G = (N, T, S, P)$ like in the proof of Lemma 1, let Lab be the set of labels of the rules in P, and denote $U' = N \cup T \cup \{B\}$. We now construct the ωH sytem with forbidding contexts

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

where V, T and A are as in the proof of Lemma 5 and R contains the following rules with forbidding 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 Lemma 5, the forbidding contexts in the ωH system with forbidding contexts constructed above control the derivation sequences possible in γ. Hence, according to the arguments given in Lemma 5, we conclude $L(\gamma) = L(G)$. □

Remark 6. Note that in the rules $(p_1; D_1, D_2)$ of the ωH system with forbidding contexts constructed in the proof of Lemma 7, the pairs (D_1, D_2) of forbidding contexts are of the special form $(M, \emptyset)$ with $M \subseteq V - T$, i.e. we only check the non-occurrence of some nonterminals in the first term of the splicing. In the same way as for the construction in Lemma 5, we can show that the reflexivity assumption does not change the generated language. □

Lemma 8. $EH(\omega FIN, fFIN) \subseteq EH(\omega[1], fFIN)$.

Proof. For an ωH system with forbidding contexts $\gamma = (V, T, A, R)$ with $A = \{w_1, w_2, \ldots, w_n\}$, we construct an ωH system

$$\gamma' = (V \cup \{c\}, T, \{w\}, R'),$$

where

$$w = ccw_1cw_2c...cw_ncc$$

and

$$\begin{aligned} R' &= \{(p; F_1 \cup \{c\}, F_2 \cup \{c\}) \mid (p; F_1, F_2) \in R\} \cup \\ &\quad \{(\#cc\$c\#; \emptyset, \emptyset), (\#c\$cc\#; \emptyset, \emptyset)\}. \end{aligned}$$

The splicing rule $\#cc\$c\#$ applied to two copies of w allows us to cut one w at the symbol c in front of some axiom w_i, i.e. as one new string we obtain $w_ic...w_ncc$, and the splicing rule $\#c\$cc\#$ applied to this string and to another copy of w then allows us to separate the axiom w_i.

As the rules from R can only be applied to strings not containing the nonterminal c any more, we obviously obtain $L(\gamma') = L(\gamma)$. □

Another way of controlling the work of splicing rules is to impose sequencing restrictions (like in programmed grammars or in graph controlled grammars etc., see [Dassow, Păun, 1989]). Probably also such other control mechanisms used in formal language theory for regulating the work of context-free grammars can be used as in Theorems 2 and 3 above, but we do not follow further this direction here. It is a matter of practical feasibility to choose the one or the other of these regulating mechanisms, according to the possibility to implement them. The previous Theorems 1, 2, and 3 in a convincing manner prove that the full power of Turing machines can be reached by considering extended H systems with finite components (axiom and sets of rules), provided such natural control mechanisms are involved.

5. Universal H systems

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.

Surprisingly enough, universal H systems exist for all the types of extended H systems we have considered in the previous sections, and this is a consequence of the very proofs of Lemmas 1, 5, and 7.

Definition 7. 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, $A_U \subseteq V_U^*, A_U \in F_1$, 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 $\gamma_U' = (V_U, T, A_U \cup A_\gamma, R_U)$. □

The particularizations of this definition to mH systems or to ωH systems with permitting respectively forbidding 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.

Lemma 9. *Given an extended H system $\gamma = (V, T, A, R)$ of type (F_1, F_2), for F_1, F_2 families of languages closed under λ-free morphisms, we can construct an extended H system $\gamma' = (\{c_1, c_2\} \cup T, T, A', R')$ of the same type (F_1, F_2), such that $L(\gamma) = L(\gamma')$. This is also true for γ being an mH system or an ωH system with permitting respectively forbidding contexts.*

Proof. If $V - T = \{Z_1, \ldots, Z_n\}$, then we consider the morphism $h : V^* \longrightarrow (\{c_1, c_2\} \cup T)^*$, defined by

$$\begin{aligned} h(Z_i) &= c_1 c_2^i c_1, \ 1 \leq i \leq n, \\ h(a) &= a, \ a \in T. \end{aligned}$$

Then

$$\begin{aligned} A' &= h(A), \\ R' &= \{h(u_1)\#h(u_2)\$h(u_3)\#h(u_4) \mid u_1\#u_2\$u_3\#u_4 \in R\}. \end{aligned}$$

The equality $L(\gamma) = L(\gamma')$ is obvious, because due to the form of the axioms in A' and of the rules in R', the blocks $c_1 c_2^i c_1$, $1 \leq i \leq n$, are never broken by splicing, they behave in the same way as the corresponding symbols Z_i do.

For the case when γ is an mH or an ωH system with permitting respectively forbidding contexts, the definitions of the corresponding components of γ' are obvious, transferring them from γ to γ' by the morphism h. □

¿From a practical point of view, the main results of this paper are the following two theorems:

Theorem 4. *For every given alphabet T there exists an mH system of type $(m[2], FIN)$ 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, 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 $G'_U = (N_U, T, w(G), P_U)$. (The language $L(G'_U)$ 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 [Turing, 1936] 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 [Calude, Păun, 1981]).

For a given universal type-0 grammar $G_U = (N_U, T, -, P_U)$, we follow the construction in the proof of Lemma 1, obtaining an mH system $\gamma_1 = (V_1, T, A_1, R_1)$, where the axiom (with multiplicity 1) $w_0 = X_1^2 Y S X_2^2$ is not considered. Remark that all other axioms in A_1 (all having infinite multiplicity) and the rules in R_1 depend on N_U, T and P_U only, hence they are fixed.

As in the proof of Lemma 2, we now pass from γ_1 to $\gamma_2 = (V_2, T, A_2, R_2)$, with at most two axioms in A_2. In fact, as A_1 contains only axioms with infinite multiplicity, A_2 consists of only one string (that one denoted by z in the proof of Lemma 2), namely with infinite multiplicity.

We now follow the proof of Lemma 9, codifying all symbols in $V_2 - T$ 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 $\gamma_0 = (V, T, A, R)$. From the Turing/Church thesis we know that $L(\gamma_0) \in RE$, hence there is 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). Construct the code of G_0, $w(G_0)$, as imposed by the definition of universal type-0 grammars one uses, consider the string

$$w'_0 = X_1^2 Y w(G_0) X_2^2,$$

corresponding to the axiom w_0 in the proof of Lemma 1, then codify w'_0 over $\{c_1, c_2\} \cup T$ as we have done above with the axioms of γ_2. By $w(\gamma_0)$ denote the obtained string. Then $L(\gamma'_U) = L(\gamma_0)$, for $\gamma'_U = (\{c_1, c_2\} \cup T, T, \{(w(\gamma_0), 1)\} \cup A_U, R_U)$.

This can be seen easily: In the proof of Lemma 1, the system γ simulates the work of G, starting from the axiom S of G, bracketed as in $X_1^2 Y S X_2^2$. If instead of S we put an arbitrary string x over the alphabet of G, then we obtain in γ exactly the language of terminal strings y such that $x \Longrightarrow^* y$ in G. If we start from a universal grammar G_U and S is replaced by the code $w(G_0)$ of a type-0 grammar G_0 equivalent with γ_0, then the system γ_U, associated as above with G_U, will simulate the work of G_U, starting from $w(G_0)$. Hence $L(\gamma'_U) = L(G'_U) = L(G_0) = L(\gamma_0)$, for $G'_U = (N_U, T, w(G_0), P_U)$. □

Theorem 5. *For every given alphabet T, there is an ωH system of type $(\omega[1], FIN)$ with permitting respectively forbidden contexts that is universal for the class of ωH systems with permitting respectively forbidding contexts and with the terminal alphabet T.*

Proof. This can be proved in the same way as in Theorem 4 above, using the constructions given in the proofs of Lemmas 5, 7, and 9. □

Remark that the universal H systems γ_U furnished by the previous proofs contain only one axiom and that in order to enable them to simulate any given H system γ, we only have to add to γ_U one more axiom (of multiplicity one in the case of mH systems).

The computers based on γ_U seem to be quite economical, as concerning their programming.

6. Conclusions

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, [Păun, **a**], [Păun, **c**], [Păun et al.], or additional languages such as Dyck languages, palindrom languages etc. [Yokomori, Kobayaski, 1995]). Our results in Sections 3 and 4 are the strongest possible of this type, because we only use the splicing, the intersection with a language of the form T^* (which according to [Păun et al.], 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 forbidding contexts, such features are essential and cannot be removed; indeed, in view of [Culik, Harju, 1991] and [Pixton, 1995], ordinary finite splicing systems can produce regular languages only.

It is rather interesting to notice in this context the tremendous influence that certain apparently minor changes in the structure of an H system have on its generative power. For instance, in the case of mH systems, the transition from one axiom to two axioms causes an increase from the power of finite automata to the full power of Turing machines (Theorem 1). Similarly in the case of ωH systems the addition of permitting contexts, one for each rule, strengthens the generative power to that of the computability power of Turing machines (Lemma 5).

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 4 and 5 rely on one hand on the lemmas in Sections 3 and 4, and on the other hand on the existence of universal type-0 grammars and on the possibility of commuting from an H system to a type-0 grammar and conversely. This reduces the problem of the existence of universal H systems to the existence of universal Chomsky grammars (or 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 a universal H system. In short, we leave this task to a joint team of language theorists and practitioners of DNA computing.

Acknowledgements

Very useful discusssions with Arto Salomaa, Gabriel Thierrin, Helmut Jürgensen, Tom Head, and Dennis Pixton are gratefully acknowledged.

References

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

[Adleman, 1995] L. M. Adleman, On constructing a molecular computer, Manuscript in circulation, January 1995.

[Calude, Păun, 1981] C. Calude, Gh. Păun, Global syntax and semantics for recursively enumerable languages, *Fundamenta Informatica*, 4, 2 (1981), 245 – 254.

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

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

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

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

[Eilenberg, 1971] S. Eilenberg, *Automata, Languages and Machines, A*, Academic Press, New York, 1974.

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

[Goldberg, 1989] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, Mass., 1989.

[Harrison, 1978] M. A. Harrison, *Introduction to Formal Language Theory*, Addison-Wesley, Reading, Mass., 1978.

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

[Head et al.] 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.

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

[Koza, Rice, 1992] J. H. Koza, J. P. Rice, *Genetic Algorithms: The Movie*, MIT Press, Cambridge, Mass., 1992.

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

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

[Mateescu et al., 1995] A. Mateescu, Gh. Păun, G. Rozenberg, A. Salomaa, Simple splicing systems, *Techn. Report 95 – 09*, Leiden Univ., Dept. of Computer Science, 1995.

[Minsky, 1961] M. L. Minsky, Recursive unsolvability of Post's problem of 'tag' and other topics in the theory of Turing machines, *Annals of Math.,* 74, 3 (1961), 437 – 455.

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

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

[Păun, **b**] Gh. Păun, On the power of the splicing operation, *Intern. J. Computer Math.*, to appear.

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

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

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

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

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

[van derWalt, 1971] A. P. J. van der Walt, Random context grammars, *Proc. IFIP Congress* 1970, North-Holland Publ. Co., Amsterdam, 1971, 66 – 68.

[Yokomori, Kobayaski, 1995] 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.