DNA SPLICING SYSTEMS AND POST SYSTEMS

C. FERRETTI,[a] S. KOBAYASHI
Department of Computer Science and Information Mathematics
University of Electro-Communications
1-5-1 Chofugaoka, Chofu, Tokyo 182, JAPAN

This paper concerns the formal study on the generative powers of extended splicing (H) systems. First, using a classical result by Post which characterizes the recursively enumerable languages in terms of his Post Normal systems, we establish several new characterizations of extended H systems which not only allow us to have very simple alternative proof methods for the previous results mentioned above, but also give a new insight into the relationships between families of extended H systems. We show a kind of normal form for extended H systems exactly characterizing the class of regular languages. We also show a new representation result for the family of context-free languages in terms of extended H systems.

1 Introduction

A series of papers on "theoretical" research on DNA computing has lately been attracting much attention in computer science, which has been originated by Tom Head's work on splicing systems (or H systems) and their languages for modeling DNA recombination.[3]

In investigating the computational power of extended H systems, all of the previous work construct a fairly complicated H system to characterize the recursive enumerability of an extended H system, from which one can hardly recognize the essentials necessary for achieving the desired complexity. This will also cause a real "bottleneck" when we think of designing a feasible implementation method for building DNA computer in a test tube.

In this paper, we propose a new method for realizing the universal computational power of extended H systems which is based on a classical but beautiful characterization of the recursively enumerable languages in terms of Post systems.[7] In his historical paper, Post has proposed a new string rewriting system having rules of the form : $uX \rightarrow Xw$, where u, w : strings over the (total) alphabet involved, X: variable identifiable with any string. For example, one can obtain a new string vw from uv by applying the rule, where X is identified with a string v.

The key idea used in our method is illustrated in Figure 1 where one application of original Post rule is simulated by a number of basic types of rewriting

[a] {ferretti,satoshi}@gold.cs.uec.ac.jp.
The first author was on leave from Dept. of Computer Science, Univ. of Milano.

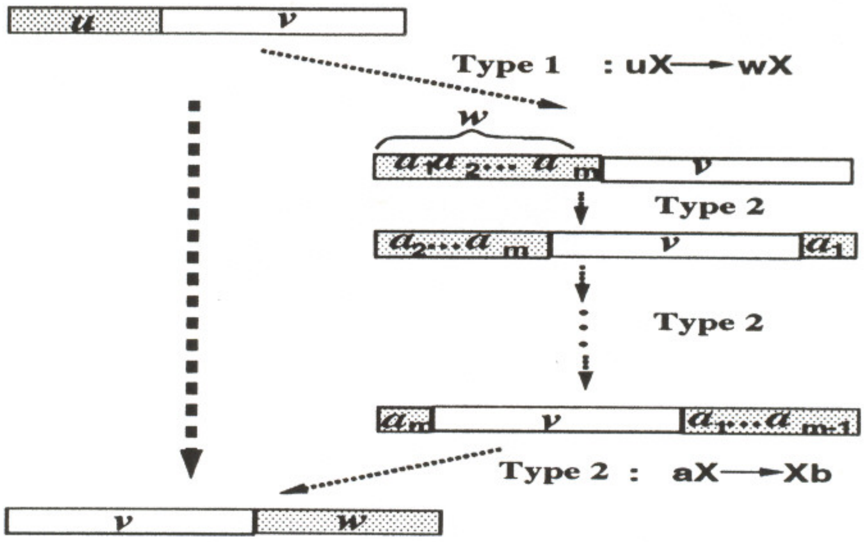

Figure 1. Decomposing Into Two Basic Types

steps so that those steps may easily be carried out by splicing operations. In fact, such a decomposition of a rule $uX \rightarrow Xw$ can be realized as follows : We first introduce new rule $uX \rightarrow wX$ which is called *Type 1*, then use a number of simple original rule of the form : $aX \rightarrow Xb$ (a, b : symbol), called *Type 2.* It is easy to see that this transformation of the rule application can be performed in a deterministic manner, so that the simulation process works well.

2 Preliminary

V^* is the set of all (finite length) strings over a finite alphabet V. The empty string is denoted by λ, and $V^+ = V^* - \{\lambda\}$. For a string $x \in V^*$, $|x|$ denotes the length of x, i.e., the number of symbols from V comprising x. The families of recursively enumerable languages, context-free languages, regular languages and finite languages are denoted by $\mathcal{RE}$, $\mathcal{CF}$, $\mathcal{REG}$ and $\mathcal{FIN}$, respectively.

An *extended Head system* (or EH system) is a quadruple $H = (V, \Sigma, A, R)$, where V is an alphabet of H, $\Sigma(\subseteq V)$ is the terminal alphabet, $A(\subseteq V^*)$ is the set of *axioms*, and R is the set of *splicing rules* and $R \subseteq V^*\#V^*\$V^*\#V^*$ ($\$, \#$ are special symbols not in V).

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

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

For an EH system $H = (V, \Sigma, A, R)$ and a language $L \subset V^*$, we write $\sigma(L) = \{z \in V^* | (x, y) \vdash_r (z, w) \text{ or } (x, y) \vdash_r (w, z), \text{ for some } x, y \in L, r \in R\}$, and define $\sigma^*(L) = \bigcup_{i\geq 0} \sigma^i(L)$ where

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

The language generated by H is defined by $L(H) = \sigma^*(A) \cap \Sigma^*$.

A *multi-set* M on V^* is taken as a recursive function $M : V^* \to \mathbf{N} \cup \{\infty\}$, where $\mathbf{N}$ is the set of non-negative integers. The set $\{w \in V^* | M(w) > 0\}$ is called the support of M and is denoted by $supp(M)$. A multi-set M is represented by a set of pairs $\{(x, M(x)) | x \in supp(M)\}$. For two multi-sets M_1, M_2, $(M_1 - M_2)(x) = M_1(x) - M_2(x)$ if $M_1(x) \geq M_2(x)$ (for all $x \in V^*$), and $(M_1 \cup M_2)(x) = M_1(x) + M_2(x)$.

An EH system $H = (V, \Sigma, A, R)$ is called EH system with *multiplicity* if A is a multi-set on V^*. For such an $H = (V, \Sigma, A, R)$ and two multi-sets M_1, M_2 on V^*, we define

$$\begin{array}{lll} M_1 \Longrightarrow M_2 & \text{iff} & \text{there exist } x, y, z, w \in V^* \text{ such that} \\ & \text{(i)} & M_1(x) \geq 1, M_1(y) \geq 1 \text{ and if } x = y, \text{ then } M_1(x) \geq 2, \\ & \text{(ii)} & (x, y) \vdash_r (z, w) \text{ , for some } r \in R, \\ & \text{(iii)} & M_2 = (M_1 - \{(x,1),(y,1)\}) \cup \{(z,1),(w,1)\}. \end{array}$$

The language generated by an EH system H with multiplicity is defined by

$$L(H) = \{w \in \Sigma^* | w \in supp(M), \text{ for some } M \text{ such that } A \Longrightarrow^* M\},$$

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

For two families of languages $\mathcal{F}_1, \mathcal{F}_2$, we denote

$$\begin{aligned} \mathrm{EH}(\omega\mathcal{F}_1, \mathcal{F}_2) &= \{L(H) | H = (V, \Sigma, A, R), A \in \mathcal{F}_1, R \in \mathcal{F}_2\} \\ \mathrm{EH}(m\mathcal{F}_1, \mathcal{F}_2) &= \{L(H) | H = (V, \Sigma, A, R) \text{ with mult.}, supp(A) \in \mathcal{F}_1, R \in \mathcal{F}_2\}. \end{aligned}$$

Further, the families $\mathrm{EH}(m[k], \mathcal{F}_2)$ and $\mathrm{EH}(\omega[k], \mathcal{F}_2)$, for each $k \geq 1$, denote the ones generated by EH systems with and without multiplicity having at most k axioms, respectively.

Some important recent results related to the generating power of these systems are recalled in the following statement:

Theorem 1 (Freund et al.[2])
(i) $\mathcal{REG} = \mathrm{EH}(m[1], \mathcal{FIN}) = \mathrm{EH}(\omega[1], \mathcal{FIN}) = \mathrm{EH}(\omega\mathcal{FIN}, \mathcal{FIN})$
(ii) $\mathcal{RE} = \mathrm{EH}(m[2], \mathcal{FIN}) = \mathrm{EH}(m\mathcal{FIN}, \mathcal{FIN})$

3 Post Normal Systems

A *Post Normal system* (PN system) is a quadruple $G = (V, \Sigma, P, A)$, where V and Σ are finite alphabets such that $\Sigma \subseteq V$, and P is a finite set of rules of the form : $uX \to Xw$ (X: unique variable not in V, $u, w \in V^*$). $A (\subset V^+)$ is a finite set of *axioms*.

Given strings $\alpha, \beta \in V^*$, a binary relation $\Longrightarrow$ is defined as follows:

$$\alpha \Longrightarrow \beta \quad \text{iff} \quad \exists uX \to Xw \in P \ \text{ and } \ \exists \delta \in V^* \ \ [\alpha = u\delta, \beta = \delta w].$$

The reflexive and transitive closure of $\Longrightarrow$ is denoted by $\Longrightarrow^*$.

For a given G, a language generated by G is defined as

$$L(G) = \{w \in \Sigma^* | \exists u \in A \ [u \Longrightarrow^* w]\}.$$

By $POST$, we denote the family of languages generated by Post Normal systems. This kind of system is computationally universal, as shown by:

Lemma 2 (Post[7]) $POST = \mathcal{RE}$.

We introduce two restricted models of Post systems. A *Regular Post system* (RP system) is a Post system $G = (V, \Sigma, P, A)$ with only productions of the form $uX \to wX$. Finally, a RP system $G = (V, \Sigma, P, A)$ is *pure* when $V = \Sigma$. RP systems and even pure RP systems are enough to characterize the class of regular languages:

Lemma 3 (Salomaa[8]) *Both the class of languages generated by RP systems and the class of languages generated by pure RP systems are equal to* $\mathcal{REG}$.

3.1 Generalized Regular Post Systems

We introduce a variant of a PN system which is in fact an extended model of an RP system. A *Generalized Regular Post system* (GRP system) is a quadruple $G = (V, \Sigma, P, A)$, where V and Σ are finite alphabets such that $\Sigma \subseteq V$, and P is a finite set of rules whose forms are either $uX \to wX$ or $aX \to Xb$ (X: unique variable not in V, $u, w \in V^*$, $a, b \in V$). $A (\subset V^+)$ is a finite set of *axioms*. Given strings $\alpha, \beta \in V^*$, a binary relation $\Longrightarrow$ is defined as follows:

$$\alpha \Longrightarrow \beta \qquad \text{iff} \qquad \begin{array}{l} \text{either } \ \exists uX \to wX \in P \ \text{ and } \ \exists \delta \in V^* \ \ [\alpha = u\delta, \beta = w\delta] \\ \text{or } \ \exists aX \to Xb \in P \ \text{ and } \ \exists \delta \in V^* \ \ [\alpha = a\delta, \beta = \delta b]. \end{array}$$

The reflexive and transitive closure of $\Longrightarrow$ is denoted by $\Longrightarrow^*$.

For a given G, a language generated by G is defined as

$$L(G) = \{w \in \Sigma^* | \exists u \in A \ [u \Longrightarrow^* w]\}.$$

By $GRPS$, we denote the family of languages generated by Generalized Regular Post systems.

This kind of system is universal, as shown by:

Lemma 4 (Yokomori et al.[9]) $POST = GRPS = \mathcal{RE}$.

Remark 1 *Actually, it is shown by Yokomori et al.* [9] *that it is enough to consider only GRP systems in the following* 2-restricted *form:* $G = (V \cup \overline{V}, \Sigma, P, A)$, *with* $\overline{V} = \{\overline{b} \mid b \in V\}$, $\Sigma \subseteq V, A \subset V^+$, *and productions of the form* $uX \to wX$ *or* $\overline{b}X \to Xb$, *with* $|u|, |w| \leq 2$ $(u, w \in V^*)$ *and* $b \in V$.

3.2 CF Post Systems

These paragraphs are devoted to prove a new result concerning how to characterize in terms of GRP systems the class of context-free languages. We start from the definition of a specific kind of Post system. A *CF Post system* is a GRP system $G = (V, \Sigma, P, A)$, with all the production rules of the form $BX \to CDX$ or $BX \to Xb$, where $B, C, D \in V - \Sigma$ and $b \in \Sigma$.

Theorem 5 *For a given CF grammar $G = (N, \Sigma, P, S)$ in Chomsky normal form, construct a CF Post system $G' = (N \cup \Sigma, \Sigma, P', \{S\})$ having*

$$P' \quad = \quad \{AX \to BCX \mid A \to BC \in P\} \cup \{AX \to Xa \mid A \to a \in P\}$$

Then, we have $L(G) = L(G')$.

Proof. Any w in the context-free language $L(G)$ has a successful left-most derivation $d(w)$ in G, and by the nature of the Post system G' only such $d(w)$ can be successfully realized by G'. This implies that $L(G) \subseteq L(G')$.

For any w' in $L(G')$, the successful derivation of w' in G' is corresponding to some left-most derivation in G: $w' \in L(G)$, i.e., $L(G') \subseteq L(G)$. □

In this case, we can see how a GRP system operates compared to a grammar: in a GRP system the rewritings occur only on the *prefix* of the string being transformed, while, in general, in a grammar the rewriting can occur also inside the string being generated. This leads, in the case of CF Post systems, to the need of a leftmost derivation to generate the same string generated by the grammar. This issue has relevance when considering EH systems, because in this model we have splicing rules that can be applied also inside the string, but they act splitting and joining strings, not performing easily the substitution that grammars do.

Theorem 6 *CF Post systems exactly characterize the class $\mathcal{CF}$.*

Proof. By Theorem 5, we have only to prove that any CF Post system $G = (V, \Sigma, P, A)$ can only generate a context-free language. Without loss of generality we can restrict to $A = \{S\}$. To do this, consider to construct from G a push-down automaton as follows:

$$\begin{aligned} M \quad = \quad & (\{p\}, V, \Sigma, d, p, S, \{p\}) \text{ where} \\ & d(p, a, A) = (p, \lambda) \text{ if } AX \to Xa \in P \\ & d(p, \lambda, A) = (p, BC) \text{ if } AX \to BCX \in P \end{aligned}$$

Then, for any $A \in V - \Sigma$, it is easy to see that $A \Rightarrow_n w \in L(G)$ iff $(p, w, A) \vdash_n (p, \lambda, \lambda)$, by induction on n, and supposing that M accepts in the empty stack mode. Then, $L(G)$ is context-free. □

4 EH Systems: Basic Tools and Normal Forms

We describe here some formal *Basic Tools* to be used to build in a modular way any EH system we need, starting from a given GRP system $G = (V_G, \Sigma, P_G, A_G)$. We also discuss their correctness, and we will apply them to the main theorems of this paper, to give two kinds of contributions:

- simpler proofs for the statements $\mathcal{RE} \subseteq \mathrm{EH}(m\mathcal{FIN}, \mathcal{FIN})$ and $\mathcal{REG} \subseteq \mathrm{EH}(\omega\mathcal{FIN}, \mathcal{FIN})$

- normal forms for EH systems generating recursively enumerable, regular or context-free languages.

We will show that the family of EH systems that we can build using only axioms and rules specified using these Basic Tools, can indeed generate any language in $\mathcal{RE}$, if we use also multiplicity constraints. Without multiplicity, it can generate only $\mathcal{REG}$.

The idea is to reproduce in the splicing system the transformations operated by a given Post system. The strings w produced in the Post system will be produced as $\mathbf{z_h} w \mathbf{z_t}$ in the splicing system with the head marked by the prefixed nonterminal $\mathbf{z_h}$, and the tail marked by the suffixed nonterminal $\mathbf{z_t}$. Eventually, the splicing system will apply some specific rules to remove these two markers from any string, so as to allow to produce strings in Σ^*. The nonterminals $\mathbf{z_1}$, $\mathbf{z_2}$ and $\mathbf{z_3}$ are used one in each basic building tool we are going to define. A distinguished set of (nonterminal) symbols V' is required by one of these three Basic Tools. We may take V' as the set of labels for P_G.

The EH systems specified in this way operate on the set of symbols V, union of the following four disjoint subsets: $V_G, \{\mathbf{z_h}, \mathbf{z_t}\}, \{\mathbf{z_1}, \mathbf{z_2}, \mathbf{z_3}\}, V'$(the set of labels for P_G). $\Sigma \subseteq V_G$ is the set of terminals to be considered in the definition of the generated language.

We describe each Basic Tool in terms of a set of axioms and a set of rules,

- $A_{hh}(u, w)$ and $R_{hh}(u, w)$, respectively, for the *Basic Tool* 1, to recall a $\underline{\text{h}}$ead to $\underline{\text{h}}$ead substitution from string u to string w, corresponding to a rule $uX \rightarrow wX$ in a GRP system

- $A_{rs}(a, b, c)$ and $R_{rs}(a, b, c)$ for the *Basic Tool 2*, to recall a $\underline{\text{r}}$otation and then the $\underline{\text{s}}$ubstitution of symbol a to symbol b, performed using some auxiliary nonterminal $c \in V'$ unique to the corresponding rule $aX \rightarrow Xb$

- A_{dr} and R_{dr}, for the *Basic Tool* 3, to recall the $\underline{\text{dr}}$opping of head and tail markers from any string.

In the following three descriptions of basic splicing tools, it will hold $s, t \in V^*$.

$\boxed{\textbf{Basic Tool 1}}$ $(A_{hh}(u,w), R_{hh}(u,w))$:

For any production of the form $uX \rightarrow wX$, we can use the following

$$\begin{array}{lll} A_{hh}(u,w) = & \{Z_h w Z_1\} & \text{(head subs.)} \\ R_{hh}(u,w) = & \{Z_h u \# \alpha \$ Z_h w \# Z_1 \mid \alpha \in V - \{Z_1, Z_2, Z_3\}\} & \text{(head subs.)} \end{array}$$

axiom and rules respectively. This gives the splicing:

- (head subs.) $(Z_h u \vdots \alpha s, Z_h w \vdots Z_1) \vdash_{Z_h u \# \alpha \$ Z_h w \# Z_1} (Z_h u Z_1, Z_h w \alpha s)$

We see that we generate as garbage string: $\{Z_h u Z_1\}$

$\boxed{\textbf{Basic Tool 2}}$ $(A_{rs}(a,b,c), R_{rs}(a,b,c))$:

For any production of the form $aX \rightarrow Xb$, and for its label symbol $c \in V'$, we can use the following axioms:

$$\begin{array}{rlll} A_{rs}(a,b,c) = & & \{Z_h c Z_2\} & \text{(head subs.)} \\ & \cup & \{Z_2 \delta c Z_2 \mid \delta \in V - (V' \cup \{Z_1, Z_2, Z_3\})\} & \text{(split)} \\ & \cup & \{Z_2 b Z_t\} & \text{(tail subs.)} \end{array}$$

and rules:

$$\begin{array}{rlll} R_{rs}(a,b,c) = & & \{Z_h a \# \alpha \$ Z_h c \# Z_2 \mid \alpha \in V - \{Z_1, Z_2, Z_3\}\} & \text{(head subs.)} \\ & \cup & \{\alpha c \delta \# \beta \$ Z_2 \delta c \# Z_2 \mid \alpha, \beta \in V - \{Z_1, Z_2, Z_3\}, & \\ & & \delta \in V - (V' \cup \{Z_1, Z_2, Z_3\})\} & \text{(split)} \\ & \cup & \{\alpha \# c \delta Z_2 \$ Z_2 \# \delta c \beta \mid \alpha, \beta \in V - \{Z_1, Z_2, Z_3\}, & \\ & & \delta \in V - (V' \cup \{Z_1, Z_2, Z_3\})\} & \text{(join)} \\ & \cup & \{\alpha \# c Z_t \$ Z_2 \# b Z_t \mid \alpha \in V - \{Z_1, Z_2, Z_3\}\} & \text{(tail subs.)} \end{array}$$

This gives the following splicings:

- (head subs.) $(Z_h a \vdots \alpha s, Z_h c \vdots Z_2) \vdash_{Z_h a \# \alpha \$ Z_h c \# Z_2} (Z_h a Z_2, Z_h c \alpha s)$

- (split) $(\underline{s \alpha c \delta \vdots \beta t}, Z_2 \delta c \vdots Z_2) \vdash_{\alpha c \delta \# \beta \$ Z_2 \delta c \# Z_2} (s \alpha c \delta Z_2, Z_2 \delta c \beta t)$

- (join) $(s \alpha \vdots c \delta Z_2, Z_2 \vdots \delta c \beta t) \vdash_{\alpha \# c \delta Z_2 \$ Z_2 \# \delta c \beta} (\underline{s \alpha \delta c \beta t}, Z_2 c \delta Z_2)$

- (tail subs.) $(s \alpha \vdots c Z_t, Z_2 \vdots b Z_t) \vdash_{\alpha \# c Z_t \$ Z_2 \# b Z_t} (s \alpha b Z_t, Z_2 c Z_t)$

We see that we generate as garbage strings: $\{Z_h a Z_2, Z_2 c \delta Z_2, Z_2 c Z_t\}$

The way Basic Tool 2 operates needs a brief explanation. We first substitute in the head of the string a with some special symbol c. Then we move c toward the tail of the string step by step, swapping it with any symbol not from V' appearing at its right. To do this with a splicing system, not able to directly substitute substrings, but only cutting and joining, we do a first "split" splicing to separate the string in two parts. Then, these same two parts are pasted with the "join" splicing, resulting in the same starting single string, but with the two symbols swapped (c and δ in the two strings underlined in the previous description).

$\boxed{\textbf{Basic Tool 3}}$ (A_{dr}, R_{dr}):

To produce a string s from any string $\mathtt{Z_h} s \mathtt{Z_t}$ we can use the following

$$\begin{array}{rll} A_{dr} = & \{\mathtt{Z_3}\} & \text{(drop)} \\ R_{dr} = & \{\mathtt{Z_h}\#\$\#\mathtt{Z_3}\} & \text{(drop head)} \\ \cup & \{\mathtt{Z_3}\#\$\#\mathtt{Z_t}\} & \text{(drop tail)} \end{array}$$

axiom and rules respectively. This gives these splicings:

- (drop head) $(\mathtt{Z_h}\vdots s, \vdots\mathtt{Z_3}) \vdash_{\mathtt{Z_h}\#\$\#\mathtt{Z_3}} (\mathtt{Z_h}\mathtt{Z_3}, s)$
- (drop tail) $(\mathtt{Z_3}\vdots, s\vdots\mathtt{Z_t}) \vdash_{\mathtt{Z_3}\#\$\#\mathtt{Z_t}} (\mathtt{Z_3}\mathtt{Z_t}, s)$

We see that we generate as garbage strings: $\{\mathtt{Z_h Z_3}, \mathtt{Z_3 Z_t}, \mathtt{Z_h Z_3 Z_t}\}$ (the last one generated by successive splicing steps)

Now we can observe that the splicing rules introduced by these Basic Tools have some nice properties, even with some biological interest. We only point out here that in $R_{rs}(a,b,c)$ and R_{dr} they specify only splicings occurring between two splicing sites of equal length, and this length goes from 1 to 4. Something similar happens in real DNA splicing processes, were the restriction enzymes define sites of length 6 mostly.[10]

Other important properties are related to the interactions between these sets of axioms and rules, when put together to build a splicing system. We sketch here the proofs of some:

Lemma 7 (Inter-tool lemma) *There is no interaction between axioms or garbage strings produced by one Basic Tool, with rules from a different Basic Tool.*

This is easily proved by noting that all the rules require the presence of the specific marker $\mathtt{Z}_i, 1 \le i \le 3$.

Lemma 8 (Rotation lemma) *As long as only one string* $\mathtt{Z_h} sct \mathtt{Z_t}$ *is present in the splicing system, then the iterated application of* $R_{rs}(a,b,c)$ *gives* $\mathtt{Z_h} stb \mathtt{Z_t}$.

This is the point where a constraint on the multiplicity of strings is needed. Otherwise, one could use the "join" splicing rule on a pair of strings coming from two different "split" splicings, eventually producing a string not produced by the GRP system being reproduced.

Lemma 9 (Head-substitution lemma) *If there's any string* $\mathtt{Z_h}us$ *in the splicing system, then one application of* $R_{hh}(u,w)$ *gives the string* $\mathtt{Z_h}ws$.

This is true also for GRP systems where we have pairs of productions like $uX \to (u'v)X$,$u'X \to wX$, even if in this case the interaction between axioms and rules corresponding to the two productions produces a new garbage string $\mathtt{Z_h}wv\mathtt{Z_1}$, by splicing the garbage string of the first production with rule and axiom of the second production. Finally, we have the simple lemma

Lemma 10 (Trimming lemma) R_{dr} *can produce the string* s *from* $\mathtt{Z_h}s\mathtt{Z_t}$.

All these properties allow us to easily state here the first main application of these Basic Tools. We show that we build a kind of normal form EH system for any language in $\mathcal{RE}$. A *k-limited* EH system is an EH system where any splicing rule $u_1\#u_2\$u_3\#u_4$ has the property: $|u_1|+|u_2| \le k, |u_3|+|u_4| \le k$.

Theorem 11 *Any recursively enumerable language can be generated by a 4-limited EH system with multiplicity.*

Proof. By Lemma 4 and the accompanying remark, it is sufficient to show how to build an EH system, with multiplicity, generating the same language of a given 2-restricted GRP system $G = (V_G, \Sigma, P, A_G)$. Without loss of generality consider $A_G = \{S\}$.

Then we build the EH system with multiplicity $H = (V, \Sigma, A, R)$ with:

- $V = V_G \cup V'_G \cup \{\mathtt{Z_h}, \mathtt{Z_t}, \mathtt{Z_1}, \mathtt{Z_2}, \mathtt{Z_3}\}, where V'_G = \{a_b \mid a, b \in V_G\}$
- $A = \{\mathtt{Z_h}S\mathtt{Z_t}\} \cup A_{dr} \cup (\cup_{uX\to wX \in P} A_{hh}(u,w)) \cup (\cup_{aX\to Xb\in P} A_{rs}(a,b,a_b))$, where $\mathtt{Z_h}S\mathtt{Z_t}$ has multiplicity 1, and the others have multiplicity ∞,
- $R = R_{dr} \cup (\cup_{uX\to wX\in P} R_{hh}(u,w)) \cup (\cup_{aX\to Xb\in P} R_{rs}(a,b,a_b))$.

First we observe that we can reproduce any Post derivation of G in the splicing system, eventually removing the head and tail markers to have the correct strings in Σ^*. Then $L(G) \subseteq L(H)$.

Moreover, any string in Σ^* produced by H cannot come from a garbage string, because they contain some $\mathtt{Z}_i, 1 \le i \le 3$. We are allowed to apply here the previous lemmas, having multiplicity one on the starting axiom $\mathtt{Z_h}S\mathtt{Z_t}$, and having chosen carefully the special auxiliary symbols for the Basic Tool 2. Then, it is proved that $L(G) \supseteq L(H)$.

It is easy to check that in each $R_{hh}(u,w)$ this length is bounded by $2 + \max(|u|,|w|)$, then, if we start from a 2-restricted GRP system, we are building a 4-limited EH system. □

We can state specific characterization results for the family of EH systems being described here:

Corollary 12 *A language can be generated by an EH system with multiplicity, built by Basic Tools* 1, 2 *and* 3 *iff it is recursively enumerable.*

Proof. (Sketch) By the Turing/Church thesis and Theorem 11. □

5 A Simple Form for $\mathcal{REG}$

Dealing with regular languages allows us to build much simpler EH systems.

Theorem 13 *Any regular language can be generated by an EH system without multiplicity, using only Basic Tools* 1 *and* 3 *("head subs." and "drop"), and with only* 4 *nonterminal symbols.*

Proof. By Lemma 3, it is sufficient to show how to build an EH system generating the same language of a given (pure) RP system.

Having only productions of the form $uX \to wX$, we will use only the Basic Tools for "head subs." and "drop". This also means that we don't need multiplicity constraints on the axioms, which is only required by Basic Tool 2.

Moreover, if the given RP system is pure, we can build an EH system of the simple form $H = (\Sigma \cup \{\mathtt{Z_h}, \mathtt{Z_t}, \mathtt{Z_1}, \mathtt{Z_3}\}, \Sigma, A, P)$. □

We can state specific characterization results for the sub-family of EH systems being described in the previous lemma:

Theorem 14 *A language can be generated by an EH system using only Basic Tools* 1 *and* 3 *iff it is regular.*

Proof. By known results [1,6] concerning the regularity of splicing systems H without nonterminals, the definition of $L(EH) = L(H) \cap \Sigma^*$, and the closure property of the class of regular languages, we know that any language generated by an EH system is regular. Conversely, by Theorem 13 we know how to find, for each regular language, an EH system generating the same language. □

6 A subclass of EH Systems for CF Languages

This section deals with a level of the Chomsky hierarchy seldom considered in the studies of the languages generated by extended splicing systems. Concerning context-free languages, only some closure properties of families like EH($\omega\mathcal{CF}, \mathcal{FIN}$) has been recently studied.[5] Here we try to give a characterization, in terms of EH systems with finite set of axioms and finite set of rules, and with multiplicity, for $\mathcal{CF}$.

Let's consider to generate any context-free language by use of a CF Post system, as discussed in Section 3.2. We aim at performing the productions of the Post system, which we have proved can be restricted to be of the form $AX \rightarrow BCX$ or $AX \rightarrow Xa$, using the Basic Tools previously defined.

Theorem 15 *Any context-free language on Σ^* can be generated by an EH system with multiplicity, built by Basic Tools of the simplified form (X, Y, Z, X_y symbols not in Σ, y symbol in Σ):*

$$\begin{array}{lll} A_{hh}(X, YZ) & A_{rs}(X, y, X_y) & A_{dr} \\ R_{hh}(X, YZ) & R_{rs}(X, y, X_y) & R_{dr} \end{array}$$

Proof. Using the characterization of $\mathcal{CF}$ in terms of CF Post systems, given in Theorem 5, we know we need Basic Tool 2, because we have to represent also Post productions of the form $AX \rightarrow Xa$. The use of Basic Tool 2 requires constraints on the multiplicity of the axioms of the splicing system we build. It is possible to completely specify the EH splicing system H starting from a CF Post system G, and to prove $L(H) = L(G)$, in the same way as done in Theorem 11; then we have only to observe that given the restriction on the structure of CF Post systems, the Basic Tools used will be of the simple form described in the statement we are proving. □

Unfortunately, it is not easy to show whether the family of EH systems of the form described in the previous theorem generates only context-free languages, so to give a complete characterization of $\mathcal{CF}$ in terms of splicing systems. In contrast, it is not hard to see that any member of the family of EH systems just mentioned can be simulated by a linear-bounded Turing machine.[4] It would be nice, at least, to prove that this family of EH systems generates only a proper subset of the family of context-sensitive languages.

7 Conclusions

Using the classical result due to E. L. Post in formal language theory, we have proposed a new method for charactering the universal computability of extended H systems, and have shown some characterizations of regular and context-free languages in the framework of extended H systems with finite axioms and finite rules. In our analysis, it turned out that the universal computability can be realized by three Basic Tools of splicing operations, namely Basic Tool 1 for *substituting prefix*, Basic Tool 2 for *rotating from head to tail*, and Basic Tool 3 for *dropping end-markers*. And, we have shown that the class of regular (or context-free) languages is achieved by a simple combination from these three Basic Tools. Multiplicity is required only when rules from Basic Tool 2 (or rules of the form $aX \rightarrow Xb$) has to be performed.

Open problems: Is the special subclass of extended H systems introduced to represent the class $\mathcal{CF}$ in Theorem 15 exactly generating the class $\mathcal{CF}$?

Moreover, to simulate languages beyond $\mathcal{REG}$ is it necessary to put multiplicity constraints on axiom strings, in our current framework ?

Acknowledgements

This work was supported in part by a Joint JSPS/EC Grant No.94043 and Grants-in-Aid for Scientific Research No.94234 (for the first author), No.07780310 and No.07249201 (for the second author) from the Ministry of Education, Science and Culture, Japan.

References

1. K. Culik II and T. Harju. Dominoes and the regularity of DNA splicing languages. *Discrete Applied Mathematics*, 31:261–277, 1991.
2. R. Freund, L. Kari, and Gh. Paun. DNA computing based on splicing : The existence of universal computers. Technical report, Fachgruppe Informatik, Tech. Univ. Wien.
3. T. Head. Formal language theory and DNA : An analysis of the generative capacity of specific recombinant behaviors. *Bulletin of Mathematical Biology*, 49:737–759, 1987.
4. S.Y. Kuroda. Classes of languages and linear-bounded automata. *Information and Control*, 7:2, 207–223, 1964.
5. Gh. Paun, G. Rozenberg, and A. Salomaa. Computing by splicing. *submitted*, 1995.
6. D. Pixton. Regularity of splicing languages. *Discrete Applied Mathematics, (To appear)*, 1995.
7. E. L. Post. Formal reductions of the general combinatorial decision problem. *American Journal of Mathematics*, 65:197–215, 1943.
8. A. Salomaa. *Computation and Automata.* Cambridge University Press, 1985.
9. T. Yokomori, S. Kobayashi, C. Ferretti. On the power of circular splicing systems and DNA computability. Technical report, Dept. of Comput. Sci. and Inform. Math., Univ. of Electro-Commun., TR 95-01, 1995.
10. J.D. Watson, J. Tooze, D.T. Kurtz. *Recombinant DNA: a Short Course.* Freeman, New York, 1983.