Acta Informatica 35, 401–420 (1998)

DNA computing, sticker systems, and universality*

Lila Kari[1], **Gheorghe Păun**[2], **Grzegorz Rozenberg**[3], **Arto Salomaa**[4], **Sheng Yu**[1]

[1] Department of Computer Science, University of Western Ontario, London, Ontario, Canada N6A 5B7
[2] Institute of Mathematics of the Romanian Academy, P.O. Box 1 – 764, RO-70700 Bucureşti, Romania
[3] Department of Computer Science, Leiden University, P.O. Box 9512, 2300 RA Leiden, The Netherlands
[4] Academy of Finland and Turku University, Department of Mathematics, FIN-20500 Turku, Finland

Received: 10 October 1996 / 16 April 1997

Abstract. We introduce the *sticker systems*, a computability model, which is an abstraction of the computations using the Watson-Crick complementarity as in Adleman's DNA computing experiment, [1]. Several types of sticker systems are shown to characterize (modulo a weak coding) the regular languages, hence the power of finite automata. One variant is proven to be equivalent to Turing machines. Another one is found to have a strictly intermediate power.

1. Introduction

The *sticker systems* introduced here are language generating devices based on the *sticker operation*, which, in turn, is a model of the techniques used by L. Adleman in his successful experiment of computing a Hamiltonian path in a graph by using DNA, [1]. We recall some details of the experiment in order to see the roots of our models.

One knows that DNA sequences are in fact double stranded (helicoidal) structures composed of four nucleotides, A (adenine), C (cytosine), G (guanine), and T (thymine), paired A–T, C–G according to the so-called Watson-Crick complementarity. If we have a single stranded sequence of A, C, G, T nucleotides, together with a single stranded sequence composed of the complementary nucleotides, the two sequences will be "glued" together (by hydrogen bonds), forming a double stranded DNA sequence. Figure 1 illustrates this operation.

Using this biochemical reaction, Adleman has proceeded as follows, in searching Hamiltonian paths in a graph:

* Research supported by the Academy of Finland, project 11281, the Spanish Secretaria de Estado de Universidades e Investigacion, SAB95-0357, and by the National Sciences and Engineering Research Council of Canada, Grant OGP0041630

5′–AAACTGGAG–3′ + 3′–TTTGACCTC–5′

$\Longrightarrow$ AAACTGGAG
TTTGACCTC

Fig. 1.

- codify the nodes by single stranded DNA sequences of length 20 and put all these strings in a test tube,
- if the node i is codified by the string x_i and the node j is codified by the string x_j, and there is an arrow from node i to node j in the graph, then add to the test tube a single stranded DNA sequence y_{ij} such that if $x_i = x_i'x_i''$, $x_j = x_j'x_j''$, each of x_i', x_i'', x_j', x_j'' being strings of length 10, then $y_{ij} = y_{ij}'y_{ij}''$, where y_{ij}' is the Watson-Crick complement of x_i'' and y_{ij}'' is the Watson-Crick complement of x_j'.

Due to the complementarity, the string y_{ij} will match the corresponding parts of x_i and x_j, linking them and producing in this way a sequence of length 40, as illustrated by Fig. 2.

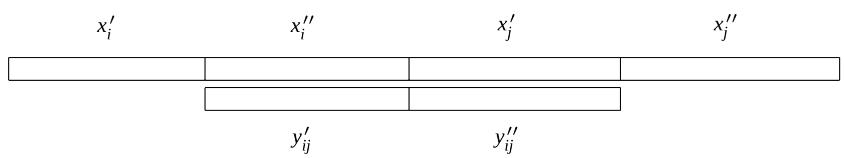

Fig. 2.

This “domino game” can continue, identifying longer and longer paths in the considered graph. By a filtering procedure which is not of interest here, one then can check whether or not paths with specified properties exist (for instance, Hamiltonian paths).

We extract from this experiment only the basic ingredient: the operation of prolonging to the right a sequence of (single or double) symbols by using given single stranded strings, matching them with portions of the current sequence according to a complementarity relation.

The formal model of this operation is the *sticker operation* defined in the following section.

This operation can be used in building a generative/computing device: start from a given set of incomplete double stranded sequences (axioms), plus two sets of single stranded complementary sequences. Iterating the right prolongation using elements of these latter sets, we get “computations” of possibly arbitrary length. Stop when a complete double stranded sequence is obtained, that is when no "sticky end" still exists. We obtain in this way a language.

The generative power of several variants of such mechanisms is investigated here. The unrestricted case corresponds to the Adleman experiment and it is proved to characterize – modulo a weak coding – the regular languages. When an additional restriction is imposed, namely to use the same sequence of complementary strings from the two initial sets, then, rather surprisingly, we get a characterization of recursively enumerable languages. Whether or not such a restriction can be implemented in the DNA framework is a practical problem which we cannot answer, but providing that it can be done, computationally universal DNA "computers" could be designed just using the Watson-Crick complementarity, plus the techniques required in the mentioned restriction.

This reminds us the results obtained in a series of papers (see references in [10], [13], [16]) about the possibility of designing universal (and programmable) DNA "computers" based on the operation of *splicing*, introduced in [9] as a model of the recombinant behavior of DNA under the influence of restriction enzymes and ligases.

2. The sticker operation

Let V be an alphabet (a finite set of abstract symbols) endowed with a symmetric relation ρ (of *complementarity*), $\rho \subseteq V \times V$. Let # be a special symbol not in V, denoting an empty space (the *blank* symbol).

Using the elements of $V \cup \{\#\}$ we construct the *composite* symbols of the following sets:

$$\begin{pmatrix} V \\ V \end{pmatrix}_\rho = \left\{ \begin{pmatrix} a \\ b \end{pmatrix} \mid a, b \in V, (a, b) \in \rho \right\},$$
$$\begin{pmatrix} \# \\ V \end{pmatrix} = \left\{ \begin{pmatrix} \# \\ a \end{pmatrix} \mid a \in V \right\},$$
$$\begin{pmatrix} V \\ \# \end{pmatrix} = \left\{ \begin{pmatrix} a \\ \# \end{pmatrix} \mid a \in V \right\}.$$

We denote

$$W_\rho(V) = \begin{pmatrix} V \\ V \end{pmatrix}_\rho^* S(V),$$

where

$$S(V) = \begin{pmatrix} \# \\ V \end{pmatrix}^* \cup \begin{pmatrix} V \\ \# \end{pmatrix}^*,$$

and we call the elements of this set *well-started sequences* (in general, X^* is the set of all strings, including the empty one denoted by λ, composed of elements of X, and X^+ is the set $X^* - \{\lambda\}$). Stated otherwise, the elements of $W_\rho(V)$ start with pairs of symbols in V, as selected by the complementarity relation, and end either by a suffix consisting of pairs $\binom{\#}{a}$ or with a suffix consisting of pairs $\binom{b}{\#}$, for $a, b \in V$ (the symbols $\binom{\#}{a}$, $\binom{b}{\#}$ are not mixed).

The sticker operation, denoted by μ, is a partially defined mapping from $W_\rho(V) \times S(V)$ to $W_\rho(V)$, defined as follows. For $x \in W_\rho(V), y \in S(V), z \in W_\rho(V)$, we write

$$\mu(x, y) = z$$

if and only if one of the following cases holds:

1. $x = \binom{a_1}{b_1} \dots \binom{a_k}{b_k} \binom{a_{k+1}}{\#} \dots \binom{a_{k+r}}{\#} \binom{a_{k+r+1}}{\#} \dots \binom{a_{k+r+p}}{\#}$,
$y = \binom{\#}{c_1} \dots \binom{\#}{c_r}$,
$z = \binom{a_1}{b_1} \dots \binom{a_k}{b_k} \binom{a_{k+1}}{c_1} \dots \binom{a_{k+r}}{c_r} \binom{a_{k+r+1}}{\#} \dots \binom{a_{k+r+p}}{\#}$,
for $k \geq 0, r \geq 1, p \geq 1$,
$a_i \in V, 1 \leq i \leq k+r+p,\ b_i \in V, 1 \leq i \leq k,\ c_i \in V, 1 \leq i \leq r$,
and $(a_{k+i}, c_i) \in \rho, 1 \leq i \leq r$;

2. $x = \binom{a_1}{b_1} \dots \binom{a_k}{b_k} \binom{a_{k+1}}{\#} \dots \binom{a_{k+r}}{\#}$,
$y = \binom{\#}{c_1} \dots \binom{\#}{c_r} \binom{\#}{c_{r+1}} \dots \binom{\#}{c_{r+p}}$,
$z = \binom{a_1}{b_1} \dots \binom{a_k}{b_k} \binom{a_{k+1}}{c_1} \dots \binom{a_{k+r}}{c_r} \binom{\#}{c_{r+1}} \dots \binom{\#}{c_{r+p}}$,
for $k \geq 0, r \geq 0, p \geq 0, r+p \geq 1$,
$a_i \in V, 1 \leq i \leq k+r,\ b_i \in V, 1 \leq i \leq k,\ c_i \in V, 1 \leq i \leq r+p$,
and $(a_{k+i}, c_i) \in \rho, 1 \leq i \leq r$;

3. $x = \binom{a_1}{b_1} \dots \binom{a_k}{b_k} \binom{\#}{b_{k+1}} \dots \binom{\#}{b_{k+r}} \binom{\#}{b_{k+r+1}} \dots \binom{\#}{b_{k+r+p}}$,
$y = \binom{c_1}{\#} \dots \binom{c_r}{\#}$,
$z = \binom{a_1}{b_1} \dots \binom{a_k}{b_k} \binom{c_1}{b_{k+1}} \dots \binom{c_{k+r}}{b_{k+r}} \binom{\#}{b_{k+r+1}} \dots \binom{\#}{b_{k+r+p}}$,
for $k \geq 0, r \geq 1, p \geq 1$,
$a_i \in V, 1 \leq i \leq k,\ b_i \in V, 1 \leq i \leq k+r+p,\ c_i \in V, 1 \leq i \leq r$,
and $(c_i, b_{k+i}) \in \rho, 1 \leq i \leq r$;

4. $x = \binom{a_1}{b_1} \dots \binom{a_k}{b_k} \binom{\#}{b_{k+1}} \dots \binom{\#}{b_{k+r}}$,

$$y = \binom{c_1}{\#} \cdots \binom{c_r}{\#} \binom{c_{r+1}}{\#} \cdots \binom{c_{r+p}}{\#},$$

$$z = \binom{a_1}{b_1} \cdots \binom{a_k}{b_k} \binom{c_1}{b_{k+1}} \cdots \binom{c_r}{b_{k+r}} \binom{c_{r+1}}{\#} \cdots \binom{c_{r+p}}{\#},$$

for $k \geq 0, r \geq 0, p \geq 0, r + p \geq 1$,

$a_i \in V, 1 \leq i \leq k,\ b_i \in V, 1 \leq i \leq k + r,\ c_i \in V, 1 \leq i \leq r + p$,

and $(c_i, b_{k+i}) \in \rho, 1 \leq i \leq r$.

In case 1 we add complementary symbols on the lower level without completing all the blank spaces. In case 2 we complete the blank spaces on the lower level of x and possibly add more composite symbols of the form $\binom{\#}{c}$. Cases 3 and 4 are symmetric to cases 1 and 2, respectively, completing blank spaces on the upper level of the string.

Figure 3 picturally illustrates these cases.

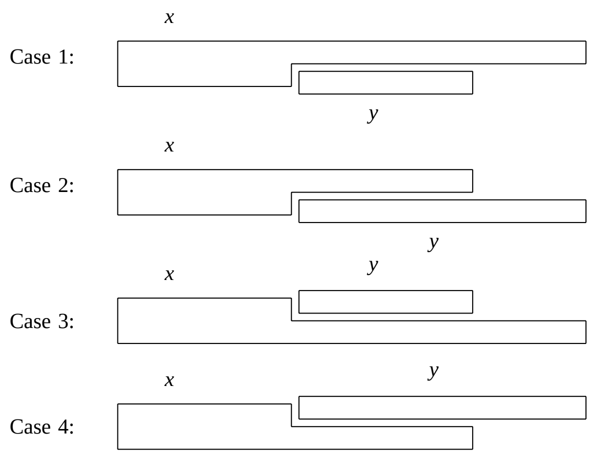

Fig. 3.

Note that in all cases the string y must contain at least one composite symbol and that cases 2 and 4 allow the prolongation of “blunt” strings in $W_\rho(V)$: when $r = 0$, there is no blank position in x.

Of course, for strings x, y which do not satisfy any of the previous conditions, $\mu(x, y)$ is not defined.

3. Sticker systems

Using the sticker operation we can define a generating/computing mechanism as follows:

A *sticker system* is a construct

$$\gamma = (V, \rho, A, B_d, B_u),$$

where V is an alphabet, $\rho \subseteq V \times V$ is a symmetric relation on V, A is a finite subset of $W_\rho(V)$ (of axioms), and B_d and B_u are finite subsets of $\binom{\#}{V}^+$ and $\binom{V}{\#}^+$, respectively.

The idea behind such a machinery is the following. We start with the sequences in A and we prolong them to the right with the strings in B_d, B_u according to the sticker operations (the elements of B_d are used on the lower row, *down*, and those of B_u are used on the *upper* row). When no blank symbol is present, we obtain a string over the alphabet $\binom{V}{V}_\rho$. The language of all such strings is the language generated by γ.

Formally, we define this language as follows.

For two strings $x, z \in W_\rho(V)$ we write

$$x \Longrightarrow z \quad \text{iff} \quad z = \mu(x, y) \text{ for some } y \in B_d \cup B_u.$$

We denote by $\Longrightarrow^*$ the reflexive and transitive closure of the relation $\Longrightarrow$.

A sequence $x_1 \Longrightarrow x_2 \Longrightarrow \ldots \Longrightarrow x_k, x_1 \in A$, is called a *computation* in γ (of length $k-1$). A computation as above is *complete* if $x_k \in \binom{V}{V}_\rho^*$ (no blank symbol is present in the last string of composite symbols).

The language generated by γ, denoted by $L(\gamma)$, is defined by

$$L(\gamma) = \{w \in \binom{V}{V}_\rho^* \mid x \Longrightarrow^* w,\ x \in A\}.$$

Therefore, only the complete computations are taken into account when defining $L(\gamma)$. Note that a complete computation can be continued since we allow prolongations starting from blunt sequences.

One sees the close resemblance with the operations used in the Adleman experiment: B_d corresponds to the codes of graph nodes, B_u corresponds to the complementary strings identifying the arrows in the graph (or conversely). The fact that we use here also a given set of axioms (and, in several results, a weak coding is applied to the language of words of composite symbols generated by our devices) adds flexibility to the model and makes it more similar to usual generating mechanisms investigated in formal language theory.

A complete computation $x_1 \Longrightarrow x_2 \Longrightarrow \ldots \Longrightarrow x_k, x_1 \in A, x_k \in \binom{V}{V}_\rho^*$, with respect to γ, is said to be:

- *primitive* if no properly initial part of it is complete;
- *balanced* if in each step $x_i \Longrightarrow x_{i+1}$ one uses a sticker operation corresponding to cases 2 or 4 in Sect. 2. Moreover, cases 2 and 4 alternate from a step to the next one.

Thus, in a primitive computation we do not use sticker operations as in cases 1 – 4 with $p = 0$, except in the last step. In a balanced computation we allow $p = 0$, but from a step to the next one we have to change the set B_d, B_u from which we take the string to be used.

Let us denote by $L_p(\gamma), L_b(\gamma), L_{pb}(\gamma)$ the languages of the strings $w \in \begin{pmatrix} V \\ V \end{pmatrix}^*_\rho$ obtained by a complete computation of γ that is primitive, balanced, both primitive and balanced, respectively.

Assume now that the strings in the sets B_d, B_u are labelled in a one-to-one manner by natural numbers from 1 to $\mathrm{card}(B_\alpha)$, $\alpha \in \{d, u\}$; denote by $e_\alpha : B_\alpha \longrightarrow \{1, \ldots, \mathrm{card}(B_\alpha)\}, \alpha \in \{d, u\}$, the labellings. For a computation

$$D : x_1 \Longrightarrow x_2 \Longrightarrow \ldots \Longrightarrow x_k,\ x_1 \in A, x_k \in \begin{pmatrix} V \\ V \end{pmatrix}^*_\rho,$$

and for $1 \leq j \leq k-1$, we denote

$$e_\alpha(x_j \Longrightarrow x_{j+1}) = \begin{cases} e_\alpha(y), & \text{if } x_{j+1} = \mu(x_j, y), y \in B_\alpha, \\ \lambda, & \text{otherwise,} \end{cases}$$

and we define

$$e_\alpha(D) = e_\alpha(x_1 \Longrightarrow x_2)e_\alpha(x_2 \Longrightarrow x_3)\ldots e_\alpha(x_{k-1} \Longrightarrow x_k),$$

for $\alpha \in \{d, u\}$. We say that $e_d(D)$ is the d-control word and $e_u(D)$ is the u-control word associated with D.

A computation D such that $e_d(D) = e_u(D)$ is called *coherent*. When $|e_d(D)| = |e_u(D)|$ (where $|x|$ is the length of the string x) we say that D is a *fair* computation.

We denote by $L_c(\gamma)$ and $L_f(\gamma)$ the languages of the strings in $\begin{pmatrix} V \\ V \end{pmatrix}^*_\rho$ that are obtained by a coherent complete computation and, respectively, by a fair complete computation in γ. Clearly, each coherent computation is also fair.

By the definition, $L_\alpha(\gamma) \subseteq L(\gamma)$, for all $\alpha \in \{p, b, pb, c, f\}$.

We denote by *SL*, *PSL*, *BSL*, *PBSL*, *CSL*, *FSL* the families of languages of the form $L(\gamma), L_p(\gamma), L_b(\gamma), L_{pb}(\gamma), L_c(\gamma), L_f(\gamma)$, respectively, defined as above. (By *REG* and *RE* we denote the families of regular and recursively enumerable languages, respectively.)

In the following sections we will investigate these six families of languages generated by sticker systems. We would also like to investigate coherent primitive, coherent balanced, coherent primitive and balanced languages, as well as fair primitive, fair balanced languages etc. (Note that a balanced computation is not necessarily a fair one, because it can start and stop in the same set B_d, B_u). But we will not consider them in this paper.

4. Characterizing the regular languages

We now begin our investigations concerning the generative capacity of sticker systems. We will first show in this section that many of the basic variants yield only regular languages. Then we show that each regular language can be represented as a weak coding of a language generated by a sticker system of one of these basic types. (A *weak coding* is a morphism $h : V_1^* \longrightarrow V_2^*$ such that $h(a) \in V_2 \cup \{\lambda\}$ for all $a \in V_1$. If $h(a) \in V_2$ for all $a \in V_1$, then h is called a *coding*.)

Lemma 1. $SL \subseteq REG$.

Proof. Let $\gamma = (V, \rho, A, B_d, B_u)$ be a sticker system. We denote

$$d = \max\{|x| \mid x \in A \cup B_d \cup B_u\}.$$

We construct the right-linear grammar $G = (N, T, S, P)$ with

$$\begin{aligned} N &= \{\Big[\binom{a_1}{\#}\dots\binom{a_k}{\#}\Big],\ \Big[\binom{\#}{a_1}\dots\binom{\#}{a_k}\Big] \mid a_i \in V, \\ &\qquad 1 \le i \le k, 1 \le k \le d\} \\ &\cup\ \{S, X\}, \\ T &= \binom{V}{V}_\rho, \end{aligned}$$

and P contains the following rules:

1.1) $S \to \binom{a_1}{b_1}\dots\binom{a_n}{b_n}\Big[\binom{a_{n+1}}{\#}\dots\binom{a_{n+k}}{\#}\Big]$,
for $\binom{a_1}{b_1}\dots\binom{a_n}{b_n}\binom{a_{n+1}}{\#}\dots\binom{a_{n+k}}{\#} \in A$,

1.2) $S \to \binom{a_1}{b_1}\dots\binom{a_n}{b_n}\Big[\binom{\#}{b_{n+1}}\dots\binom{\#}{b_{n+k}}\Big]$,
for $\binom{a_1}{b_1}\dots\binom{a_n}{b_n}\binom{\#}{b_{n+1}}\dots\binom{\#}{b_{n+k}} \in A$,

1.3) $S \to \binom{a_1}{b_1}\dots\binom{a_n}{b_n}X$,
for $\binom{a_1}{b_1}\dots\binom{a_n}{b_n} \in A$.

In all cases, $a_i, b_i \in V, 1 \le i \le n+k$, and $k \ge 1, n \ge 0$.

2.1) $\Big[\binom{a_1}{\#}\dots\binom{a_n}{\#}\Big] \to \binom{a_1}{b_1}\dots\binom{a_m}{b_m}\Big[\binom{a_{m+1}}{\#}\dots\binom{a_n}{\#}\Big]$,
for $\binom{\#}{b_1}\dots\binom{\#}{b_m} \in B_d$ with $m < n$,

2.2) $\Big[\binom{a_1}{\#}\dots\binom{a_n}{\#}\Big] \to \binom{a_1}{b_1}\dots\binom{a_n}{b_n}X$,
for $\binom{\#}{b_1}\dots\binom{\#}{b_n} \in B_d$,

2.3) $\left[\binom{a_1}{\#}\dots\binom{a_n}{\#}\right] \rightarrow \binom{a_1}{b_1}\dots\binom{a_n}{b_n}\left[\binom{\#}{b_{n+1}}\dots\binom{\#}{b_m}\right]$,
for $\binom{\#}{b_1}\dots\binom{\#}{b_m} \in B_d$ with $m > n$.

(We prolong the current terminal string of symbols $\binom{a}{b} \in \binom{V}{V}_\rho$ to the right, using an element of B_d.)

3.1) $\left[\binom{\#}{b_1}\dots\binom{\#}{b_n}\right] \rightarrow \binom{a_1}{b_1}\dots\binom{a_m}{b_m}\left[\binom{\#}{b_{m+1}}\dots\binom{\#}{b_n}\right]$,
for $\binom{a_1}{\#}\dots\binom{a_m}{\#} \in B_u$ with $m < n$,

3.2) $\left[\binom{\#}{b_1}\dots\binom{\#}{b_n}\right] \rightarrow \binom{a_1}{b_1}\dots\binom{a_n}{b_n}X$,
for $\binom{a_1}{\#}\dots\binom{a_n}{\#} \in B_u$,

3.3) $\left[\binom{\#}{b_1}\dots\binom{\#}{b_n}\right] \rightarrow \binom{a_1}{b_1}\dots\binom{a_n}{b_n}\left[\binom{a_{n+1}}{\#}\dots\binom{a_m}{\#}\right]$,
for $\binom{a_1}{\#}\dots\binom{a_m}{\#} \in B_u$ with $m > n$.

(We prolong the current terminal string of symbols $\binom{a}{b} \in \binom{V}{V}_\rho$ to the right, using an element of B_u.)
In all rules of types $2.i), 3.i), i = 1, 2, 3$, the symbols $\left[\binom{a_1}{\#}\dots\binom{a_n}{\#}\right]$ $\left[\binom{\#}{b_1}\dots\binom{\#}{b_n}\right]$, respectively, are arbitrary symbols in N.

4.1) $X \rightarrow \left[\binom{a_1}{\#}\dots\binom{a_n}{\#}\right]$, for $\binom{a_1}{\#}\dots\binom{a_n}{\#} \in B_u,\ n \geq 1$,

4.2) $X \rightarrow \left[\binom{\#}{b_1}\dots\binom{\#}{b_n}\right]$, for $\binom{\#}{b_1}\dots\binom{\#}{b_n} \in B_d,\ n \geq 1$.
(A complete computation can be continued using these rules.)

5) $X \rightarrow \lambda$.

It is easy to see that $L(G) = L(\gamma)$: at every step we can use an element of B_d or an element of B_u such that the current sticky end is shorter than d. Therefore, the terminals in N can control the process in the same way as the sticky ends. We conclude that $L(\gamma)$ is a regular language. □

Lemma 2. *PSL* $\subseteq$ *REG*.

Proof. Starting from a sticker system γ, we construct a right-linear grammar G' as in the previous proof, but without using the nonterminal symbol X (this means that the rules of types 1.3), 2.2), and 3.2) become terminal rules, and the rules of types 4.1), 4.2), and 5) are no longer used). In this way, no complete computation in γ can be continued by the corresponding derivation in G, that is $L(G) = L_p(\gamma)$. Consequently, $L_p(\gamma) \in REG$. □

Lemma 3. $BSL \subseteq REG$.

Proof. We proceed as in the proof of Lemma 1, but instead of using one symbol X we consider two nonterminals X_u, X_d. Then we introduce rules of type 1.3) with both X_u and X_d instead of X, in rules of type 2.2) we replace X with X_d, in rules of type 3.2) we replace X with X_u, in rules of type 4.1) we replace X with X_u, and in rules of type 4.2) we replace X with X_d; moreover, the rules of types 2.1) and 3.1) are removed; finally, instead of $X \to \lambda$ we introduce both rules $X_d \to \lambda$ and $X_u \to \lambda$.

In this way, only balanced computations in γ are simulated in the obtained grammar. Denoting this grammar by G'', we obtain $L(G'') = L_b(\gamma)$. Therefore, $L_b(\gamma) \in REG$. □

Combining the ideas of the proofs of Lemmas 2 and 3 we get:

Lemma 4. $PBSL \subseteq REG$.

Modulo a weak coding, the opposite inclusions are also true.

Lemma 5. *Every regular language can be represented as a weak coding of a language in* $SL \cap PSL \cap BSL \cap PBSL$.

Proof. Consider a regular grammar $G = (N, T, S, P)$, assume it λ-free (at most the λ-rule $S \to \lambda$ is present and then S does not appear in the right hand side of the rules), and construct the sticker system

$$\gamma = (V, \rho, A, B_d, B_u),$$

with

$$\begin{aligned}
V &= \{[X,a]_i \mid X \in N, a \in T, i = 1,2\} \\
&\cup \{(X,a)_i \mid X \in N, a \in T, i = 1,2\} \\
&\cup \{[Z,\cdot], (Z,\cdot)\}, \quad \text{where } Z \text{ is a new symbol}, \\
\rho &= \{([X,a]_i, (X,a)_i) \mid X \in N, a \in T, i = 1,2\} \\
&\cup \{([Z,\cdot], (Z,\cdot))\}, \\
A &= \left\{ \begin{pmatrix} [S,a]_1 \\ \# \end{pmatrix} \mid S \to aX \in P, a \in T, X \in N \right\} \\
&\cup \left\{ \begin{pmatrix} [S,a]_1 \\ (S,a)_1 \end{pmatrix} \mid S \to a \in P, a \in T \right\} \\
&\cup \{\lambda \mid S \to \lambda \in P\}, \\
B_d &= \left\{ \begin{pmatrix} \# \\ (X,a)_1 \end{pmatrix} \begin{pmatrix} \# \\ (Y,b)_2 \end{pmatrix} \mid X \to aY \in P, \text{ and } \; Y \to bY' \in P \right. \\
&\qquad\qquad \left. \text{or } Y \to b \in P, \; X, Y, Y' \in N, a, b \in T \right\} \\
&\cup \left\{ \begin{pmatrix} \# \\ (X,a)_1 \end{pmatrix} \begin{pmatrix} \# \\ (Z,\cdot) \end{pmatrix} \mid X \to a \in P, a \in T \right\} \\
&\cup \left\{ \begin{pmatrix} \# \\ (Z,\cdot) \end{pmatrix} \begin{pmatrix} \# \\ (Z,\cdot) \end{pmatrix} \right\},
\end{aligned}$$

$$
\begin{aligned}
B_u \quad = \quad & \left\{ \binom{[X,a]_2}{\#} \binom{[Y,b]_1}{\#} \mid X \to aY \in P, \text{ and } Y \to bY' \in P \right. \\
& \qquad \left. \text{or } Y \to b \in P,\ X, Y, Y' \in N, a, b \in T \right\} \\
\cup \quad & \left\{ \binom{[X,a]_2}{\#} \binom{[Z,\cdot]}{\#} \mid X \to a \in P, a \in T \right\} \\
\cup \quad & \left\{ \binom{[Z,\cdot]}{\#} \right\}.
\end{aligned}
$$

Every computation has to start by using a string in B_d, it continues by alternately using elements of B_d and B_u, and can be completed only by using the string $\binom{[Z,\cdot]}{\#}$ in B_u. A complete computation cannot continue further by using strings that contain symbols other than $[Z,\cdot]$ or $(Z,\cdot)$, because the relation ρ allows only the matching of symbols $[X,a]_i, (X,a)_j$ with $i = j$.

Consider now the weak coding $g : \binom{V}{V}^*_\rho \longrightarrow T^*$ defined by

$$
\begin{aligned}
& g(\binom{[X,a]_i}{(X,a)_i}) = a, \text{ for } X \in N, a \in T, i = 1, 2, \\
& g(\binom{[Z,\cdot]}{(Z,\cdot)}) = \lambda.
\end{aligned}
$$

From the construction of γ and the definition of g one can easily see that $L(G) = g(L(\gamma)) = g(L_\alpha(G))$, for all $\alpha \in \{p, b, pb, f\}$. $\square$

For the case of primitive computations, the proof above can be modified in such a way to have $L(G)$ equal to a coding of $L_p(\gamma)$: if we remove all occurrences of symbols $\binom{\#}{(Z,\cdot)}, \binom{[Z,\cdot]}{\#}$, then the computation stops when completing an element of $\binom{V}{V}^*_\rho$. A computation simulating a derivation in G is also balanced, hence a coding also suffices for the case of primitive and balanced computations.

In the non-primitive case we cannot avoid using symbols $\binom{\#}{(Z,\cdot)}, \binom{[Z,\cdot]}{\#}$ (hence we cannot avoid using a weak coding in the statement of Lemma 5), because otherwise we can continue a computation corresponding to a derivation in G by adding further symbols $\binom{[X,a]_i}{(X,a)_i}$, $i = 1, 2$; such symbols cannot be removed even if we then use a weak coding.

For a family F of languages, we denote by $wcode(F)$ the family of languages of the form $g(L)$, for $L \in F$ and g a weak coding.

Theorem 1. $REG = wcode(SL) = wcode(PSL) = wcode(BSL) = wcode(PBSL)$.

Proof. The family REG is closed under arbitrary morphisms, hence from Lemmas 1, 2, 3, 4 we obtain $wcode(F) \subseteq REG$, for $F \in \{SL, PSL, BSL, BPSL\}$. Lemma 5 proves the opposite inclusions. $\square$

Thus, the Adleman way of computing cannot transgress the power of finite automata.

5. Characterizing the recursively enumerable languages

We are now ready to give our main result: the family *CSL* is computationally universal, in the sense that $wcde(CSL) = RE$. Our proof consists of two steps, one being a modification of the classical proof of the characterization of recursively enumerable languages by means of equality sets and, the other, a specific construction with sticker systems. The essence of our proof can be described as follows. The notion of coherence comes very close to the idea of the twin-shuffle languages [19]. Hence, the generative capacity of the latter can be carried over to sticker systems. We now begin the details.

From the definitions, it is clear that all languages generated by sticker systems are context-sensitive. Moreover, from the Turing-Church thesis we have the following lemma:

Lemma 6. $wcode(CSL) \subseteq RE$.

In view of the fact that, at the first sight, the operation of prolongation to the right based on matching symbols related by a complementarity relation does not look very powerful, the following result is rather surprising.

Lemma 7. *Every recursively enumerable language can be represented as a weak coding of a language in the family CSL.*

Our proof of this lemma is based on the following representation of an arbitrary recursively enumerable language $L \subseteq T^*$:

$$L = h_T(h_1(E(h_1, h_2)) \cap R). \tag{1}$$

where h_1 and h_2 are two morphisms, R is a regular language, $E(h_1, h_2)$ is the equality set of h_1 and h_2, and h_T is a special projective morphism defined by

$$h_T(a) = \begin{cases} a, & \text{if } a \in T; \\ \lambda, & \text{if } a \notin T. \end{cases}$$

A representation which is very similar to the above has been shown in [18], [19]:

$$L = h_T(E(h_1, h_2) \cap R). \tag{2}$$

The difference between (1) and (2) is that (1) uses the h_1 image of the equality set of h_1 and h_2, i.e., $h_1(E(h_1, h_2))$ $(= h_2(E(h_1, h_2)))$, but (2) uses the equality set itself, i.e., $E(h_1, h_2)$. Unfortunately, we have found neither a proof for (1) in the literature nor a way to derive (1) from (2) directly. We give a proof for (1) in the following, which is a modification of the proof for (2) in [19] (Theorem 6.9).

Lemma 8. *For each recursively enumerable language $L \subseteq T^*$, there exist two λ-free morphisms $h_1, h_2 : \Sigma_2^* \to \Sigma_1^*$, a regular language $R \subseteq \Sigma_1^*$, and a projection $h_T : \Sigma_1^* \to T^*$ such that*

$$L = h_T(h_1(E(h_1, h_2)) \cap R). \tag{3}$$

Proof. Let L be an arbitrary recursively enumerable language generated by a phrase-structure grammar $G = (N, T, P, S)$, where N and T are the finite sets of nonterminals and terminals, respectively, P is the finite set of productions:

$$p_i : \alpha_i \to \beta_i, \quad i = 1, \ldots, n,$$

and $S \in N$ is the starting nonterminal. Without loss of generality, we assume that for each production $p_i : \alpha_i \to \beta_i$, $\beta_i \neq \lambda$, except for the production $S \to \lambda$ if $\lambda \in L$.

Define $T' = \{a' \mid a \in T\}$, $T'' = \{a'' \mid a \in T\}$, and $P' = \{p' \mid p \in P\}$. Denote by V and V_1 the sets $N \cup T$ and $N \cup T'$, respectively. For notational purpose, we also define a morphism $d : V^* \to V_1^*$ by $d(A) = A$ for $A \in N$ and $d(a) = a'$ for $a \in T$. Note that d is a bijection; thus, the inverse of d, d^{-1}, is well defined.

Let

$$\Sigma_1 = V \cup T' \cup \{B, F, \$\}, \tag{4}$$
$$\Sigma_2 = \Sigma_1 \cup T'' \cup P \cup P', \tag{5}$$

where B, F, and \$ are not in V, V_1, or V_2. The morphisms $h_1, h_2 : \Sigma_2^* \to \Sigma_1^*$, depending on G, are defined by the following:

(i)	$h_1(B) = BS\$$,	$h_2(B) = B$,	
(ii)	$h_1(\$) = \$$,	$h_2(\$) = \$$,	
(iii)	$h_1(p_i) = d(\beta_i)$,	$h_2(p_i) = d(\alpha_i)$,	for $p_i : \alpha_i \to \beta_i \in P$,
(iv)	$h_1(p_i') = \beta_i$,	$h_2(p_i') = d(\alpha_i)$,	for $p_i : \alpha_i \to \beta_i \in P$,
(v)	$h_1(A) = A$,	$h_2(A) = A$,	for $A \in N$,
(vi)	$h_1(a') = a'$,	$h_2(a') = a'$,	for $a' \in T'$,
(vii)	$h_1(a'') = a$,	$h_2(a'') = a'$,	for $a'' \in T''$,
$(viii)$	$h_1(a) = F$,	$h_2(a) = a$,	for $a \in T$,
(ix)	$h_1(\$') = F$,	$h_2(\$') = \$$,	
(x)	$h_1(F) = F$,	$h_2(F) = FF$.	

The regular language R is defined by the regular expression

$$BS(\$V_1^*)^*\$T^*F^+.$$

Note that $\alpha_i, \beta_i \neq \lambda$ for all i above. So, both h_1 and h_2 are λ-free morphisms. If $\lambda \in L$, then we introduce an additional symbol @ to Σ_2 and define

$$h_1(@) = h_2(@) = BS\$F.$$

It is easy to see that by defining $h_1(@)$ and $h_2(@)$, we will not introduce any other new words to $h_1(E(h_1, h_2)) \cap R$. Therefore, we assume that $\lambda \notin L$ in the following arguments.

We define $h_T : \Sigma_1^* \to T$ by

$$h_T(a) = \begin{cases} a, & \text{if } a \in T; \\ \lambda, & \text{if } a \notin T. \end{cases}$$

Now we show that the equation (3) holds.

Our proof for that $x \in L$ implies $x \in h_T(h_1(E(h_1,h_2)) \cap R)$ is similar to the one for Theorem 6.9 of [19]. So, we omit the formal proof. Instead, we use an example to explain our idea informally. The example is a modified version of the one from [19] (page 112).

Let L be generated by the following phrase-structure grammar G:

$$\begin{array}{lll} p_1 : S \to ACCC, & p_2 : CC \to CD, & p_3 : AC \to a, \\ p_4 : DC \to ACC, & p_5 : ACC \to C, & p_6 : C \to b. \end{array}$$

A derivation sequence for the word ab is

$$S \Longrightarrow ACCC \Longrightarrow ACDC \Longrightarrow aDC \Longrightarrow aACC \Longrightarrow aC \Longrightarrow ab. \tag{6}$$

According to this derivation sequence, we define

$$x = Bp_1\$Ap_2C\$p_3DC\$a'p_4\$a'p_5C\$a''p_6'\$'abFFF.$$

By the above definitions of h_1 and h_2, we have

$$h_1(x) = h_2(x) = BS\$ACCC\$ACDC\$a'DC\$a'ACC\$a'C\$abFFFFFF.$$

Then, clearly, $x \in E(h_1,h_2)$, $h_1(x) \in R$, and $h_T(h_1(x)) = ab$. Thus, $ab \in h_T(h_1(E(h_1,h_2)) \cap R)$.

Conversely, let $w \in h_T(h_1(E(h_1,h_2)) \cap R)$, i.e., $w = h_T(y)$ for some $y \in h_1(E(h_1,h_2)) \cap R$. Then by the definition of R, y is in the form

$$BS\$y_1\$y_2\$\ldots\$y_tF^l,$$

where $y_1,\ldots,y_{t-1} \in V_1^*$, $y_t \in T^*$, and $l > 0$. Let $y = h_1(x)$ for some $x \in E(h_1,h_2)$. Then

$$x = Bx_1\$x_2\$\ldots\$x_t\$'x_{t+1}F^m$$

such that $h_2(x_1) = S$, $h_1(x_i) = h_2(x_{i+1}) = y_i$, for $1 \le i \le t$, and $l = 2m$ and $h_1(x_{t+1}) = F^{m-1}$. Note that if $x_j = x_{j+1}$ for some j, $1 \le j < t$, then we can construct a new word x' by deleting $x_j\$$ from x so that $h_T(h_1(x') \cap R) = h_T(h_1(x) \cap R) = w$. So, without loss of generality, we assume that $x_j \ne x_{j+1}$ for all j, $1 \le j < t$. (It is clear that $x_t \ne x_{t+1}$). Then the following are clear:

(1) $x_1 = p$ (or $x_1 = p'$ if $t = 1$) for some $p : S \to \gamma$ in P,
(2) $x_i \in (V_1 \cup P)^*P(V_1 \cup P)^*$, for $2 \le i < t$,
(3) $x_t \in (V_2 \cup P')^*$,
(4) $x_{t+1} \in T^*$,
(5) $h_1(x_i) = y_i$ and $h_2(x_i) = y_{i-1}$, for $1 \le i \le t$ (letting $y_0 = S$).

By (2) and (5) above and (*iii*) of the definition of h_1 and h_2, it follows that

$$d^{-1}(y_{i-1}) \Rightarrow_G^+ d^{-1}(y_i),$$

$2 \le i \le t-1$. Note also that $S \Rightarrow_G d^{-1}(y_1)$ and $d^{-1}(y_{t-1}) \Rightarrow_G^+ y_t$. Therefore, we have $S \Rightarrow_G^+ y_t$, i.e., $y_t \in L$. Since $w = h_T(y) = y_t$, we have proved that $w \in L$. □

Proof of Lemma 7. Let $L \subseteq T^*$ be an arbitrary recursively enumerable language. By Lemma 8, $L = h_T(h_1(E(h_1,h_2)) \cap R)$ for some λ-free morphisms $h_1, h_2 : \Sigma_2^* \to \Sigma_1^*$, regular language $R \subseteq \Sigma_1^*$, and projection $h_T : \Sigma_1^* \to T^*$ defined by

$$h_T(X) = \begin{cases} X, & \text{if } X \in T, \\ \lambda, & \text{otherwise,} \end{cases}$$

Let $\Sigma_2 = \{b_0, b_1, \ldots, b_{n-1}\}$, for some integer $n > 0$, and R be accepted by a deterministic finite automaton $M = (Q, \Sigma_1, \delta, q_1, F)$, where $Q = \{q_0, q_1, \ldots, q_{m-1}\}$, for some $m > 0$. We construct the sticker system

$$\gamma = (V, \rho, A, B_d, B_u)$$

where

$$\begin{aligned}
V &= \Sigma_1 \cup Q \cup \{[q,j] \mid q \in Q, 0 \le j \le m-1\},\\
\rho &= \{(X,X) \mid X \in \Sigma_1\} \cup \{(q,q), ([q,j],q), (q,[q,k]), ([q,j],[q,k]) \mid q \in Q,\\
&\qquad 0 \le j,k \le m-1\},\\
A &= \left\{ \binom{q_0}{\#} \right\},\\
B_d &= \left\{ \binom{\#}{[q_{i_0},j]} \binom{\#}{a_1} \binom{\#}{q_{i_1}} \binom{\#}{q_{i_1}} \binom{\#}{a_2} \binom{\#}{q_{i_2}} \binom{\#}{q_{i_2}} \ldots \right.\\
&\qquad \binom{\#}{q_{i_{t_i-1}}} \binom{\#}{q_{i_{t_i-1}}} \binom{\#}{a_{t_i}} \binom{\#}{q_{i_{t_i}}} \Bigg|\\
&\qquad a_1 a_2 \ldots a_{t_i} = h_2(b_i), b_i \in \Sigma_2, \quad 0 \le j \le m-1,\\
&\qquad \left. \delta(q_{i_k}, a_{k+1}) = q_{i_{k+1}}, 0 \le k < t_i \right\} \bigcup\\
&\qquad \left\{ \binom{\#}{q_i} \binom{\#}{q_i} \Bigg| q_i \in F \right\},\\
B_u &= \left\{ \binom{a_1}{\#} \binom{[q_{i_1},j]}{\#} \binom{q_{i_1}}{\#} \binom{a_2}{\#} \binom{q_{i_2}}{\#} \binom{q_{i_2}}{\#} \ldots \right.\\
&\qquad \binom{a_{t_i}}{\#} \binom{q_{i_{t_i}}}{\#} \binom{q_{i_{t_i}}}{\#} \Bigg|\\
&\qquad a_1 a_2 \ldots a_{t_i} = h_1(b_i), b_i \in \Sigma_2, \quad 0 \le j \le m-1,\\
&\qquad \left. \delta(q_{i_k}, a_{k+1}) = q_{i_{k+1}}, 1 \le k < t_i \right\} \bigcup\\
&\qquad \left\{ \binom{q_i}{\#} \Bigg| q_i \in F \right\}.
\end{aligned}$$

Note that each string of B_d or B_u contains an integer j, $0 \le j < m$, which is paired with the state that appears first (from the left) in the string. The function of this integer will become clear later.

Denote by $r_d(i,j,k)$, $0 \le i < n$ and $0 \le j,k < m$, a string in B_d which is constructed with the word $h_2(b_i)$ and the state q_j as the first state that is paired with the integer k. Similarly, denote by $r_u(i,j,k)$ a string in B_u.

Assume that F contains $l > 0$ states and

$$F = \{f_0, f_1, \ldots, f_{l-1}\},\ l \le m.$$

Then we denote by $r_d(n,0,j)$ the string $\binom{\#}{f_j}\binom{\#}{f_j}$ and by $r_u(n,j,0)$ the string $\binom{f_j}{\#}$.

It is clear that $card(B_d) = card(B_u) = nm^2 + l$. Define the labelling mappings $e_d : B_d \to \{1, \ldots, card(B_d)\}$ by

$$e_d(r_d(i,j,k)) = i \cdot m^2 + j \cdot m + k + 1$$

and $e_u : B_u \to \{1, \ldots, card(B_u)\}$ by

$$e_u(r_u(i,j,k)) = i \cdot m^2 + k \cdot m + j + 1.$$

Let u denote a string in A. By the above construction of γ, it is clear that $u \Longrightarrow_\gamma^* z$ if and only if there is a sequence

$$q_{i_0} a_1 q_{i_1} a_2 \ldots q_{i_{t-1}} a_t q_{i_t}$$

such that (1) $q_{i_0} = q_0$, $q_{i_t} \in F$, and $\delta(q_{i_{k-1}}, a_k) = q_{i_k}$; and (2) there is $x \in \Sigma_2^*$ such that $h_1(x) = h_2(x) = a_1 a_2 \ldots a_t$.

Consider also the weak coding $g : \binom{V}{V}_\rho^* \longrightarrow T^*$ defined by

$$g(\binom{\alpha}{\beta}) = \begin{cases} a, & \text{if } \alpha = \beta = a, a \in T, \\ \lambda, & \text{otherwise.} \end{cases}$$

We show that $g(L_c(\gamma)) = h_T(h_1(E(h_1,h_2)) \cap R)$ in the following.

Let $w \in h_T(h_1(E(h_1,h_2)) \cap R)$. Then there exist $x = b_{i_1} b_{i_2} \ldots b_{i_s} \in E(h_1,h_2)$ and $y = h_1(x) = h_2(x)$ such that $y \in R$ and $w = h_T(y)$. Let $y = a_1 a_2 \ldots a_t$. Then we have a state sequence $q_{j_1}, q_{j_2}, \ldots, q_{j_{t+1}}$ of M such that $q_{j_1} = q_0$, $q_{j_{t+1}} = f_r \in F$ for some r, $0 \le r < l$, and $\delta(q_{j_k}, a_k) = q_{j_{k+1}}$ for $1 \le k \le t$. Note that $h_1(x) = h_1(b_{i_1}) \ldots h_1(b_{i_s}) = a_1 \ldots a_t$. Let $h_1(b_{i_k}) = a_{\alpha_k} \ldots a_{\alpha_{k+1}-1}$, $1 \le k < s$, and $h_1(b_{i_s}) = a_{\alpha_s} \ldots a_t$. Similarly, let $h_2(b_{i_k}) = a_{\beta_k} \ldots a_{\beta_{k+1}-1}$, $1 \le k < s$, and $h_2(b_{i_s}) = a_{\beta_s} \ldots a_t$. Then, there exists a computation D such that D uses the following strings from B_d:

$$r_d(i_1, j_{\beta_1}, j_{\alpha_1+1}), \ldots, r_d(i_s, j_{\beta_s}, j_{\alpha_s+1}), r_d(n,0,r)$$

and the following from B_u:

$$r_u(i_1, j_{\alpha_1+1}, j_{\beta_1}), \ldots, r_u(i_s, j_{\alpha_s+1}, j_{\beta_s}), r_u(n,r,0).$$

Let the result of the computation D be z. Clearly, by the definition of g, $g(z) = w$. It is also easy to see that $e_d(D) = e_u(D)$ by the definitions of e_d and e_u.

We now show that if $w \in g(L_c(\gamma))$, then $w \in h_T(h_1(E(h_1, h_2)) \cap R)$. We have $w \in g(z)$ for some z that is the result of a computation D of γ and $e_u(D) = e_d(D)$. By the construction of the sticker system γ, one can observe that z corresponds to a sequence

$$q_{i_0}, a_1, q_{i_1}, a_2, \ldots, q_{i_{t-1}}, a_t, q_{i_t}$$

where $q_{i_0} = q_0$, $q_{i_t} \in F$, and $\delta(q_{i_{k-1}}, a_k) = q_{i_k}$, for $1 \leq k \leq t$. Then it is clear that $a_1 a_2 \ldots a_t \in R$. By the definition of B_d and B_u and the fact that $e_u(D) = e_d(D)$, it follows that $a_1 a_2 \ldots a_t = h_1(x) = h_2(x)$ for some $x \in \Sigma_2$. Denote $a_1 a_2 \ldots a_t$ by y. Then, $y \in h_1(E(h_1, h_2)) \cap R$. It is easy to show that $w = g(z) = h_T(y)$. Therefore, $w \in h_T(h_1(E(h_1, h_2)) \cap R)$. $\square$

From Lemmas 6 and 7 we get

Theorem 2. *$RE = wcode(CSL)$.*

6. An intermediate case

For the fair computations we have

Theorem 3. *$REG \subset wcode(FSL) \subset RE$.*

Proof. The inclusion $REG \subseteq wcode(FSL)$ follows from the proof of Lemma 5.

For the strictness, let us consider the sticker system

$$\begin{aligned}
\gamma \quad = \quad & (V, \rho, A, B_d, B_u), \\
& V = \{a, a', b, b'\}, \\
& \rho = \{(a, a'), (b, b')\}, \\
& A = \left\{ \binom{a}{\#} \right\}, \\
& B_d = \left\{ \binom{\#}{a'},\ \binom{\#}{b'}\binom{\#}{b'} \right\}, \\
& B_u = \left\{ \binom{a}{\#}\binom{a}{\#},\ \binom{b}{\#} \right\}.
\end{aligned}$$

Starting with the unique axiom in A, we have to use $\binom{\#}{a'}$ from B_d and we obtain a blunt sequence. We can continue with any string in B_d and B_u. However, due to the complementarity restrictions, if a symbol b or b' is introduced, then we have to continue by using composite symbols $\binom{\#}{b'}$ and $\binom{b}{\#}$ until obtaining again a blunt sequence.

Thus, let us intersect the language $L_f(\gamma)$ with the regular language $\binom{a}{a'}^+ \binom{b}{b'}^+$. We obtain a language consisting of strings of the form $\binom{a}{a'}^{2n+1} \binom{b}{b'}^{2m}$, $n, m \geq 1$, produced by computations where

– the first string in B_d is used $2n+1$ times,
– the second string in B_d is used m times,
– the first string in B_u is used n times (one occurrence of a is introduced by the axiom),
– the second string in B_u is used $2m$ times.

Due to the fairness, we must have

$$2n + 1 + m = n + 2m,$$

which means that

$$n = m - 1.$$

The language $\{\binom{a}{a'}^{2m-1}\binom{b}{b'}^{2m} \mid m \geq 1\}$ is not a regular one, hence $L_f(\gamma)$ is not regular.

From the Turing-Church thesis we have $wcode(FSL) \subseteq RE$. For the strictness, we shall prove that $wcode(FSL) \subseteq MAT^{\lambda}$, where MAT^{λ} is the family of languages generated by context-free matrix grammars with arbitrary rules. Because $MAT^{\lambda} \subset RE$ ([6], [8]), we obtain $wcode(FSL) \subset RE$.

Consider a sticker system $\gamma = (V, \rho, A, B_d, B_u)$. Define

$$\begin{aligned}
V' &= \{a' \mid a \in V\},\\
L(A) &= \{[a_1, b_1'] \dots [a_k, b_k'] a_{k+1} \dots a_{k+r} \mid k, r \geq 0, k + r \geq 1,\\
&\quad \binom{a_1}{b_1} \dots \binom{a_k}{b_k}\binom{a_{k+1}}{\#} \dots \binom{a_{k+r}}{\#} \in A\}\\
&\cup \{[a_1, b_1'] \dots [a_k, b_k'] b_{k+1}' \dots b_{k+r}' \mid k, r \geq 0, k + r \geq 1,\\
&\quad \binom{a_1}{b_1} \dots \binom{a_k}{b_k}\binom{\#}{b_{k+1}} \dots \binom{\#}{b_{k+r}} \in A\}\\
&\cup \{\lambda \mid \lambda \in A\},\\
L(B_d) &= \{b_1' \dots b_k' \mid k \geq 1, \binom{\#}{b_1} \dots \binom{\#}{b_k} \in B_d\},\\
L(B_u) &= \{a_1 \dots a_k \mid k \geq 1, \binom{a_1}{\#} \dots \binom{a_k}{\#} \in B_u\}.
\end{aligned}$$

Consider the new symbols s, d, d' and construct the languages

$$\begin{aligned}
&L_1 = \{xd' \mid x \in L(B_d)\}^+,\\
&L_2 = \{xd \mid x \in L(B_u)\}^+,\\
&L_1' = L_1 \sqcup\!\sqcup\, c^+,\\
&L_2' = L_2 \sqcup\!\sqcup\, c^+,\\
&L_3 = (L(A)L_1' \sqcup\!\sqcup\, L_2') \cap \{[a, b'] \mid a, b \in V\}^*(VV' \cup \{cd', dc\})^*.
\end{aligned}$$

($\sqcup\!\sqcup$ is the shuffle operation: $x \sqcup\!\sqcup\, y = \{x_1y_1 \dots x_ny_n \mid n \geq 1, x = x_1 \dots x_n, y = y_1 \dots y_n, x_i, y_i \in V^*, 1 \leq i \leq n\}$.)

Clearly, L_1, L_2 are regular languages, hence also L_3 is regular: the family *REG* is closed under the shuffle operation and under intersection.

Consider the gsm Q which:

- leaves unchanged the symbols $[a, b'], a, b \in V$,
- replaces each pair ab' by $[a, b'], a, b \in V$,
- replaces each pair cd' by $[c, d']$ and each pair dc by $[d, c]$.

The language $Q(L_3)$ is also regular, over the alphabet

$$U = \{[a, b'] \mid a, b \in V\} \cup \{[c, d'], [d, c]\}.$$

Let $G = (N, U, S, P)$ be a regular grammar for $Q(L_3)$ and construct the matrix grammar

$$G' = (N', \binom{V}{V}_\rho, S', M),$$

where

$$\begin{aligned} N' &= N \cup U \cup \{S'\}, \\ M &= \{(S' \to S)\} \cup \{(r) \mid r \in P\} \\ &\cup \{([a, b'] \to \binom{a}{b}) \mid a, b \in V\} \\ &\cup \{([c, d'] \to \lambda, [d, c] \to \lambda)\}. \end{aligned}$$

It is easy to see that $L(G')$ contains all the strings $w \in \binom{V}{V}_\rho^*$ such that $x \Longrightarrow^* w$ in γ, $x \in A$, and this is a fair derivation: the matrix $([c, d'] \to \lambda, [d, c] \to \lambda)$ checks whether or not the number of symbols d and d' is the same.

The family MAT^λ is closed under arbitrary morphisms ([5]), hence the image by a weak coding of $L_f(\gamma)$ is in the family MAT^λ. □

Open problem. Is the family *FSL* included in the family of context-free languages (or even in the family of linear languages) ?

7. Final remarks

We have found characterizations of families *REG* and *RE* using variants of sticker systems. Several further research directions are of interest:

- Define variants of sticker systems which characterize families of languages different from *REG* and *RE*.
- Look for universal sticker systems, in the sense of universal Turing machines. This is important to the construction of universal and *programmable* DNA sticker systems.
- Look for other variants which characterize *RE*, but not using the coherence restriction. This is again important to finding “realistic” models of DNA computers based on the sticker operation.

- Investigate the descriptional complexity (the size) of sticker systems and languages. Several parameters are very natural: the number of axioms, the length of the longest axiom, the number of elements in the sets B_d, B_u, the length of the longest string in B_d, B_u, etc. Do these measures give infinite hierarchies of sticker languages?

References

1. L. M. Adleman: Molecular computation of solutions to combinatorial problems, Science, 226 (Nov. 1994), 1021 – 1024.
2. L. M. Adleman: On constructing a molecular computer. In: R.J. Lipton, E.B. Baum (eds.) DNA Based Computers, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 27, American Math. Soc., 1996, 1 – 22.
3. E. Csuhaj-Varju, L. Freund, L. Kari, Gh. Păun: DNA computing based on splicing: universality results, First Annual Pacific Symp. on Biocomputing, Hawaii, Jan. 1996.
4. K. Culik II: A purely homomorphic characterization of recursively enumerable sets, Journal of the ACM 26 (1979) 345-350.
5. J. Dassow, Gh. Păun: Regulated Rewriting in Formal Language Theory. Berlin Heidelberg New York: Springer, 1989.
6. J. Dassow, Gh. Păun, A. Salomaa: Grammars with controlled derivations. In: G. Rozenberg, A. Salomaa (eds.) Handbook of Formal Languages. Berlin Heidelberg New York: Springer, 1997.
7. R. Freund, L. Kari, Gh. Păun: DNA computing based on splicing: The existence of universal computers, Technical Report 185-2/FR-2/95, TU Wien, 1995.
8. D. Hauschild, M. Jantzen: Petri nets algorithms in the theory of matrix grammars, Acta Informatica, 31 (1994), 719 – 728.
9. T. Head: Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviors, Bull. Math. Biology, 49 (1987), 737 – 759.
10. T. Head, Gh. Păun, D. Pixton: Language theory and molecular genetics. Generative mechanisms suggested by DNA recombination. In: G. Rozenberg, A. Salomaa (eds.) Handbook of Formal Languages. Berlin Heidelberg New York: Springer, 1997.
11. R. J. Lipton: Using DNA to solve NP-complete problems, Science, 268 (Apr. 1995), 542 – 545.
12. R. J. Lipton: Speeding up computations via molecular biology. In: R.J. Lipton, E.B. Baum (eds.) DNA Based Computers, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 27, American Math. Soc., 1996, 67 – 74.
13. Gh. Păun: Splicing. A challenge to formal language theorists, Bulletin EATCS, 57 (1995), 183 – 194.
14. Gh. Păun: Regular extended H systems are computationally universal, J. Automata, Languages and Combinatorics, 1, 1 (1996), 27 – 36.
15. Gh. Păun, G. Rozenberg, A. Salomaa: Computing by splicing, Theor. Computer Sci., 168 (1996), 321 – 336.
16. Gh. Păun, A. Salomaa: DNA computing based on the splicing operation, Mathematica Japonica, 43, 3 (1996), 607 – 632.
17. A. Salomaa: Formal Languages. New York, London: Academic Press, 1973.
18. A. Salomaa: Equality sets for homomorphisms of free monoids, Acta Cybernetica 4 (1978) 127-139.
19. A. Salomaa: Jewels of Formal Language Theory. Rockville, MD: Computer Science Press, 1981.