Matrix Insertion-Deletion Systems

Ion Petre[1]   Sergey Verlan[2]

[1]Department of IT, Åbo Akademi University, Turku 20520 Finland
ipetre@abo.fi
[2]Laboratoire d'Algorithmique, Complexité et Logique,
Département Informatique,
Université Paris Est,
61, av. Général de Gaulle, 94010 Créteil, France
verlan@univ-paris12.fr

Abstract

In this article, we consider for the first time the operations of insertion and deletion working in a matrix controlled manner. We show that, similarly as in the case of context-free productions, the computational power is strictly increased when using a matrix control: computational completeness can be obtained by systems with insertion or deletion rules involving at most two symbols in a contextual or in a context-free manner and using only binary matrices.

1   Introduction

The operations of insertion and deletion were first considered with a linguistic motivation [15, 5, 18]. Another inspiration for these operations comes from the fact that the insertion operation and its iterated variants are generalized versions of Kleene's operations of concatenation and closure [11], while the deletion operation generalizes the quotient operation. A study of properties of the corresponding operations may be found in [7, 8, 9]. Insertion and deletion also have interesting biological motivations, e.g., they correspond to a mismatched annealing of DNA sequences; these operations are also present in the evolution processes in the form of point mutations as well as in RNA editing, see the discussions in [1, 2, 22] and [20]. These biological motivations of insertion-deletion operations led to their study in the framework of molecular computing, see, for example, [3, 10, 20, 23].

In general, an insertion operation means adding a substring to a given string in a specified (left and right) context, while a deletion operation means removing a substring of a given string from a specified (left and right) context. A finite set of insertion-deletion rules, together with a set of axioms provide a language

1

generating device: starting from the set of initial strings and iterating insertion-deletion operations as defined by the given rules, one obtains a language.

Even in their basic variants, insertion-deletion systems are able to characterize the recursively enumerable languages. Moreover, as it was shown in [16], the context dependency may be replaced by insertion and deletion of strings of sufficient length, in a context-free manner. If the length is not sufficient (less or equal to two) then such systems are not able to generate more than the recursive languages and a characterization of them was shown in [24].

Similar investigations were continued in [17, 12, 13] on insertion-deletion systems with one-sided contexts, i.e., where the context dependency is present only from the left or only from the right side of all insertion and deletion rules. The papers cited above give several computational completeness results depending on the size of insertion and deletion rules. We recall the interesting fact that some combinations are not leading to computational completeness, i.e., there are recursively enumerable languages that cannot be generated by such devices.

Like in the case of context-free rewriting, it is possible to consider a graph-controlled variant of insertion-deletion systems. Thus the rules cannot be applied at any time, as their applicability depends on the current "state", changed by a rule application. Such a formalization is rather similar to the definition of insertion-deletion P systems [19], however it is even simpler and more natural. The article [4] focuses on one-sided graph-controlled insertion-deletion systems where at most two symbols may be present in the description of insertion and deletion rules. This correspond to systems of size $(1,1,0;1,1,0)$, $(1,1,0;1,0,1)$, $(1,1,0;2,0,0)$, and $(2,0,0;1,1,0)$, where the first three numbers represent the maximal size of the inserted string and the maximal size of the left and right contexts, while the last three numbers represent the same information, but for deletion rules. It is known that such systems are not computationally complete [14], while the corresponding P systems variants and graph-controlled variants are computationally complete.

In this article we introduce a new type of control, similar to the one used in matrix grammars. More precisely, insertion and deletion rules are grouped in sequences, called matrices, and either the whole sequence is applied consecutively, or no rule is applied. We show that in the case of such control the computational power of systems of size $(1,1,0;2,0,0)$ and $(2,0,0;1,1,0$ is strictly increasing. Moreover, we show that binary matrices suffice to achieve this result, hence we obtain a similar characterization like in the case of the binary normal form for matrix grammars.

2 Definitions

We do not present the usual definitions concerning standard concepts of the theory of formal languages and we only refer to textbooks such as [21] for more details.

The empty string is denoted by λ.

In the following, we will use special variants of the *Geffert* normal form for type-0 grammars (see [6] for more details).

A grammar $G = (N, T, P, S)$ is said to be in *Geffert normal form* [6] if $N = \{S, A, B, C, D\}$ and P only contains context-free rules of the forms $S \to uSv$ with $u \in \{A, C\}^*$ and $v \in \{B, D\}^*$ as well as $S \to x$ with $x \in (T \cup \{A, B, C, D\})^*$ and two (non-context-free) erasing rules $AB \to \lambda$ and $CD \to \lambda$.

We remark that we can easily transform the linear rules from the Geffert normal form into a set of left-linear and right-linear rules (by increasing the number of non-terminal symbols, e.g., see [19]). More precisely, we say that a grammar $G = (N, T, P, S)$ with $N = N' \cup N''$, $S, S' \in N'$, and $N'' = \{A, B, C, D\}$, is in the *special Geffert normal form* if, besides the two erasing rules $AB \to \lambda$ and $CD \to \lambda$, it only has context-free rules of the following forms:

$$
\begin{aligned}
X &\to bY, \quad X, Y \in N', b \in T \cup N'', \\
X &\to Yb, \quad X, Y \in N', b \in T \cup N'', \\
S' &\to \lambda.
\end{aligned}
$$

Moreover, we may even assume that, except for the rules of the forms $X \to Sb$ and $X \to S'b$, for the first two types of rules it holds that the right-hand side is unique, i.e., for any two rules $X \to w$ and $U \to w$ in P we have $U = X$.

The computation in a grammar in the special Geffert normal form is done in two stages. During the first stage, only context-free rules are applied. During the second stage, only the erasing rules $AB \to \lambda$ and $CD \to \lambda$ are applied. These two erasing rules are not applicable during the first stage as long as the left and the right part of the current string are still separated by S (or S') as all the symbols A and C are generated on the left side of these middle symbols and the corresponding symbols B and D are generated on the right side. The transition between stages is done by the rule $S' \to \lambda$. We remark that all these features of a grammar in the special Geffert normal form are immediate consequences of the proofs given in [6].

2.1 Insertion-deletion systems

An *insertion-deletion system* is a construct $ID = (V, T, A, I, D)$, where V is an alphabet; $T \subseteq V$ is the set of *terminal* symbols (in contrast, those of $V - T$ are called *non-terminal* symbols); A is a finite language over V, the strings in A are the *axioms*; I, D are finite sets of triples of the form (u, α, v), where u, α ($\alpha \neq \lambda$), and v are strings over V. The triples in I are *insertion rules*, and those in D are *deletion rules*. An insertion rule $(u, \alpha, v) \in I$ indicates that the string α can be inserted between u and v, while a deletion rule $(u, \alpha, v) \in D$ indicates that α can be removed from between the context u and v. Stated in another way, $(u, \alpha, v) \in I$ corresponds to the rewriting rule $uv \to u\alpha v$, and $(u, \alpha, v) \in D$ corresponds to the rewriting rule $u\alpha v \to uv$. By $\Rightarrow_{ins}$ we denote the relation defined by the insertion rules (formally, $x \Rightarrow_{ins} y$ if and only if $x = x_1 uv x_2, y = x_1 u\alpha v x_2$, for some $(u, \alpha, v) \in I$ and $x_1, x_2 \in V^*$), and by

$\Rightarrow_{del}$ the relation defined by the deletion rules (formally, $x \Rightarrow_{del} y$ if and only if $x = x_1 u\alpha v x_2, y = x_1 uv x_2$, for some $(u, \alpha, v) \in D$ and $x_1, x_2 \in V^*$). By $\Rightarrow$ we refer to any of the relations $\Rightarrow_{ins}, \Rightarrow_{del}$, and by $\Rightarrow^*$ we denote the reflexive and transitive closure of $\Rightarrow$.

The language generated by ID is defined by

$$L(ID) = \{w \in T^* \mid x \Rightarrow^* w \text{ for some } x \in A\}.$$

The complexity of an insertion-deletion system $ID = (V, T, A, I, D)$ is described by the vector $(n, m, m'; p, q, q')$ called *size*, where

$$n = \max\{|\alpha| \mid (u, \alpha, v) \in I\}, \qquad p = \max\{|\alpha| \mid (u, \alpha, v) \in D\},$$
$$m = \max\{|u| \mid (u, \alpha, v) \in I\}, \qquad q = \max\{|u| \mid (u, \alpha, v) \in D\},$$
$$m' = \max\{|v| \mid (u, \alpha, v) \in I\}, \qquad q' = \max\{|v| \mid (u, \alpha, v) \in D\}.$$

By $INS_n^{m,m'} DEL_p^{q,q'}$ we denote the families of insertion-deletion systems having the size $(n, m, m'; p, q, q')$.

If one of the parameters n, m, m', p, q, q' is not specified, then instead we write the symbol $*$. In particular, $INS_*^{0,0} DEL_*^{0,0}$ denotes the family of languages generated by *context-free insertion-deletion systems*. If one of numbers from the pairs m, m' and/or q, q' is equal to zero (while the other one is not), then we say that the corresponding families have a one-sided context. Finally we remark that the rules from I and D can be put together into one set of rules R by writing $(u, \alpha, v)_{ins}$ for $(u, \alpha, v) \in I$ and $(u, \alpha, v)_{del}$ for $(u, \alpha, v) \in D$.

2.2 Matrix insertion-deletion systems

Like context-free grammars, insertion-deletion systems may be extended by adding some additional controls. We discuss here the adaptation of the idea of matrix grammars for insertion-deletion systems.

A *matrix insertion-deletion system* is a construct

$$\gamma = (V, T, A, M) \text{ where}$$

- V is a finite alphabet,

- $T \subseteq V$ is the *terminal alphabet*,

- $A \subseteq V^*$ is a finite set of *axioms*,

- $M = r_1, \ldots, r_n$ is a finite set of sequences of rules, called *matrices*, of the form $r_i : [r_{i1}, \ldots r_{ik}]$ where r_{ij}, $1 \leq i \leq n$, $1 \leq j \leq k$ is an insertion or deletion rule over V.

The sentential form (also called configuration) of γ is a string $w \in V^*$. A transition $w \Longrightarrow_{r_i} w'$, for $1 \leq i \leq n$, is performed if there exist $w_1, \ldots, w_k \in V^*$ such that $w \Rightarrow_{r_{i1}} w_1 \Rightarrow_{r_{i2}} \ldots \Rightarrow_{r_{ik}} w_k$ and $w_k = w'$.

4

The language generated by γ is defined by

$$L(\gamma) = \{w \in T^* \mid x \Longrightarrow^* w \text{ for some } x \in A\}.$$

By $Mat_k INS_n^{m,m'} DEL_p^{q,q'}$, $k > 1$, we denote the families of matrix insertion-deletion systems having matrices with at most k rules and insertion and deletion rules of size $(n, m, m'; p, q, q')$.

3   Computational completeness

For all the variants of insertion and deletion rules considered in this section, we know that the basic variants without using matrix control cannot achieve computational completeness (see [14], [17]). The computational completeness results from this section are based on simulations of derivations of a grammar in the special Geffert normal form. These simulations associate a group of insertion and deletion rules to each of the right- or left-linear rules $X \to bY$ and $X \to Yb$. The same holds for (non-context-free) erasing rules $AB \to \lambda$ and $CD \to \lambda$. We remark that during the derivation of a grammar in the special Geffert normal form, any sentential form contains at most one non-terminal symbol from N'.

We start with the following affirmation: if the size of the matrices is sufficiently large, then corresponding systems are computationally complete. This is quite obvious for matrices of size 3.

Theorem 1. $Mat_3 INS_1^{1,0} DEL_2^{0,0} = RE$.

Proof. The proof is based on a simulation of a type-0 grammar in the Geffert normal form (as presented in Section 2). Let $G = (V, T, S, P)$ be such a grammar. We construct the system $\gamma = (V, T, \{S\}, M)$ as follows.

For every rule $r : A \to xy \in P$, $x, y \in V$ we add to M the matrix
$r : [(A, y, \lambda)_{ins}, (A, x, \lambda)_{ins}, (\lambda, A, \lambda)_{del}]$.

For rules $AB \to \lambda \in P$, $CD \to \lambda \in P$ and $S' \to \lambda$ we add to M following matrices:
$AB : [(\lambda, AB, \lambda)_{del}, (\lambda, \lambda, \lambda)_{ins}, (\lambda, \lambda, \lambda)_{ins}]$,
$CD : [(\lambda, CD, \lambda)_{del}, (\lambda, \lambda, \lambda)_{ins}, (\lambda, \lambda, \lambda)_{ins}]$ and
$S' : [(\lambda, S', \lambda)_{del}, (\lambda, \lambda, \lambda)_{ins}, (\lambda, \lambda, \lambda)_{ins}]$.

It is clear that $L(\gamma) = L(G)$. Indeed, rules of type $A \to xy$ are simulated by consecutively inserting y and x after A and finally deleting A. The rules $AB \to \lambda$ and $CD \to \lambda$ are simulated by directly erasing 2 symbols and the rule $S' \to \lambda$ by directly erasing S'. $\qquad\square$

A similar result can be obtained in the case of systems having rules of size $(1, 1, 0; 1, 1, 0)$.

Theorem 2. $Mat_3 INS_1^{1,0} DEL_1^{1,0} = RE$.

Proof. The proof is done like in the previous theorem. Right- and left-linear rules are simulated exactly in the same manner. The rule $AB \to \lambda$ can be

simulated by three matrices (providing that the axiom is $\{\$S\}$):

$AB.1 : [(\lambda, K_{AB}, \lambda)_{ins}, (\lambda, \$, \lambda)_{del}, (\lambda, \lambda, \lambda)_{ins}]$,

$AB.2 : [(K_{AB}, A, \lambda)_{del}, (K_{AB}, B, \lambda)_{del}, (\lambda, \lambda, \lambda)_{ins}]$ and

$AB.3 : [(\lambda, K_{AB}, \lambda)_{del}, (\lambda, \$, \lambda)_{ins}, (\lambda, \lambda, \lambda)_{ins}]$.

They simulate $AB \to \lambda$ by introducing in the string a symbol K_{AB} in a context-free manner and after that by deleting one copy of adjacent A and B. The validity follows from the observation that there can be at most only one copy of K_{AB} in the string (because it's insertion and deletion is synchronized with the deletion and insertion of a special symbol \$ initially present in only one copy). In order to delete this symbol at the end of the computation the matrix $[(\lambda, \$, \lambda)_{del}, (\lambda, \lambda, \lambda)_{ins}, (\lambda, \lambda, \lambda)_{ins}]$ shall be used.

The rule $CD \to \lambda$ is simulated similarly and the rule $S' \to \lambda$ by directly erasing S'. $\qquad\square$

By taking deletion rules with a right context in the previous theorem we obtain.

Theorem 3. $Mat_3INS_1^{1,0}DEL_1^{0,1} = RE$.

We give below the proof for the case of systems of size $(2, 0, 0; 1, 1, 0)$.

Theorem 4. $Mat_3INS_2^{0,0}DEL_1^{1,0} = RE$.

Proof. The proof is based on a simulation of a type-0 grammar in the Geffert normal form (as presented in Section 2). Let $G = (V, T, S, P)$ be such a grammar. We construct the system $\gamma = (V \cup V', T, \{S\}, M)$ as follows ($V' = \{X_A, Y_A \mid A \in V\} \cup \{K_{AB}, K_{CD}\}$).

For every rule $r : A \to bC \in P$ we add to M the matrix

$r : [(\lambda, bC, \lambda)_{ins}, (C, A, \lambda)_{del}]$.

For every rule $r : A \to Cb \in P$ we add to M the matrices

$r.1 : [(\lambda, X_A Y_A, \lambda)_{ins}, (Y_A, A, \lambda)_{del}]$,

$r.2 : [(\lambda, Cb, \lambda)_{ins}, (b, Y_A, \lambda)_{del}, (\lambda, \lambda, \lambda)_{del}]$ and

$r.3 : [(\lambda, \$, \lambda)_{del}, (\lambda, X_A, \lambda)_{del}, (\lambda, \$, \lambda)_{ins}]$.

For rules $AB \to \lambda \in P$ and $CD \to \lambda \in P$ we add to M following six matrices:

$AB.1 : [(\lambda, K_{AB}, \lambda)_{ins}, (\lambda, \$, \lambda)_{del}, (\lambda, \lambda, \lambda)_{ins}]$, $\qquad CD.1 : [(\lambda, K_{CD}, \lambda)_{ins}, (\lambda, \$, \lambda)_{del}, (\lambda, \lambda, \lambda)_{ins}]$,

$AB.2 : [(K_{AB}, A, \lambda)_{del}, (K_{AB}, B, \lambda)_{del}, (\lambda, \lambda, \lambda)_{ins}]$, $\quad CD.2 : [(K_{CD}, C, \lambda)_{del}, (K_{CD}, D, \lambda)_{del}, (\lambda, \lambda, \lambda)_{ins}]$,

$AB.3 : [(\lambda, K_{AB}, \lambda)_{del}, (\lambda, \$, \lambda)_{ins}, (\lambda, \lambda, \lambda)_{ins}]$, $\qquad CD.3 : [(\lambda, K_{CD}, \lambda)_{del}, (\lambda, \$, \lambda)_{ins}, (\lambda, \lambda, \lambda)_{ins}]$.

The rule $S' \to \lambda$ is simulated by the matrix that introduces symbol \$ S' : $[(\lambda, S', \lambda)_{del}, (\lambda, \$, \lambda)_{ins}, (\lambda, \lambda, \lambda)_{ins}]$.

It is clear that $L(\gamma) = L(G)$. Indeed, any rule $A \to bC$ is simulated directly by inserting bC and deleting A in the context of C. The right position for the insertion is insured by the uniqueness of A. Rules $r : A \to Cb$ are simulated in a different way. First A is replaced by $X_A Y_A$ and after that Y_A is rewritten

6

by Cb as in the previous case. We remark that by inserting $X_A Y_A$ we insure that there is no symbol b before Y_A. This permits to correctly place Cb. The additional symbol X_A remaining in the string is deleted during the second stage (when symbol $\$$ is introduced). As before, in order to delete $\$$ at the end of the computation the matrix $[(\lambda, \$, \lambda)_{del}, (\lambda, \lambda, \lambda)_{ins}, (\lambda, \lambda, \lambda)_{ins}]$ shall be used. $\square$

Since the matrix control is a particular case of the graph control we obtain

Theorem 5. *[14] For any $k > 0$, $REG \setminus Mat_k INS_2^{0,0} DEL_2^{2,0} \neq \emptyset$.*

4 Computational completeness for binary matrices

In this section we show that binary matrices suffice for computational completeness.

Theorem 6. $Mat_2 INS_2^{0,0} DEL_1^{1,0} = RE$.

Proof. The proof is based on a simulation if type-0 grammar in the Geffert normal form (as presented in Section 2). Let $G = (V, T, S, P)$ be such a grammar. We construct the system $\gamma = (V \cup V', T, w, M)$ as follows.

$V' = \{\#_k^r, K^r \mid r \in P, 1 \leq k \leq 5\} \cup \{K_{AB}, K_{CD}, \$\}$, and $w = \{\$S\}$.

For every rule $r : A \to bC \in P$ we add to M the matrix

$$r.1 : [(\lambda, bC, \lambda)_{ins}, \ (C, A, \lambda)_{del}].$$

For every rule $r : A \to Cb \in P$ we add to M following matrices:

$$
\begin{aligned}
&r.1 : [(\lambda, \#_1^r \#_2^r, \lambda)_{ins}, \ (\#_2^r, A, \lambda)_{del}] \\
&r.2 : [(\lambda, C, \lambda)_{ins}, \ (C, \#_1^r, \lambda)_{del}] \\
&r.3 : [(\lambda, \#_3^r \#_4^r, \lambda)_{ins}, \ (\#_4^r, \#_2^r, \lambda)_{del}] \\
&r.4 : [(\lambda, \#_5^r b, \lambda)_{ins}, \ (b, \#_4^r, \lambda)_{del}] \\
&r.5 : [(\lambda, \$, \lambda)_{del}, \ (\lambda, K^r, \lambda)_{ins}] \\
&r.6 : [(\lambda, K^r, \lambda)_{del}, \ (\lambda, \$, \lambda)_{ins}] \\
&r.7 : [(K^r, \#_3^r, \lambda)_{del}, \ (K^r, \#_5^r, \lambda)_{del}]
\end{aligned}
$$

For rules $AB \to \lambda \in P$ and $CD \to \lambda \in P$ we add to M following matrices:

$AB.1 : [(\lambda, \$, \lambda)_{del}, \ (\lambda, K_{AB}, \lambda)_{ins}]$ $AB.1 : [(\lambda, \$, \lambda)_{del}, \ (\lambda, K_{CD}, \lambda)_{ins}]$

$AB.2 : [(\lambda, K_{AB}, \lambda)_{del}, \ (\lambda, \$, \lambda)_{ins}]$ $AB.2 : [(\lambda, K_{CD}, \lambda)_{del}, \ (\lambda, \$, \lambda)_{ins}]$

$AB.3 : [(K_{AB}, A, \lambda)_{del}, \ (K_{AB}, B, \lambda)_{del}]$ $AB.3 : [(K_{CD}, C, \lambda)_{del}, \ (K_{CD}, D, \lambda)_{del}]$

The rule $S' \to \lambda$ can be simulated by the following matrix:

$$S' : [(\lambda, S', \lambda)_{del}, \ (\lambda, \lambda, \lambda)_{ins}].$$

We claim that $L(\gamma) = L(G)$. First we show that $L(\gamma) \supseteq L(G)$. Let $w_1 A w_2$ be a sequential form in G (initially S) and let $w_1 A w_2 \Rightarrow_r w_1 b C w_2$ be a derivation in G. We show that in γ we obtain the same result:

$$w_1 A w_2 \Longrightarrow_{r.1} w_1 b C w_2.$$

We remark that if the sequence bC is not inserted before A, then the second rule from the matrix will not be applicable (we recall that $w_1 w_2$ does not contain non-terminals from $V \setminus T$ and that $b \neq \lambda$.

Consider now the following derivation in G: $w_1 A w_2 \Rightarrow_r w_1 C b w_2$. This derivation is simulated in γ as follows.

$$\$ w_1 A w_2 \Longrightarrow_{r.1} \$ w_1 \#_1^r \#_2^r w_2 \Longrightarrow_{r.2} \$ w_1 C \#_2^r w_2 \Longrightarrow_{r.3}$$
$$\Longrightarrow_{r.3} \$ w_1 C \#_3^r \#_4^r w_2 \Longrightarrow_{r.4} \$ w_1 C \#_3^r \#_5^r b w_2 \Longrightarrow_{r.5}$$
$$\Longrightarrow_{r.5} w_1 C K^r \#_3^r \#_5^r b w_2 \Longrightarrow_{r.6} w_1 C K^r b w_2 \Longrightarrow^{r.7} \$ w_1 C b w_2.$$

So the grammar G is simulated as follows. Firstly the first stage of the generation is simulated by simulating left-linear and after that right-linear productions. After changing to the second stage, rules $AB.i$ and $CD.i$ $(1 \leq i \leq 3)$ can be applied, removing symbols A, B, C, D.

Now in order to prove the converse inclusion $L(\gamma) \subseteq L(G)$ we show that no other words can be obtained in γ.

We start by observing that matrices $r.1 - r.4$ for a rule $r : A \to Cb$ as well as $r.1$ for a rule $r : A \to bC$ have the form $[(\lambda, x, \lambda)_{ins}, \ (x, y, \lambda)_{del}], x \in V \cup V^2, y \in V$. It is not difficult to see that if x was not already present in the string then such a matrix correspond to the rewriting rule $y \to x$. Indeed, since x is not present in the string, it should have been inserted before the symbol y, which is deleted afterwards.

The matrices $r.5 - r.7$ insure that a sequence of symbols $\#_3^r \#_5^r$ is deleted. This is performed by introducing into the string a new special symbol K^r. If it is not introduced before $\#_3^r$, then nothing happens and K^r can be replaced by \$. Otherwise, it can delete the two symbols in the sequence. The validity of the simulation is ensured by the fact that the symbol \$ is always present in at most one copy.

In a similar way rules $AB.1 - AB.3$ and $CD.1 - CD.3$ act.

In order to conclude that the simulation of the rule $A \to Cb$ does not yield other words we give the following remarks:

- Symbol A is replaced by a pair of symbols $\#_1^r \#_2^r$, where $\#_1^r$ evolves to C and $\#_2^r$ evolves to b.

- Symbol b is inserted if and only if $\#_3^r$ (and $\#_4^r$) is present in the string. This insures that this symbol is separated from any non-terminal that can be derived from C and hence the insertion of this symbol cannot interfere with some other insertion that could be operated.

Since no other words can be generated we can reconstruct a derivation in G starting from a derivation in γ. For this it is enough to follow configurations where there is a non-terminal from $V \setminus \{A, B, C, D\}$ in order to reconstruct the first stage of the derivation from G. The deletion of AB and CD has a direct correspondence to the second stage of G. So, $L(\gamma) = L(G)$. $\qquad\square$

Theorem 7. $Mat_2INS_1^{1,0}DEL_2^{0,0} = RE$.

Proof. The proof is based on a simulation if type-0 grammar in the Geffert normal form (as presented in Section 2). Let $G = (V, T, S, P)$ be such a grammar. We construct the system $\gamma = (V \cup V', T, w, M)$ as follows.

$V' = \{p, p' \mid p : A \to Cb \in P\} \cup \{p, p_2, p_3, \#_p, \#'_p, C_1^p, C_2^p \mid p : A \to bC \in P\} \cup \{X, Y\}$, and $w = \{XSY\}$.

For every rule $p : A \to Cb \in P$ we add to M following matrices:

$$p.1 : [(\lambda, A, \lambda)_{del}, \ (Y, p, \lambda)_{ins}]$$
$$p.2 : [(X, b, \lambda)_{ins}, \ (Y, p', \lambda)_{ins}]$$
$$p.3 : [(X, C, \lambda)_{ins}, \ (\lambda, p'p, \lambda)_{del}]$$

For every rule $p : A \to bC \in P$ we add to M following matrices:

$$p.1 : [(\lambda, A, \lambda)_{del}, \ (Y, p, \lambda)_{ins}]$$
$$p.2 : [(X, C_1^p, \lambda)_{ins}, \ (Y, \#_p, \lambda)_{ins}]$$
$$p.3 : \left[(Y, \#'_p, \lambda)_{ins}, \ (Y, p_2, \lambda)_{ins}\right]$$
$$p.4 : \left[(Y, p_3, \lambda)_{ins}, \ (\lambda, \#'_p\#_p, \lambda)_{del}\right]$$
$$p.5 : [(X, b, \lambda)_{ins}, \ (\lambda, p_2, \lambda)_{del}]$$
$$p.6 : [(\lambda, X, \lambda)_{del}, \ (C_1^p, C_2^p, \lambda)_{ins}]$$
$$p.7 : [(C_2^p, X, \lambda)_{ins}, \ (\lambda, p_3p, \lambda)_{del}]$$
$$p.8 : [(X, C, \lambda)_{ins}, \ (\lambda, C_1^pC_2^p, \lambda)_{del}]$$

For rules $AB \to \lambda \in P$ and $CD \to \lambda \in P$ we add to M following matrices:

$$AB : [(\lambda, AB, \lambda)_{del}, \ (\lambda, \lambda, \lambda)_{ins}] \qquad CD : [(\lambda, CD, \lambda)_{del}, \ (\lambda, \lambda, \lambda)_{ins}]$$

We also add to M the matrices $XY : [(\lambda, X, \lambda)_{del}, \ (\lambda, Y, \lambda)_{del}]$ and $S' : [(\lambda, S', \lambda)_{del}, \ (\lambda, \lambda, \lambda)_{ins}]$.

We claim that $L(\gamma) = L(G)$. First we show that $L(\gamma) \supseteq L(G)$. The simulation uses the following idea. Symbol X marks the site where the non-terminal is

situated, while symbol Y marks a position in the string (for commodity we mark the end of the string). The sequence of insertions and deletions is synchronized between these two positions: inserting something at position X also inserts or deletes symbols at position Y. Finally, symbols at position Y are checked to form some particular order. So in some sense Y corresponds to a "stack" where some information is stored and after that the "stack" is checked to be in some specific form.

More precisely, let $w_1 A w_2$ be a sequential form in G (initially S) and let $w_1 A w_2 \Rightarrow_r w_1 C b w_2$ be a derivation in G. We show that in γ we obtain the same result:

$$w_1 X A w_2 Y \Longrightarrow_{p.1} w_1 X w_2 Y p \Longrightarrow_{p.2} w_1 X b w_2 Y p' p \Longrightarrow_{p.2}^{k-1}$$
$$\Longrightarrow_{p.2}^{k-1} w_1 X b^k w_2 Y p' p^k \Longrightarrow_{p.3} w_1 X C b^k w_2 Y p'^{k-1}.$$

Since there are no rules eliminating p' by itself (it can be eliminated only if p is following it, which is no more possible), the above string can become terminal if and only if one insertion is done at the second step (i.e. $k = 1$). Hence we obtain the string $w_1 X C b w_2 Y$, i.e. we correctly simulated the corresponding production of the grammar.
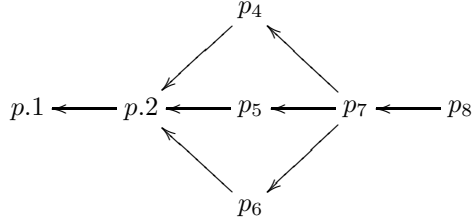
We remark that the rule $p.2$ can be used at any time, but this yields again a symbol p' after Y which cannot be removed.

Now consider the following derivation in G: $w_1 A w_2 \Rightarrow_r w_1 b C w_2$. This derivation is simulated in γ as follows.

$$w_1 X A w_2 Y \Longrightarrow_{p.1} w_1 X w_2 Y p \Longrightarrow_{p.2} w_1 X C_1^p w_2 Y \#_p p \Longrightarrow_{p.3}$$
$$\Longrightarrow_{p.3} w_1 X C_1^p w_2 Y p_2 \#_p' \#_p p \Longrightarrow_{p.4} w_1 X C_1^p w_2 Y p_3 p_2 p \Longrightarrow_{p.5}$$
$$\Longrightarrow_{p.5} w_1 X b C_1^p w_2 Y p_3 p \Longrightarrow_{p.6} w_1 b C_1^p C_2^p w_2 Y p_3 p \Longrightarrow_{p.7}$$
$$\Longrightarrow_{p.7} w_1 b C_1^p C_2^p X w_2 Y \Longrightarrow_{p.8} w_1 b^s X C w_2 Y$$

The deletion rules $AB \to \lambda$, $CD \to \lambda$ and $S' \to \lambda$ are simulated directly by rules AB and CD and symbols X and Y are eliminated by the rule XY. Hence $L(G) \subseteq L(\gamma)$.

Now in order to prove the inclusion $L(\gamma) \subseteq L(G)$ we we show that only specific sequences of rule application can lead to a terminal string. The case of the simulation of rules of type $A \to Cb$ is discussed above. We shall concentrate now on the simulation of rules of type $A \to bC$. We give below the rules' dependency graph, where by $x \leftarrow y$ we indicate that in order to apply y, we should apply at least one time x.

$$p_4$$

$$p.1 \longleftarrow p.2 \longleftarrow p_5 \longleftarrow p_7 \longleftarrow p_8$$

$$p_6$$

Indeed, if rule $p.1$ is not applied first, then additional symbols are added after Y and it is clear that they cannot be eliminated. If $p.3$ is applied before p_2, then the introduced symbol $\#_p'$ can never be deleted (as there is no symbol $\#_p$ afterwards). Rule $p.4$ involves symbols introduced by $p.2$ and $p.3$, so it cannot be used before. Rule $p.5$ cannot be applied before $p.3$, however its application can be interchanged with the application of $p.4$. Rule $p.6$ can be applied once after $p.2$, while in order to apply $p.7$ we need to apply before rules $p.6$, $p.5$ and $p.4$. The rule $p.8$ is applicable only after rule $r.7$.

It is still possible to apply some rules several times. Now we show that each rule must be applied exactly once. Since there is only one copy of A, only one copy of p will be available. Hence rule $p.7$ will be applied only one time. We can also deduce that only one copy of p_3 shall be produced, hence rule $p.4$ should be applied only one time. But this implies the uniqueness of symbols $\#_p$ and $\#_p'$, hence a single application of rules $p.2$ and $p.3$. The last affirmation implies that p_2 is generated only once, hence $p.5$ can be applied only once. From $p.6$ we can deduce that C_2^p is inserted once, hence $p.8$ is executed only one time. Finally, from the $p.2$ we can deduce that C_1^r is inserted only once, so after its deletion in $p.8$ no more copies will remain.

So, we obtain that any terminal derivation in γ needs an application of a specific sequence of rules. Hence, it is enough to look at strings from γ containing a non-terminal from $V \setminus \{A, B, C, D\}$ in order to reconstruct the first stage of the derivation from G. The deletion of AB and CD has a direct correspondence to the second stage of G. This implies that $L(\gamma) \subseteq L(G)$. $\qquad\square$

5 Conclusions

In this article we have introduced the mechanism of a matrix control to the operations of insertion and deletion. We investigated the case of systems with insertion and deletion rules of size $(1, 1, 0; 1, 1, 0)$, $(1, 1, 0; 1, 0, 1)$, $(1, 1, 0; 2, 0, 0)$ and $(2, 0, 0; 1, 1, 0)$ and we have shown that the corresponding matrix insertion-deletion systems are computationally complete. In the case of first two systems matrices of size 3 are used, while in the case of the last two systems binary matrices are sufficient. Since a matrix control is a particular case of a graph control (having an input/output node and series of linear paths starting and ending in this node), we obtain [14] that matrix insertion-deletion systems having rules of size $(2, 0, 0; 2, 0, 0)$ are not computationally complete.

We remark that our results for matrix insertion-deletion systems are different from the results on graph-controlled systems obtained in [4] and previous works. In the graph-controlled case, the total number of nodes in the graph is minimized, while in the matrix case the depth of the graph (corresponding to the size of matrices) is minimized.

We did not succeed to show the computational completeness of systems of size $(1, 1, 0; 1, 1, 0)$ and $(1, 1, 0; 1, 0, 1)$ having binary matrices. This gives an interesting topic for the further research.

References

[1] R. Benne. *RNA Editing: The Alteration of Protein Coding Sequences of RNA*. Ellis Horwood, Chichester, West Sussex, 1993.

[2] F. Biegler, M. J. Burrell, and M. Daley. Regulated RNA rewriting: Modelling RNA editing with guided insertion. *Theor. Comput. Sci.*, 387(2):103 – 112, 2007. Descriptional Complexity of Formal Systems.

[3] M. Daley, L. Kari, G. Gloor, and R. Siromoney. Circular contextual insertions/deletions with applications to biomolecular computation. In *SPIRE/CRIWG*, pages 47–54, 1999.

[4] R. Freund, M. Kogler, Y. Rogozhin, and S. Verlan. Graph-controlled insertion-deletion systems. In I. McQuillan and G. Pighizzini, editors, *Proceedings Twelfth Annual Workshop on Descriptional Complexity of Formal Systems*, volume 31 of *EPTCS*, pages 88–98, 2010.

[5] B. Galiukschov. Semicontextual grammars. *Matem. Logica i Matem. Lingvistika*, pages 38–50, 1981. Tallin University, (in Russian).

[6] V. Geffert. Normal forms for phrase-structure grammars. *ITA*, 25:473–498, 1991.

[7] D. Haussler. *Insertion and Iterated Insertion as Operations on Formal Languages*. PhD thesis, Univ. of Colorado at Boulder, 1982.

[8] D. Haussler. Insertion languages. *Information Sciences*, 31(1):77–89, 1983.

[9] L. Kari. *On Insertion and Deletion in Formal Languages*. PhD thesis, University of Turku, 1991.

[10] L. Kari, G. Păun, G. Thierrin, and S. Yu. At the crossroads of DNA computing and formal languages: Characterizing RE using insertion-deletion systems. In *Proc. of 3rd DIMACS Workshop on DNA Based Computing*, pages 318–333. Philadelphia, 1997.

[11] S. C. Kleene. Representation of events in nerve nets and finite automata. In C. Shannon and J. McCarthy, editors, *Automata Studies*, pages 3–41. Princeton University Press, Princeton, NJ, 1956.

[12] A. Krassovitskiy, Y. Rogozhin, and S. Verlan. Further results on insertion-deletion systems with one-sided contexts. In C. Martín-Vide, F. Otto, and H. Fernau, editors, *Language and Automata Theory and Applications, Second International Conference, LATA 2008, Tarragona, Spain, March 13-19, 2008. Revised Papers*, volume 5196 of *Lecture Notes in Computer Science*, pages 333–344. Springer, 2008.

[13] A. Krassovitskiy, Y. Rogozhin, and S. Verlan. Computational power of P systems with small size insertion and deletion rules. In T. Neary, D. Woods, A. K. Seda, and N. Murphy, editors, *Proceedings International Workshop on The Complexity of Simple Programs, Cork, Ireland, 6-7th December 2008*, volume 1 of *EPTCS*, pages 108–117, 2009.

[14] A. Krassovitskiy, Y. Rogozhin, and S. Verlan. Computational power of insertion-deletion (P) systems with rules of size two. *Natural Computing*, (in publication), 2011.

[15] S. Marcus. Contextual grammars. *Rev. Roum. Math. Pures Appl.*, 14:1525–1534, 1969.

[16] M. Margenstern, G. Păun, Y. Rogozhin, and S. Verlan. Context-free insertion-deletion systems. *Theor. Comput. Sci.*, 330(2):339–348, 2005.

[17] A. Matveevici, Y. Rogozhin, and S. Verlan. Insertion-deletion systems with one-sided contexts. In J. O. Durand-Lose and M. Margenstern, editors, *Machines, Computations, and Universality, 5th International Conference, MCU 2007, Orléans, France, September 10-13, 2007, Proceedings*, volume 4664 of *Lecture Notes in Computer Science*, pages 205–217. Springer, 2007.

[18] G. Păun. *Marcus Contextual Grammars*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[19] G. Păun. *Membrane Computing. An Introduction*. Springer-Verlag, 2002.

[20] G. Păun, G. Rozenberg, and A. Salomaa. *DNA Computing: New Computing Paradigms*. Springer, 1998.

[21] G. Rozenberg and A. Salomaa, editors. *Handbook of Formal Languages*. Springer-Verlag, Berlin, 1997.

[22] W. D. Smith. DNA computers in vitro and in vivo. In R. Lipton and E. Baum, editors, *Proceedings of DIMACS Workshop on DNA Based Computers*, DIMACS Series in Discrete Math. and Theoretical Computer Science, pages 121–185. Amer. Math. Society, 1996.

[23] A. Takahara and T. Yokomori. On the computational power of insertion-deletion systems. In M. Hagiya and A. Ohuchi, editors, *DNA Computing, 8th International Workshop on DNA Based Computers, DNA8, Sapporo, Japan, June 10-13, 2002, Revised Papers*, volume 2568 of *Lecture Notes in Computer Science*, pages 269–280. Springer, 2002.

[24] S. Verlan. On minimal context-free insertion-deletion systems. *Journal of Automata, Languages and Combinatorics*, 12(1-2):317–328, 2007.