Descriptional Complexity of Graph-Controlled Insertion-Deletion Systems

Henning Fernau[1], Lakshmanan Kuppusamy[2(✉)], and Indhumathi Raman[3]

[1] Fachbereich 4 – Abteilung Informatikwissenschaften, Universität Trier, 54286 Trier, Germany
fernau@uni-trier.de

[2] School of Computing Science and Engineering, VIT University, Vellore 632 014, India
klakshma@vit.ac.in

[3] School of Information Technology and Engineering, VIT University, Vellore 632 014, India
indhumathi.r@vit.ac.in

Abstract. We consider graph-controlled insertion-deletion systems and prove that the systems with sizes (i) $(3; 1, 1, 1; 1, 0, 1)$, (ii) $(3; 1, 1, 1; 1, 1, 0)$ and (iii) $(2; 2, 0, 0; 1, 1, 1)$ are computationally complete. Moreover, graph-controlled insertion-deletion systems simulate linear languages with sizes $(2; 2, 0, 1, 1, 0, 0)$, $(2; 2, 1, 0; 1, 0, 0)$, $(3; 1, 0, 1; 1, 0, 0)$, or $(3; 1, 1, 0; 1, 0, 0)$. Simulations of metalinear languages are also studied. The parameters in the size $(k; n, i', i''; m, j', j'')$ of a graph-controlled insertion-deletion system denote (from left to right) the maximum number of components, the maximal length of the insertion string, the maximal length of the left context for insertion, the maximal length of the right context for insertion; a similar list of three parameters concerning deletion follows.

Keywords: Insertion-deletion systems · Graph-controlled systems · Descriptional complexity measures · Computational completeness

1 Introduction

Insertion and deletion operations frequently occur in DNA processing and RNA editing. In the theoretical process of mismatched annealing of DNA sequences, certain segments of the strands are either inserted or deleted [18]. During RNA editing, some fragments of messenger RNA are inserted or deleted [2,3]. The motivation for insertion operations can be found in [7], where this operation and its iterated variant were introduced as a generalization of concatenation and Kleene's closure. The deletion operation was introduced in [10]. Insertion and deletion operations together were introduced into formal language theory in [11]. The corresponding grammatical mechanism is called *insertion-deletion system* (abbreviated as ins-del system). Informally, if a string η is inserted between two

C. Câmpeanu et al. (Eds.): DCFS 2016, LNCS 9777, pp. 111–125, 2016.
DOI: 10.1007/978-3-319-41114-9_9

parts w_1 and w_2 of a string w_1w_2 to get $w_1\eta w_2$, we call the operation *insertion*, whereas if a substring δ is deleted from a string $w_1\delta w_2$ to get w_1w_2, we call the operation *deletion*. Suffixes of w_1 and prefixes of w_2 are called *contexts*.

Several variants of ins-del systems have been considered in literature, like ins-del P systems [1], tissue P systems with ins-del rules [14], context-free ins-del systems [16], matrix ins-del systems [13,17], etc. All the mentioned papers (as well as [19]) attempted to characterize the recursively enumerable languages (*i.e.*, they show computational completeness) using ins-del systems. We refer to the survey article [20] for details of variants thereof.

One of the important variants of ins-del systems is *graph-controlled ins-del systems* introduced in [5] and further studied in [9]. In such a system, the concept of a component is introduced and is associated with every insertion or deletion rule. The transition is performed by choosing any applicable rule from the set of rules of the current component and by moving the resultant string to the target component specified in the rule. If the transition of strings from component to component establishes a tree structure for a given system, then this system can also be seen as an ins-del P system. The objective is to obtain computationally completeness results with few components and small descriptional complexity measures of the ins-del rules.

For an ins-del system, the descriptional complexity measures are based on the size comprising of (i) the maximal length of the insertion string, denoted by n, (ii) the maximal length of the left context and right context used in insertion rules, denoted by i' and i'', respectively, (iii) the maximal length of the deletion string, denoted by m, (iv) the maximal length of the left context and right context used in deletion rules denoted by j' and j'', respectively. The size of an ins-del system is denoted by $(n, i', i''; m, j', j'')$.

Initially, computationally completeness results for graph-controlled ins-del systems were obtained with 5 components [12], then reduced to 4 components with sizes $(1, 1, 0; 2, 0, 0)$, $(2, 0, 0; 1, 1, 0)$, $(1, 1, 0; 1, 1, 0)$, $(1, 1, 0; 1, 0, 1)$ [5] and then later reduced to 3 components with sizes $(1, 2, 0; 1, 1, 0)$, $(1, 1, 0; 1, 2, 0)$ [8]. In [9], even graph-controlled ins-del systems with only 2 components and sizes $(1, 1, 0; 1, 2, 0)$, $(1, 2, 0; 1, 1, 0)$ were shown to be computationally complete. As an ins-del system without graph-control can be seen as a graph-controlled ins-del system with just one component, it is remarkable in this context to note that such system with size $(1, 1, 1; 1, 1, 1)$ are computationally complete; see [19].

In this paper, we prove the computational completeness of the following graph-controlled ins-del systems: (i) 3 components with size $(1, 1, 1; 1, 1, 0)$ or $(1, 1, 1; 1, 0, 1)$; (ii) 2 components with size $(2, 0, 0; 1, 1, 1)$. We also simulate linear grammars by graph-controlled ins-del systems having (i) 3 components with size $(1, 0, 1; 1, 0, 0)$ or $(1, 1, 0; 1, 0, 0)$; (ii) 2 components with size $(2, 0, 1; 1, 0, 0)$ or $(2, 1, 0; 1, 0, 0)$. We also extend the simulation technique to metalinear languages.

2 Preliminaries

We assume that the readers are familiar with the standard notations used in formal language theory. However, we now recall a few notations here.

Let $\mathbb{N}$ denote the set of positive integers, and $[1 \ldots k] = \{i \in \mathbb{N} \colon 1 \leq i \leq k\}$. Given an *alphabet* (finite set) Σ, Σ^* denotes the free monoid generated by Σ. The elements of Σ^* are called *strings* or *words*; λ denotes the empty string. For a string $w \in \Sigma^*$, $|w|$ denotes the length of a string w and w^R denotes the reversal (mirror image) of w. Likewise, L^R and $\mathcal{L}^R$ are understood for languages L and language families $\mathcal{L}$. *RE* denotes the family of the recursively enumerable languages, The family of linear and metalinear languages is denoted by *LIN*, *MLIN*, respectively, where *MLIN* is the smallest language class containing *LIN* and is closed under concatenation. It is known from [15] that *LIN* is neither closed under concatenation nor under Kleene closure whereas *MLIN* is not closed under Kleene closure but closed under concatenation. Also, both *LIN* and *MLIN* are closed under reversal.

For the computational completeness results, we are using the fact that type-0 grammars in the special Geffert normal form are known to characterize the recursively enumerable languages. According to [5], a type-0 grammar $G = (N, T, P, S)$ is said to be in *special Geffert normal form*, SGNF for short, if

- N decomposes as $N = N' \cup N''$, where $N'' = \{A, B, C, D\}$ and N' contains at least the two nonterminals S and S',
- the only non-context-free rules in P are the two erasing rules $AB \to \lambda$ and $CD \to \lambda$,
- the context-free rules are of the following forms:
  $X \to Yb$ or $X \to bY$ where $X, Y \in N'$, $X \neq Y$, $b \in T \cup N''$, or $S' \to \lambda$.

How to construct this normal form is described in [5] and is based on [6]. Also, the derivation of a string is done in two phases. First, the context-free rules are applied repeatedly and the phase I is completed by applying the rule $S' \to \lambda$ in the derivation. In phase II, only the non-context-free erasing rules are applied repeatedly and the derivation ends. It is to be noted that as these context-free rules are more of a linear type, it is easy to see that there can be at most only one nonterminal from N' present in the derivation of G. We exploit this observation in the proofs of Theorems 2 and 4. Also, note that $X \neq Y, X, Y \in N'$ in the context-free rules.

2.1 Insertion-Deletion Systems

We now give the basic definition of insertion-deletion systems, following [11,18].

Definition 1. *An* insertion-deletion system *is a construct* $\gamma = (V, T, A, R)$*, where* V *is an alphabet,* $T \subseteq V$ *is the terminal alphabet,* A *is a finite language over* V*,* R *is a finite set of triplets of the form* $(u, \eta, v)_{ins}$ *or* $(u, \delta, v)_{del}$*, where* $(u, v) \in V^* \times V^*$*,* $\eta, \delta \in V^+$*.*

The pair (u, v) is called the *context*, η is called the *insertion string*, δ is called the *deletion string* and $x \in A$ is called an *axiom*. For all contexts of t where $t \in \{ins, del\}$, if $u = \lambda$ ($v = \lambda$), then we call the operation t to be right context (left context). If both $u, v = \lambda$ for a rule, then it means, the corresponding insertion/deletion can be done freely anywhere in the string and is called context-free insertion/deletion. An insertion rule will be of the form $(u, \eta, v)_{ins}$, which means that the string η is inserted between u and v. A deletion rule will be of the form $(u, \delta, v)_{del}$, which means that the string δ is deleted between u and v. In other words, $(u, \eta, v)_{ins}$ corresponds to the rewriting rule $uv \to u\eta v$, and $(u, \delta, v)_{del}$ corresponds to the rewriting rule $u\delta v \to uv$.

Consequently, for $x, y \in V^*$ we can write $x \Rightarrow y$ if y can be obtained from x by using either an insertion rule or a deletion rule which is given as follows:

1. $x = x_1 uvx_2,\ y = x_1 u\eta vx_2$, for some $x_1, x_2 \in V^*$ and $(u, \eta, v)_{ins} \in R$.
2. $x = x_1 u\delta vx_2,\ y = x_1 uvx_2$, for some $x_1, x_2 \in V^*$ and $(u, \delta, v)_{del} \in R$.

The language generated by γ is defined by

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

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

2.2 Graph-Controlled Insertion-Deletion Systems

A *graph-controlled insertion-deletion system* with k components, or (k-)GCID for short, is a construct $\Pi = (k, V, T, A, H, i_0, i_f, R)$ where

- k is the number of components,
- V is an alphabet,
- $T \subseteq V$ is the terminal alphabet,
- $A \subseteq V$ is a finite set of axioms,
- H is a set of labels associated (in a one-to-one manner) to the rules in R,
- $i_0 \in [1 \dots k]$ is the initial component,
- $i_f \in [1 \dots k]$ is the final or target component, and
- R is a finite set of rules of the form (i, r, j) where r is an insertion rule of the form $(u, \eta, v)_{ins}$ or deletion rule of the form $(u, \delta, v)_{del}$ and $i, j \in [1 \dots k]$.

A rule of the form $l : (i, r, j)$, where $l \in H$ is the label associated to the rule, denotes that the string is sent from component i (for short denoted as Ci) to Cj after the application of the insertion or deletion rule r on the string.

A configuration of Π is represented by $(w)_i$ where i is the number of the current component (initially i_0) and w is the current string. A transition $(w)_i \Rightarrow (w')_j$ is performed if there exists a rule $l : (i, r, j)$ in R such that $w \Rightarrow w'$ on applying the insertion or deletion rule r; in this case, we also write $(w)_i \Rightarrow_l (w')_j$ or $(w')_j \Leftarrow_l (w)_i$. By $(w)_i \underset{\Leftarrow_{l'}}{\overset{\Rightarrow_l}{}} (w')_j$, we mean that $(w')_j$ is derivable from $(w)_i$ using rule l and $(w)_i$ is derivable from $(w')_j$ using rule l'. The language of a graph-controlled insertion-deletion system is the set of all terminal strings in the target component i_f reachable from an axiom and the initial component i_0. Formally,

$$L(\Pi) = \{w \in T^* \mid (x)_{i_0} \Rightarrow^* (w)_{i_f} \text{ for some } x \in A\}.$$

Next, we discuss about the size of a graph-controlled ins-del system. A graph-controlled ins-del system Π is of size $(k; n, i', i''; m, j', j'')$ (with the corresponding language classes denoted by $GCID(k; n, i', i''; m, j', j'')$) if

k = the number of components
n = $\max\{|\eta| \colon (i, (u, \eta, v)_{ins}, j) \in R\}$ (max. length of the inserted string)
i' = $\max\{|u| \colon (i, (u, \eta, v)_{ins}, j) \in R\}$ (max. length of the left context)
i'' = $\max\{|v| \colon (i, (u, \eta, v)_{ins}, j) \in R\}$ (max. length of the right context)
m = $\max\{|\delta| \colon (i, (u, \delta, v)_{del}, j) \in R\}$ (max. length of the deleted string)
j' = $\max\{|u| \colon (i, (u, \delta, v)_{del}, j) \in R\}$ (max. length of the left context)
j'' = $max\{|v| \colon (i, (u, \delta, v)_{del}, j) \in R\}$ (max. length of the right context)

Let us give some examples for GCID systems.

Example 1. The following GCID system Π_1 of size $(2; 1, 0, 0; 0, 0, 0)$ generates the language $L_1 = \{w \in \{a, b\}^* \colon |w|_a = |w|_b\}$.

$$\Pi_1 = (2, \{a, b\}, \{a, b\}, \{\lambda\}, \{r1, r2\}, 1, 1, R)\,,$$

where the rules of R are: $r1 : \ (1, (\lambda, a, \lambda)_{ins}, 2)$, $r2 : \ (2, (\lambda, b, \lambda)_{ins}, 1)$. □

Example 2. With axiom $A = \{ab, \lambda\}$, two rules grouped in singleton components $C1 = \{(1, (a, a, b)_{ins}, 2)\}$, $C2 = \{(2, (a, b, b)_{ins}, 1)\}$, initial and target component $C1$, the GCID system Π_2 can describe $L_2 = \{a^n b^n \colon n \geq 0\}$, *i.e.*, $L_2 \in GCID(2; 1, 1, 1; 0, 0, 0)$. □

Example 3. Consider the GCID system Π_3 of size $(3; 1, 0, 1; 1, 0, 0)$ as follows:

$$\Pi_3 = (3, \{S, S', a, b\}, \{a, b\}, \{SS'\}, H, 1, 1, R)\,,$$

where the rules of R are the following ones:

$$\begin{array}{llll} r1.1: & (1, (\lambda, a, S)_{ins}, 2) & r1.2: & (1, (\lambda, S, \lambda)_{del}, 3) \\ r2.1: & (2, (\lambda, b, S')_{ins}, 1) & & \\ r3.1: & (3, (\lambda, S, S')_{ins}, 1) & r3.2: & (3, (\lambda, S', \lambda)_{del}, 1) \end{array}$$

We claim that Π_3 generates $L_3 = \{a^n b^n \colon n \geq 1\}^*$. We prove our claim by discussing the working of the rules of Π_3 here. Starting with the axiom SS' in $C1$, a is inserted before S and then b is inserted before S' in order, repeatedly, and this leads to $(a^n S b^n S')$ in $C1$. After $n(\geq 0)$ cycles of repetitions, rule $r1.2$ is applied and this deletes S and we move to $C3$ with the string $a^n b^n S'$. We now have a choice of applying rule $r3.1$ or $r3.2$. In the latter case, S' is deleted and the process terminates at the target component $C1$. In the former case, we are back to the starting point in order to generate $a^n b^n a^m b^m SS'$. On repeating this process several times as desired, the process can be terminated by applying the rule $r3.2$. With these arguments, one can see that this system generates L_3. □

Observe the similarities between the examples: L_1 is the iterated shuffle closure of $(L_2 \cup L_2^R)$, while L_3 is the Kleene closure of L_2. Notice that $L_1 \notin LIN$ and $L_3 \notin MLIN$, and the latter can be proved in the same way as argued in [4, p. 137] for the Łukasiewicz language.

3 Auxiliary Results

In order to simplify the proofs of some of our main results, the following observations are helpful.

Theorem 1. *For all non-negative integers k, n, i', i'', m, j, j'', we have that*

$$GCID(k; n, i', i''; m, j', j'') = [GCID(k; n, i'', i'; m, j'', j')]^R.$$

Proof. To an ins-del rule $(x, y, z)_\mu$ with $\mu \in \{ins, del\}$, we associate the reversed rule $\rho(r) = (z^R, y^R, x^R)_\mu$. Let $\Pi = (k, V, T, A, H, i_0, i_f, R)$ be a graph-controlled insertion-deletion system with k components. Map a rule $l\colon (i, r, j) \in \Pi$ to $l\colon (i, \rho(r), j)$ in $\rho(R)$. Define $\Pi^R = (k, V, T, A^R, H, i_0, i_f, \rho(R))$. Then, an easy inductive argument shows that $L(\Pi^R) = (L(\Pi))^R$. Observing the sizes of the system now shows the claim. □

Corollary 1. *Let $\mathcal{L}$ be a language class that is closed under reversal. Then, for all non-negative integers $k, n, i', i'', m, j', j''$, we conclude that*

1. $\mathcal{L} = GCID(k; n, i', i''; m, j', j'')$ *if and only if* $\mathcal{L} = GCID(k; n, i'', i'; m, j'', j')$;
2. $\mathcal{L} \subseteq GCID(k; n, i', i''; m, j', j'')$ *if and only if* $\mathcal{L} \subseteq GCID(k; n, i'', i'; m, j'', j')$.

4 Computational Completeness Results

In this section, we prove the computational completeness results for GCID systems of sizes (i) $(3; 1, 1, 1; 1, 1, 0)$ (ii) $(3; 1, 1, 1; 1, 0, 1)$ and (iii) $(2; 2, 0, 0; 1, 1, 1)$. One may note that, in the first (second) system, the deletion is left context (right context) and in the third system, the insertions are performed in a context-free manner.

Theorem 2. $GCID(3; 1, 1, 1; 1, 1, 0) = RE$.

Proof. Consider a type-0 grammar $G = (N, T, P, S)$ in SGNF. We build a GCID system Π such that $L(\Pi) = L(G)$. Let $\Pi = (3, V, T, \{S\}, H, 1, 1, R)$. The rules in P are labelled injectively with labels from $[1 \dots |P|]$. Let $V = N \cup T \cup \{p\colon p \in [1 \dots |P|]\}$. R is defined as follows. The rules are classified into components $C1$, $C2$ and $C3$ as indicated by the first character following the rule label.
We simulate the rule $p\colon X \to bY$ by the following ins-del rules:

$$\begin{array}{llll} p1.1: & (1, (\lambda, p, X)_{ins}, 2) & & \\ p2.1: & (2, (\lambda, b, p)_{ins}, 3), & p2.2: & (2, (Y, X, \lambda)_{del}, 1) \\ p3.1: & (3, (b, Y, p)_{ins}, 3), & p3.2: & (3, (Y, p, \lambda)_{del}, 2) \end{array}$$

We simulate the rule $q\colon X \to Yb$ by the following ins-del rules:

$$\begin{array}{llll} q1.1: & (1, (\lambda, q, X)_{ins}, 2) & & \\ q2.1: & (2, (\lambda, Y, q)_{ins}, 3), & q2.2: & (2, (\lambda, q, \lambda)_{del}, 1) \\ q3.1: & (3, (q, b, X)_{ins}, 3), & q3.2: & (3, (b, X, \lambda)_{del}, 2) \end{array}$$

We simulate the rule f: $AB \to \lambda$ by the following ins-del rules:

$$\begin{array}{ll} f1.1: & (1,(\lambda,f,A)_{ins},2) \\ f2.1: & (2,(\lambda,f,\lambda)_{del},1), \;\; f2.2: \;\; (2,(f,A,\lambda)_{del},3) \\ f3.1: & (3,(f,B,\lambda)_{del},2) \end{array}$$

We simulate the rule g: $CD \to \lambda$ by the following ins-del rules:

$$\begin{array}{ll} g1.1: & (1,(\lambda,g,C)_{ins},2) \\ g2.1: & (2,(\lambda,g,\lambda)_{del},1), \;\; g2.2: \;\; (2,(g,C,\lambda)_{del},3) \\ g3.1: & (3,(g,D,\lambda)_{del},2) \end{array}$$

We simulate the rule $h: S' \to \lambda$ by the ins-del rule $h1.1: (1,(\lambda,S',\lambda)_{del},1)$.

We now proceed to prove that $L(\Pi) = L(G)$. We do this by explaining how the simulation of the rules of G should work and why no other malicious derivations are possible in Π.

Working of $p: X \to bY$: Consider the string $\alpha X\beta$ in $C1$. Then there is a unique sequence of rule applications in Π as follows.

$$\begin{aligned} &(\alpha X\beta)_1 \Rightarrow_{p1.1} (\alpha pX\beta)_2 \Rightarrow_{p2.1} (\alpha bpX\beta)_3 \Rightarrow_{p3.1} (\alpha bYpX\beta)_3 \\ &\Rightarrow_{p3.2} (\alpha bYX\beta)_2 \Rightarrow_{p2.2} (\alpha bY\beta)_1. \end{aligned}$$

Note that though applying the rule $p3.1$ leaves the string in $C3$ itself, rule $p3.1$ cannot be applied again (the benefit of using double-sided context). Also, only one X of N' is present in the derivation until a $Y \in N'$ is introduced, thus, $p2.2$ cannot be used before the rule $p2.1$ is applied.

Working of $q: X \to Yb$: Consider the string $\alpha X\beta$ in $C1$. On applying rule $q1.1$, we insert q before X and we get $\alpha qX\beta$ in $C2$. Now, we can apply either $q2.1$ or $q2.2$. In the latter case, we delete the just inserted marker q and end up with $\alpha X\beta$ in $C1$ (back to the starting point). Hence, we choose rule $q2.1$ eventually to move on. In this case, consider the following sequence of rule applications in Π.

$$(\alpha X\beta)_1 \underset{\Leftarrow_{q2.2}}{\overset{\Rightarrow_{q1.1}}{\rightleftharpoons}} (\alpha qX\beta)_2 \Rightarrow_{q2.1} (\alpha YqX\beta)_3 \Rightarrow_{q3.1} (\alpha YqbX\beta)_3 \Rightarrow_{q3.2} (\alpha Yqb\beta)_2$$

At this point, we again have a choice of applying rule $q2.1$ or $q2.2$. In the former case, we will again insert a Y before q yielding $\alpha YYqb\beta$ in $C3$. As $Y \in N'$ is the only nonterminal in the string, the first symbol of β cannot be X. Thus, we cannot apply any rule in $C3$ and the derivation stops with nonterminals in a non-target component. In the latter case, by applying $q2.2$ we delete q and get $\alpha Yb\beta$ in $C1$, which is the target component.

We next proceed to discuss the simulation of the non context-free erasing rules $AB \to \lambda$ and $CD \to \lambda$.

Working of $f: AB \to \lambda$: The working of the rule is shown by the following sequence of rule applications.

$$(\alpha AB\beta)_1 \underset{\Leftarrow_{f2.1}}{\overset{\Rightarrow_{f1.1}}{\rightleftharpoons}} (\alpha fAB\beta)_2 \Rightarrow_{f2.2} (\alpha fB\beta)_3 \Rightarrow_{f3.1} (\alpha f\beta)_2 \Rightarrow_{f2.1} (\alpha\lambda\beta)_1$$

Working of $g : CD \to \lambda$: Similar to the working of the rule $f : AB \to \lambda$.

The rule $(1, (\lambda, S', \lambda)_{del}, 1)$ directly erases S'. We start at S in $C1$ and by repeatedly applying the rules p, q, f, g, h, we eventually get $(S)_1 \Rightarrow_* (w)_1$. This proves that $L(G) \subseteq L(\Pi)$.

To prove the reverse relation ($L(\Pi) \subseteq L(G)$), we observe that the rules of Π are applied in groups and each group of rules corresponds to one of p, q, f, g, h. Also, it is not possible to switch between the simulation of some p, say, to that of f, as we always use unique marker symbols to prevent this from happening. This observation completes the proof. □

As RE is known to be closed under reversal, we conclude with Corollary 1:

Theorem 3. *$GCID(3; 1, 1, 1; 1, 0, 1) = RE$.*

Theorem 4. *$GCID(2; 2, 0, 0; 1, 1, 1) = RE$.*

Proof. Consider a type-0 grammar $G = (N, T, P, S)$ in SGNF. We construct a GCID system Π such that $L(\Pi) = L(G)$. Let $\Pi = (2, V, T, \{S\}, H, 1, 1, R)$. The rules from P in G are labelled injectively with labels from $[1 \dots |P|]$. The alphabet of Π is $V = N \cup T \cup \{p, p' : p \in [1 \dots |P|]\}$. R is defined as follows.
We simulate the rule p: $X \to bY$, with $X, Y \in N'$, by the following ins-del rules:

$$\begin{aligned} p1.1 &: \ (1, (\lambda, bY, \lambda)_{ins}, 2) \\ p2.1 &: \ (2, (Y, X, \lambda)_{del}, 1) \end{aligned}$$

We simulate the rule q: $X \to Yb$, with $X, Y \in N'$, by the following ins-del rules:

$$\begin{aligned} q1.1 &: \ (1, (\lambda, Yb, \lambda)_{ins}, 2) \\ q2.1 &: \ (2, (\lambda, X, Y)_{del}, 1) \end{aligned}$$

We simulate the rule f: $AB \to \lambda$, with $A, B \in N''$, by the following ins-del rules:

$$\begin{array}{lll} f1.1: \ (1, (\lambda, ff', \lambda)_{ins}, 2), & f1.2: \ (1, (A, f, f')_{del}, 1), & f1.3: \ (1, (\lambda, A, f')_{del}, 2) \\ f2.1: \ (2, (f', B, \lambda)_{del}, 1), & f2.2: \ (2, (\lambda, f', \lambda)_{del}, 1) & \end{array}$$

We simulate the rule g: $CD \to \lambda$ by the following ins-del rules:

$$\begin{array}{lll} g1.1: \ (1, (\lambda, gg', \lambda)_{ins}, 2), & g1.2: \ (1, (C, g, g')_{del}, 1), & g1.3: \ (1, (\lambda, C, g')_{del}, 2) \\ g2.1: \ (2, (g', D, \lambda)_{del}, 1), & g2.2: \ (2, (\lambda, g', \lambda)_{del}, 1) & \end{array}$$

We simulate the rule $h : S' \to \lambda$ by the ins-del rule $h1.1 : (1, (\lambda, S', \lambda)_{del}, 1)$.
We now proceed to reason why $L(\Pi) = L(G)$.

Working of $p : X \to bY$: Consider a string $\alpha X \beta$ in $C1$. The string bY is free to be inserted anywhere in the string using rule $p1.1$ and the derivation moves to $C2$. Rule $p2.1$ can be applied only if bY is inserted before X. Recall that $X, Y \in N'$ and these types of nonterminals only occur once in valid sentential forms of G (SGNF property). In this case, the X is deleted yielding bY and the derivation ends at the target component $C1$. If bY has been inserted elsewhere, then no rule of $C2$ can be applied and we are trapped in a non-target component with nonterminals in the string.

Working of $q : X \to Yb$: Similar to the working of the rule p as explained above.

Working of $f : AB \to \lambda$: Consider the string $\alpha AB\beta$ in $C1$. We introduce two markers f, f' together anywhere in the string using the rule $f1.1$ and move to $C2$. Suppose that ff' has been inserted between A and B. Now, there is a choice of applying rule $f2.1$ or $f2.2$. In the latter case, we will delete the marker f' and come to the target component $C1$ with $\alpha AfB\beta$. If we introduce ff' again, this will eventually lead to a string having the nonterminals f and A in it, thus not deriving any terminal string. This observation forces one to choose rule $f2.1$ before applying $f2.2$. In this case, there is a unique sequence of rule applications:

$$(\alpha Aff'B\beta)_2 \Rightarrow_{f2.1} (\alpha Aff'\beta)_1 \Rightarrow_{f1.2} (\alpha Af'\beta)_1 \Rightarrow_{f1.3} (\alpha f'\beta)_2 \Rightarrow_{f2.2} (\alpha\lambda\beta)_1$$

Suppose that ff' has not been inserted between A and B, then it is not difficult to see that the derived string will always contain some nonterminals.

Working of $g : CD \to \lambda$: Similar to the working of the rule $f : AB \to \lambda$.

The rule $(1, (\lambda, S', \lambda)_{del}, 1)$ directly erases S'. We start at S in $C1$ and by repeatedly applying the rules p, q, f, g, h, we eventually get $(S)_1 \Rightarrow_* (w)_1$. As argued above, no malicious derivations can lead to terminal strings in $C1$. □

5 (Meta)linear Languages

We next prove that GCID systems of sizes $(2; 2, 1, 0; 1, 0, 0)$, $(2; 2, 0, 1; 1, 0, 0)$, $(3; 1, 1, 0; 1, 0, 0)$, or $(3; 1, 0, 1; 1, 0, 0)$ can simulate all linear languages. In these systems, deletions are performed in a context-free manner. While comparing the last two sizes with the first two sizes, one may note that the length of the inserted string is reduced at the cost of increasing the number of components. We also show how to extend the simulations beyond linear languages.

Theorem 5. *LIN* $\subsetneq$ *GCID*$(2; 2, 1, 0; 1, 0, 0)$.

Proof. Consider a linear grammar $G = (N, T, P, S)$, where every rule of P is of the form $X \to Ya$ or $X \to aY$ or $X \to a$ or $X \to \lambda$. We construct a GCID system $\Pi = (2, V, T, \{S\}, H, 1, 1, R)$ for G. The rules from P in G are labelled injectively with labels from $[1 \dots |P|]$. The alphabet of Π is $V = N \cup T \cup \{p\colon p \in [1 \dots |P|]\}$. The set of rules R of Π is defined as follows.
We simulate the rule $p : X \to Ya$ by the following ins-del rules:

$$\begin{array}{ll} p1.1: \ (1, (X, p, \lambda)_{ins}, 2), & p1.2: \ (1, (p, Ya, \lambda)_{ins}, 2) \\ p2.1: \ (2, (\lambda, X, \lambda)_{del}, 1), & p2.2: \ (2, (\lambda, p, \lambda)_{del}, 1) \end{array}$$

We simulate the rule $q : X \to aY$ by the following ins-del rules:

$$\begin{array}{ll} q1.1: \ (1, (X, q, \lambda)_{ins}, 2), & q1.2: \ (1, (q, aY, \lambda)_{ins}, 2) \\ q2.1: \ (2, (\lambda, X, \lambda)_{del}, 1), & q2.2: \ (2, (\lambda, q, \lambda)_{del}, 1) \end{array}$$

We next simulate the rule $f : X \to a$ by the following ins-del rules:

$$\begin{aligned} f1.1 :&\ (1,(X,a,\lambda)_{ins},2) \\ f2.1 :&\ (2,(\lambda,X,\lambda)_{del},1) \end{aligned}$$

We now prove the theorem by discussing the working of the above rules.

Working of $p : X \to Ya$: Consider the string $\alpha X\beta$ in $C1$. On applying rule $p1.1$, we insert p after X and get $\alpha Xp\beta$ in $C2$. At this point, we have a choice of applying rule $p2.1$ or $p2.2$. In the latter case, the marker p is deleted and we move to $C1$ with $\alpha X\beta$ in the string and this is our starting point. Hence we have to use rule $p2.1$ eventually to proceed. In this case, X is deleted and move to $C1$ with $\alpha p\beta$. At this point, we note that the rule $p1.1$ cannot be applied since in linear grammar there is at most one nonterminal (in this case, X) in the string; this was already deleted in the previous step. With these arguments, we simulate the rule $X \to Ya$ as follows:

$$(\alpha X\beta)_1 \overset{\Rightarrow_{p1.1}}{\underset{\Leftarrow_{p2.2}}{}} (\alpha Xp\beta)_2 \Rightarrow_{p2.1} (\alpha p\beta)_1 \Rightarrow_{p1.2} (\alpha pYa\beta)_2 \Rightarrow_{p2.2} (\alpha Ya\beta)_1.$$

In the above sequence, we note that before the derivation $(\alpha p\beta)_1 \Rightarrow_{p1.2} (\alpha pYa\beta)_2$, the rule $p1.1$ cannot be applied since in a linear grammar there is at most one nonterminal (in this case, X) in the string and it is already deleted in the previous step.

Working of $q : X \to aY$: Similar to the working of the above rule $p : X \to Ya$. The sequence of rule applications in Π is given below for a better understanding.

$$(\alpha X\beta)_1 \overset{\Rightarrow_{q1.1}}{\underset{\Leftarrow_{q2.2}}{}} (\alpha Xq\beta)_2 \Rightarrow_{q2.1} (\alpha q\beta)_1 \Rightarrow_{q1.2} (\alpha paY\beta)_2 \Rightarrow_{q2.2} (\alpha aY\beta)_1.$$

The working of rule $f : X \to a$ is simple and straightforward. Since we start at S in $C1$ and if we repeatedly apply the rules p, q, f, we eventually get $(S)_1 \Rightarrow_* (w)_1$. This proves that $L(G) \subseteq L(\Pi)$.

For the converse direction $L(G) \supseteq L(\Pi)$, observe the remarks that we gave above when explaining the working of the simulations; apart from unnecessary additional loops in the simulation, no successful derivations are possible in Π other than those intended for the simulation of G.

The strictness of the inclusion follows from Examples 1 and 3. $\square$

Remark 1. By allowing for a few more components, we can extend the previous simulation result to cover Kleene closures of linear languages or also *MLIN*. For instance, starting with axiom $S'S$ and a third component containing rules $r3.1 :\ (3,(S',S,\lambda)_{ins},1)$ and $r3.2 :\ (3,(\lambda,S',\lambda)_{del},1)$ and changing $f2.1$ to transit to $C3$, the modified system Π' would describe $(L(G))^+$, or, by having S' as the axiom and starting in $C3$, we can get $(L(G))^*$. $\square$

Likewise, we can describe metalinear languages with three or four components.

Theorem 6. $\mathit{MLIN} \subsetneq \mathit{GCID}(4; 2, 1, 0; 1, 0, 0) \cap \mathit{GCID}(3; 2, 1, 0; 1, 0, 1)$.

Proof. If $L \in MLIN$ happens to be a linear language, we can proceed as in Theorem 5. So, we assume that $L \in MLIN - LIN$ is given. We can think of the work of a metalinear grammar G with $L(G) = L \subseteq T^*$ (generating the concatenation of k linear languages $L(G_1), \ldots, L(G_k)$ with start symbols $S_1, \ldots, S_k$, respectively, and k pairwise disjoint nonterminal alphabets $N_1, \ldots, N_k$) as follows: starting with S_1S_2' as the axiom, first, G_1 generates a terminal word. Then, $S_2' \to S_2S_3'$ is executed, and G_2 generates a terminal word, starting from S_2. This strategy continues, until $S_{k-1}' \to S_{k-1}S_k'$ is executed, followed by the generation of a terminal word by G_{k-1} and finally $S_k' \to S_k$ initiates the last grammar G_k to append a terminal word.

Let us first focus on $GCID(4; 2, 1, 0; 1, 0, 0)$. More formally, we construct a GCID system $\Pi = (4, V, T, \{S_1S_2'\}, H, 1, 1, R)$ for G. Let $V_1, \ldots, V_k$ be the alphabets resulting from the construction of GCID systems Π_i for $G_1, \ldots, G_k$ according to Theorem 5. Let $N_i = V_i - T$ and assume (w.l.o.g.) that $N_1, \ldots, N_k$ are pairwise disjoint. Let $V = \bigcup_{i=1}^{k} V_i \cup \{S_i' \colon i \in [1 \ldots k]\}$. Let R_i be the rule set of Π_i. R_i' coincides with R_i except for (possibly) terminating rules of the type $f2.1$ that target at $C3$ for $i \in [1 \ldots (k-1)]$. Let $R = \bigcup_{i=1}^{k} R_i' \cup R_T$, where R_T collects transition rules that are described in details in the following.

The work of grammar G_i, say, of G_1, is simulated (as described in the proof of Theorem 5). Then, (in general) we transit to the third component. We perform the following transition rules:

$$
\begin{array}{l}
r_{1\to 2}2.1: \ (2, (\lambda, r_{1\to 2}, \lambda)_{del}, 1)\\
r_{1\to 2}3.1: \ (3, (S_2', r_{1\to 2}, \lambda)_{ins}, 4), \ \ r_{1\to 2}3.2: \ (3, (r_{1\to 2}, S_2S_3', \lambda)_{ins}, 2)\\
r_{1\to 2}4.1: \ (4, (\lambda, S_2', \lambda)_{del}, 3)
\end{array}
$$

Similar transition rules are added to start simulations of $G_3, \ldots, G_{k-1}$. Finally, we have the rules:

$$
\begin{array}{l}
r_{k-1\to k}2.1: \ (2, (\lambda, r_{k-1\to k}, \lambda)_{del}, 1)\\
r_{k-1\to k}3.1: \ (3, (S_k', r_{k-1\to k}, \lambda)_{ins}, 4), \ \ r_{k-1\to k}3.2: \ (3, (r_{k-1\to k}, S_k, \lambda)_{ins}, 2)\\
r_{k-1\to k}4.1: \ (4, (\lambda, S_k', \lambda)_{del}, 3)
\end{array}
$$

Observe that the applications of the new rules (in comparison to what is inherited from Theorem 5) is deterministic, and due to the new components, no interference with previously introduced rules is possible. Furthermore, the context-free deletion rules in $C2$ of Theorem 5 will delete only nonterminals of N_i, $i \in [1 \ldots k]$, in the present simulation; hence, they do not interfere with the new nonterminals like S_i'.

We now turn to $GCID(3; 2, 1, 0; 1, 0, 1)$. The only real problem merging $C2$ and $C4$ was that during the simulation of G_i, possibly the symbol S_{i+1}' gets deleted. This can be prevented by requiring the right context of $r_{i\to i+1}$ in the rule that deletes S_{i+1}'. More precisely, the modified rules for P_i will be $r_{i\to i+1}3.1: \ (3, (S_{i+1}', r_{i\to i+1}, \lambda)_{ins}, 2)$ and $r_{i\to i+1}2.2: \ (2, (\lambda, S_{i+1}', r_{i\to i+1})_{del}, 3)$. The remaining technical details are left to the reader.

Remark 1 and more concretely Example 3 shows the claimed strictness of the inclusion. $\square$

Since *LIN* and *MLIN* are known to be closed under reversal [15], by using Corollary 1, we can immediately conclude the next two Theorems (7 and 8):

Theorem 7. $LIN \subsetneq GCID(2; 2, 0, 1; 1, 0, 0)$.

Theorem 8. $MLIN \subsetneq GCID(4; 2, 0, 1; 1, 0, 0) \cap GCID(3; 2, 0, 1; 1, 1, 0)$.

Theorem 9. $LIN \subsetneq GCID(3; 1, 1, 0; 1, 0, 0)$.

Proof. Consider a linear grammar $G = (N, T, P, S)$. We construct a GCID system $\Pi = (3, V, T, \{S\}, H, 1, 1, R)$. The rules from P in G are assumed to be labelled injectively with labels from the set $[1 \dots |P|]$. The alphabet of Π is $V = N \cup T \cup \{p, p' \colon p \in [1 \dots |P|]\}$. The set of rules R of Π is defined as follows.

We simulate the rule $p : X \to Ya$ by the following ins-del rules:

$$\begin{array}{lll} p1.1: \ (1, (X, p, \lambda)_{ins}, 3), & p1.2: \ (1, (p, a, \lambda)_{ins}, 2), & p1.3: \ (1, (p', Y, \lambda)_{ins}, 2) \\ p2.1: \ (2, (p, p', \lambda)_{ins}, 3), & p2.2: \ (2, (\lambda, p', \lambda)_{del}, 1) & \\ p3.1: \ (3, (\lambda, X, \lambda)_{del}, 1), & p3.2: \ (3, (\lambda, p, \lambda)_{del}, 1) & \end{array}$$

We simulate the rule $q : X \to aY$ by the following ins-del rules:

$$\begin{array}{lll} q1.1: \ (1, (X, q, \lambda)_{ins}, 3), & q1.2: \ (1, (q, q', \lambda)_{ins}, 2), & q1.3: \ (1, (q', Y, \lambda)_{ins}, 2) \\ q2.1: \ (2, (q, a, \lambda)_{ins}, 3), & q2.2: \ (2, (\lambda, q', \lambda)_{del}, 1) & \\ q3.1: \ (3, (\lambda, X, \lambda)_{del}, 1), & q3.2: \ (3, (\lambda, q, \lambda)_{del}, 1) & \end{array}$$

We simulate the rule $f : X \to a$ by the following ins-del rules:

$$\begin{array}{l} f1.1: \ (1, (X, a, \lambda)_{ins}, 3) \\ f3.1: \ (3, (\lambda, X, \lambda)_{del}, 1) \end{array}$$

Working of $p : X \to Ya$: Consider the string $\alpha X \beta$ in $C1$. On applying rule $p1.1$, we insert p after X and get $\alpha X p \beta$ in $C3$. At this point, we have a choice of applying rule $p3.1$ or $p3.2$. In the latter case, the marker p is deleted and we move to $C1$ with $\alpha X \beta$ as the string and this is our starting point. Hence, we use rule $p3.1$ eventually to proceed. Then, X is deleted and we move to $C1$ with $\alpha p \beta$. Now, the rule $p1.1$ cannot be applied since in linear grammars there is at most one nonterminal (in this case, X) in the string that was already deleted in the previous step. Hence, we simulate the rule $X \to Ya$ as follows:

$$\begin{aligned} &(\alpha X \beta)_1 \underset{\Leftarrow_{p3.2}}{\overset{\Rightarrow_{p1.1}}{}} (\alpha X p \beta)_3 \Rightarrow_{p3.1} (\alpha p \beta)_1 \Rightarrow_{p1.2} (\alpha p a \beta)_2 \Rightarrow_{p2.1} (\alpha p p' a \beta)_3 \\ &\Rightarrow_{p3.2} (\alpha p' a \beta)_1 \Rightarrow_{p1.3} (\alpha p' Y a \beta)_2 \Rightarrow_{p2.2} (\alpha Y a \beta)_1. \end{aligned}$$

Working of $q : X \to aY$: Consider the string $\alpha X \beta$ in $C1$. On applying rule $q1.1$, we insert q after X and get $\alpha X q \beta$ in $C3$. At this point, we have a choice of applying rule $q3.1$ or $q3.2$. In the latter case, the marker q will be deleted and we move back to the starting point. Hence we have to use rule $q3.1$ eventually to proceed. In this case, X is deleted and we move to $C1$ with $\alpha q \beta$ where q' is inserted after q and the string moves to $C2$ with $\alpha q q' \beta$. In $C2$, we can apply the rule $q2.1$ or $q2.2$. On applying $q2.2$, q' is deleted and the string $\alpha q \beta$ will be in $C1$ and we are back to the previous step. This is also depicted in the following derivation. This forces us to apply the rule $q2.1$ and the sequence of rule applications is shown in the derivation. With these arguments, we simulate the rule $X \to aY$ as follows:

$$(\alpha X\beta)_1 \underset{\Leftarrow_{q3.2}}{\overset{\Rightarrow_{q1.1}}{}} (\alpha Xq\beta)_3 \Rightarrow_{q3.1} (\alpha q\beta)_1 \underset{\Leftarrow_{q2.2}}{\overset{\Rightarrow_{q1.2}}{}} (\alpha qq'\beta)_2 \Rightarrow_{q2.1} (\alpha qaq'\beta)_3$$
$$\Rightarrow_{q3.1} (\alpha aq'\beta)_1 \Rightarrow_{q1.3} (\alpha aq'Y\beta)_2 \Rightarrow_{q2.2} (\alpha aY\beta)_1.$$

The working of rule $f : X \to a$ is simple and straightforward. By repeatedly applying p, q, f, we eventually get $(S)_1 \Rightarrow_* (w)_1$. Thus $L(G) \subseteq L(\Pi)$. Moreover, as argued above, no other derivations are possible for Π, entering $C1$ with a string $\alpha X\beta$. So, by induction, $L(G) \supseteq L(\Pi)$ also follows. □

As *LIN* is known to be closed under reversal, by using Corollary 1, we have:

Theorem 10. $LIN \subsetneq GCID(3; 1, 0, 1; 1, 0, 0)$.

In the literature, $GCID(4; 1, 1, 0; 1, 0, 1)$, $GCID(4; 1, 0, 1; 1, 1, 0)$ (see [5]) and i) $GCID(5; 1, 1, 0; 1, 1, 0)$, ii) $GCID(5; 1, 1, 0; 1, 0, 1)$, iii) $GCID(5; 1, 1, 0; 2, 0, 0)$, iv) $GCID(5; 1, 0, 1; 2, 0, 0)$, v) $GCID(5; 2, 0, 0; 1, 1, 0)$, vi) $GCID(5; 2, 0, 0; 1, 0, 1)$ (see [12]) describe *RE*. Thus, the generative power of $GCID(4; 1, 1, 0; 1, 0, 0)$, $GCID(4; 1, 0, 1; 1, 0, 0)$, $GCID(5; 1, 1, 0; 1, 0, 0)$, $GCID(5; 1, 0, 1; 1, 0, 0)$ is open. In the following, we discuss the power of these systems.

Remark 2. As in Remark 1, one can see that the Kleene star of each of the linear languages lies in $GCID(4; 1, 1, 0; 1, 0, 0) \cap GCID(4; 1, 0, 1; 1, 0, 0)$. Inheriting the proof idea of Theorem 6, we deduce the following from Theorems 9 and 10:

Theorem 11. $MLIN \in GCID(5; 1, 1, 0; 1, 0, 0) \cap GCID(5; 1, 0, 1; 1, 0, 0)$.

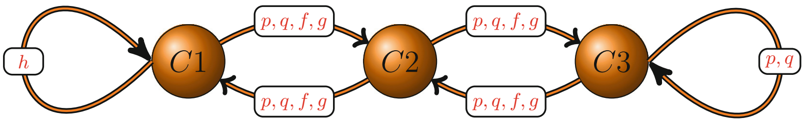

Fig. 1. Control graph structure of Theorem 2; the corresponding simple undirected graph is a path on three vertices, which corresponds to three nested membranes.

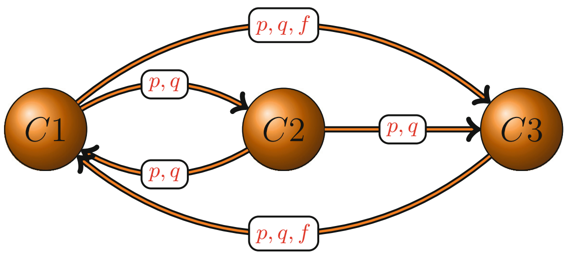

Fig. 2. Control graph structure of Theorem 9; the corresponding simple undirected graph is a cycle on three vertices, which cannot correspond to any nested membrane structure.

6 Conclusions

We have studied GCID systems of various small sizes, proving them to be either computationally complete or able to simulate at least all (meta-)linear languages. Example 2 shows (together with [20]) that two components are more powerful than one for systems of size $(1,1,1;x,y,z)$ with $y+z \leq 1$, $x \in \{0,1\}$. Proving a non-trivial simulation result for the family of context-free languages (say, by GCID systems with size $(3;1,1,0;1,1,0)$) is left open. Also, we have indicated how to simulate Kleene closures of meta-linear languages; it would be therefore interesting to see if the regular closure of the linear languages can be also simulated; refer to [15] for details of this language class.

The *underlying control graph* of a k-GCID system Π is defined to be a graph with k nodes labelled $C1$ through Ck. There exists a directed edge from Ci to Cj if and only if there exists a rule of the form (i, r, j) in R of Π. If the undirected simple graph corresponding to this underlying directed graph is a tree, then Π can be viewed as an insertion-deletion P system (see [5]). In this paper, the underlying graphs of the GCID systems that simulate the families RE and LIN (in Theorems 2, 4 and 5) are trees. Hence, the corresponding results can be immediately also read as results on insertion-deletion P systems. However, one may note that the control graph of the construction of Theorem 9 contains a triangle ($q1.3$ leads from $C1$ to $C2$, $q2.1$ from $C2$ to $C3$ and $q3.1$ from $C3$ to $C1$ in the proof of Theorem 9) and is hence not a tree. Whether or not similar results hold for insertion-deletion P systems remains open. The control graphs of the graph-controlled ins-del systems discussed in this paper are visualized in Figs. 1 and 2 for the case of Theorems 2 and 9, respectively. The annotations given on the edges tells what part of the simulation is responsible for this edge. The according pictures of the simulations in the metalinear cases are even a bit more involved (as we have four components in the first part of Theorem 6) and is hence omitted. However, as there are only connections between $C1$ and $C2$, between $C2$ and $C3$, and between $C3$ and $C4$, this corresponds again to an insertion-deletion P system.

Acknowledgement. The second author would like to acknowledge the project SR/S3/EECE/054/2010, Department of Science and Technology, New Delhi, India.

References

1. Alhazov, A., Krassovitskiy, A., Rogozhin, Y., Verlan, S.: P systems with minimal insertion and deletion. Theoret. Comput. Sci. **412**(1–2), 136–144 (2011)
2. Benne, R. (ed.): RNA Editing: The Alteration of Protein Coding Sequences of RNA. Series in Molecular Biology. Ellis Horwood, Chichester (1993)
3. Biegler, F., Burrell, M.J., Daley, M.: Regulated RNA rewriting: modelling RNA editing with guided insertion. Theoret. Comput. Sci. **387**(2), 103–112 (2007)
4. Chomsky, N., Schützenberger, M.P.: The Algebraic Theory of Context-Free Languages. Studies in Logic and the Foundations of Mathematics, pp. 118–161. North-Holland, Amsterdam (1970)

5. Freund, R., Kogler, M., Rogozhin, Y., Verlan, S.: Graph-controlled insertion-deletion systems. In: McQuillan, I., Pighizzini, G. (eds.) Proceedings Twelfth Annual Workshop on Descriptional Complexity of Formal Systems, DCFS. EPTCS, vol. 31, pp. 88–98 (2010)
6. Geffert, V.: How to generate languages using only two pairs of parentheses. J. Inf. Process. Cybern. EIK **27**(5/6), 303–315 (1991)
7. Haussler, D.: Insertion languages. Inf. Sci. **31**(1), 77–89 (1983)
8. Ivanov, S., Verlan, S.: About one-sided one-symbol insertion-deletion P systems. In: Alhazov, A., Cojocaru, S., Gheorghe, M., Rogozhin, Y., Rozenberg, G., Salomaa, A. (eds.) CMC 2013. LNCS, vol. 8340, pp. 225–237. Springer, Heidelberg (2014)
9. Ivanov, S., Verlan, S.: Universality of graph-controlled leftist insertion-deletion systems with two states. In: Durand-Lose, J., Nagy, B. (eds.) Machines, Computations, and Universality - 7th International Conference, MCU. LNCS, vol. 9288, pp. 79–93. Springer, Switzerland (2015)
10. Kari, L.: On insertion and deletion in formal languages. Ph.D. thesis, University of Turku, Finland (1991)
11. Kari, L., Thierrin, G.: Contextual insertions/deletions and computability. Inf. Comput. **131**(1), 47–61 (1996)
12. Krassovitskiy, A., Rogozhin, Y., Verlan, S.: Computational power of insertion-deletion (P) systems with rules of size two. Nat. Comput. **10**, 835–852 (2011)
13. Kuppusamy, L., Mahendran, A., Krishna, S.N.: Matrix insertion-deletion systems for bio-molecular structures. In: Natarajan, R., Ojo, A. (eds.) ICDCIT 2011. LNCS, vol. 6536, pp. 301–312. Springer, Heidelberg (2011)
14. Kuppusamy, L., Rama, R.: On the power of tissue P systems with insertion and deletion rules. Pre-Proceedings of Workshop on Membrane Computing. Report RGML, vol. 28, pp. 304–318. Univ. Tarragona, Spain (2003)
15. Kutrib, M., Malcher, A.: Finite turns and the regular closure of linear context-free languages. Discrete Appl. Math. **155**(16), 2152–2164 (2007)
16. Margenstern, M., Păun, G., Rogozhin, Y., Verlan, S.: Context-free insertion-deletion systems. Theoret. Comput. Sci. **330**(2), 339–348 (2005)
17. Petre, I., Verlan, S.: Matrix insertion-deletion systems. Theoret. Comput. Sci. **456**, 80–88 (2012)
18. Păun, G., Rozenberg, G., Salomaa, A.: DNA Computing: New Computing Paradigms. Springer, Heidelberg (1998)
19. Takahara, A., Yokomori, T.: On the computational power of insertion-deletion systems. Nat. Comput. **2**(4), 321–336 (2003)
20. Verlan, S.: Recent developments on insertion-deletion systems. Comput. Sci. J. Moldova **18**(2), 210–245 (2010)