Graph-Controlled Insertion-Deletion Systems Generating Language Classes Beyond Linearity

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

[1] Fachbereich 4 – CIRT, Universität Trier, 54286 Trier, Germany
fernau@uni-trier.de
[2] SCOPE, VIT University, Vellore 632 014, India
{klakshma,indhumathi.r}@vit.ac.in
[3] SITE, VIT University, Vellore 632 014, India

Abstract. A regulated extension of an insertion-deletion system known as graph-controlled insertion-deletion (GCID) system has several components and each component contains some insertion-deletion rules. A rule is applied to a string in a component and the resultant string is moved to the target component specified in the rule. When resources are so limited (especially, when deletion is context-free) then GCID systems are not known to describe the class of recursively enumerable languages. Hence, it becomes interesting to find the descriptional complexity of such GCID systems of small sizes with respect to language classes below RE. To this end, we consider closure classes of linear languages. We show that whenever GCID systems describe LIN with t components, we can extend this to GCID systems with just one more component to describe, for instance, 2-LIN and with further addition of one more component, we can extend to GCID systems that describe the rational closure of LIN.

Keywords: Insertion-deletion systems · Graph-controlled systems · Descriptional complexity measures · Closure classes of linear languages

1 Introduction

The origin of insertion systems comes from linguistics, under the name of semi-contextual grammars [6], as well from biology. In biology, the insertion operation is found in the process of mismatched annealing in DNA strands [14] and in RNA editing, some fragments of messenger RNA are inserted or deleted [1]. Further motivation for insertion operations can be seen in [8]. On the other hand, the deletion operation was introduced independently in [10]. Insertion and deletion operations together were introduced in [11]; the corresponding grammatical mechanism is called *insertion-deletion system* (abbreviated as ins-del system). Informally, insertion means inserting a string η between the strings w_1 and w_2, whereas deletion means deleting a substring δ from the string $w_1\delta w_2$.

Among the several variants of ins-del systems (e.g., see [15] for this), we focus on graph-controlled ins-del systems (abbreviated as GCID systems). Such a system was introduced in [5] where the concept of components is introduced,

DOI: 10.1007/978-3-319-60252-3_10

associated with insertion or deletion rules. 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. The descriptional complexity measures are based on the size, denoted by $(k; n, i', i''; m, j', j'')$ where the parameters from left to right denote (i) the number of components k (ii) the maximal length of the insertion string n, (iii) the maximal length of the left context and right context used in insertion rules, i' and i'', respectively, (iv) the maximal length of the deletion string m, (v) the maximal length of the left context and right context used in deletion rules, j' and j'', respectively. We will also refer to the last six numbers in the septuple as *ID size*, where *ID* stands for insertion-deletion.

It is known that the class of linear languages LIN is not closed under concatenation and Kleene closure. Let $\mathcal{L}_{\circ}(\mathrm{LIN})$ and $\mathcal{L}_{*}(\mathrm{LIN})$ denote the super-classes of LIN closed under concatenation and Kleene closure, respectively. It is shown in [3] that if GCID systems can describe LIN with ID size s and t components, then it can be extended to a GCID system with ID size s and $t+1$ components to describe $\mathcal{L}_{*}(\mathrm{LIN})$ and particular cases of GCID systems with ID size s and $t+2$ components describing $\mathcal{L}_{\circ}(\mathrm{LIN})$ were reported. In this paper, we generalize these results to show that even the *rational* or *regular closure* of LIN (denoted as $\mathcal{L}_{reg}(\mathrm{LIN})$) can be described by GCID systems with ID size s and $t+2$ components. We also show that a subclass of $\mathcal{L}_{reg}(\mathrm{LIN})$ containing languages which can be described as concatenation of two languages from $\mathcal{L}_{*}(\mathrm{LIN})$, can be described by GCID systems with ID size s and $t+1$ components. For the first result, we employ a new normal form for $\mathcal{L}_{reg}(\mathrm{LIN})$. Due to space restrictions, many illustrations, examples and proofs have been suppressed.

2 Preliminaries

We assume that the readers are familiar with the standard notations used in formal language theory. However, we recall a few notations. Let $\mathbb{N}$ denote the set of positive integers, and $[1 \ldots k] = \{i \in \mathbb{N} \colon 1 \leq i \leq k\}$. If Σ is an *alphabet* (finite set), then 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^R denotes the reversal (mirror image) of w. Likewise, L^R and $\mathcal{L}^R$ are understood for languages L and language families $\mathcal{L}$. The family of linear, context-free and recursively enumerable languages are denoted by LIN, CF and RE, respectively.

The language class LIN is neither closed under concatenation nor under Kleene closure. This motivates to consider several so-called closure classes of the linear languages. A detailed study of these closure classes is given in [12].

Let $\mathcal{L}_{op}(\mathcal{F})$ be the smallest language class containing $\mathcal{F}$ and being closed under the operation *op*. Since LIN is not closed under concatenation and Kleene closure, the closure classes $\mathcal{L}_{\circ}(\mathrm{LIN})$ and $\mathcal{L}_{*}(\mathrm{LIN})$ are strict supersets of LIN. The class $\mathcal{L}_{\circ}(\mathrm{LIN})$ is the class of metalinear languages. If $L \in \mathcal{L}_{\circ}(\mathrm{LIN})$, then $L = L_1 \circ L_2 \circ \cdots \circ L_k$ (in short $L_1 L_2 \ldots L_k$) for some $k \geq 1$, where $L_i \in \mathrm{LIN}$ for each $1 \leq i \leq k$. Fixing $k \geq 1$, we arrive at the class k-LIN, a subclass of $\mathcal{L}_\circ(\mathrm{LIN})$. In other words, $\mathcal{L}_\circ(\mathrm{LIN}) = \bigcup_{k\geq 1} k$-LIN and LIN $= 1 -$ LIN by definition. Similarly, if $L \in \mathcal{L}_*(\mathrm{LIN})$, then either $L \in \mathrm{LIN}$ or $L = (L')^*$ for some linear language L'. It is well known that $\mathcal{L}_*(\mathrm{LIN})$ and $\mathcal{L}_\circ(\mathrm{LIN})$ are not closed under concatenation and Kleene closure, respectively; see [12]. The class $\mathfrak{L} := \{L_1^* L_2 \mid L_1, L_2 \in \mathrm{LIN}\}$ is also considered as an extension of $\mathcal{L}_*(\mathrm{LIN})$ in [12].[1] It has a nice characterization in terms of pushdown automata with finite turns. Continuing to play around with the concatenation and Kleene closure operators and extending our notation to lists of operators, we have $\mathcal{L}_{\circ,*}(\mathrm{LIN})$, the smallest language family containing LIN and being closed under concatenation and Kleene closure. Recall that $\mathcal{L}_{reg}(\mathrm{LIN})$ is the smallest language family that contains LIN and is closed under the three regular operators: union, concatenation and Kleene closure. In our notation, this corresponds to $\mathcal{L}_{\cup,\circ,*}(\mathrm{LIN})$.

2.1 Graph-Controlled Insertion-Deletion Systems

We define graph-controlled insertion-deletion systems following [5].

Definition 1. *A* graph-controlled insertion-deletion system *(GCID system for short) with k components 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 and $V \setminus T$ is the non-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 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 a deletion rule of the form $(u, \delta, v)_{del}$, with $i, j \in [1 \dots k]$. We say that a GCID system* handles terminals properly *if terminal symbols are only inserted in non-empty contexts containing non-terminals and never get deleted.*

An insertion rule of the form $(u, \eta, v)_{ins}$ means that the string η is inserted between u and v and it corresponds to the rewriting rule $uv \to u\eta v$. Similarly, a deletion rule of the form $(u, \delta, v)_{del}$ means that the string δ is deleted between u and v and this corresponds to the rewriting rule $u\delta v \to uv$. 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.* 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. If the initial component itself is the final component, then we call the system to be a *returning* GCID system.
A graph-controlled ins-del system Π is said to be of size $(k; n, i', i''; m, j', j'')$ if

k is the number of components	
$n = \max\{\lvert\eta\rvert : (i, (u, \eta, v)_{ins}, j) \in R\}$	$m = \max\{\lvert\delta\rvert : (i, (u, \delta, v)_{del}, j) \in R\}$
$i' = \max\{\lvert u\rvert : (i, (u, \eta, v)_{ins}, j) \in R\}$	$j' = \max\{\lvert u\rvert : (i, (u, \delta, v)_{del}, j) \in R\}$
$i'' = \max\{\lvert v\rvert : (i, (u, \eta, v)_{ins}, j) \in R\}$	$j'' = \max\{\lvert v\rvert : (i, (u, \delta, v)_{del}, j) \in R\}$

[1] In [12], $\mathfrak{L}$ was called $\mathcal{L}_*$, which we avoid due to possible confusions with our Kleene closure operator notation.

In general, we follow the convention to use rule label names that are carrying some meaning as follows. For instance, if we like to describe the simulation of a rule p, then this is usually done by several rules in several components, so that $pi.j$ would refer to the jth simulation rule in component Ci. 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. We also associate a simple undirected graph on k nodes to a GCID system of k components as follows: There is an undirected edge from a node Ci to Cj $(i \neq j)$ if and only if there exists a rule of the form (i, r_1, j) or (j, r_2, i) in R of Π. If this underlying undirected simple graph is a tree structure, then we call a returning GCID system *tree-structured*. The language class generated by returning GCID systems of size s is denoted by GCID(s).

We assume the following normal form for linear grammars: $p : X \to aY$, $q : X \to Ya$ and $h : Z \to \lambda$ where $X,\ Y,\ Z \in N$, $a \in T$ as in [2–4]. We also call a returning GCID system *simple-deleting* if it contains one rule of the form $h1.1 : (1, (\lambda, Z, \lambda)_{del}, 1)$ intended to simulate $h : Z \to \lambda$ and that this rule is always the last to be applied in order to obtain a terminal string. For simplicity, we will denote the class of simple-deleting GCID systems (of size s), as well as the corresponding language family, by $\mathrm{GCID}_{SD}(s)$. Moreover, we use the subscript SDT if we want to emphasize that the control graph is tree-structured. With these notations, we rephrase the previous LIN results of [2–4] as follows, omitting the situation when even RE could be characterized.

Proposition 1. $\mathrm{LIN} \subsetneq \mathrm{GCID}_{SD}(3; 1, 1, 0; 1, 0, 0) \cap \mathrm{GCID}_{SD}(3; 1, 0, 1; 1, 0, 0)$ *and* $\mathrm{LIN} \subsetneq \mathrm{GCID}_{SDT}(3; 2, 1, 0; 1, 0, 0) \cap \mathrm{GCID}_{SDT}(3; 2, 0, 1; 1, 0, 0)$.

3 Properties of Closure Classes

In this section, we show some auxiliary results needed to describe the closure classes by GCID system and then provide a characterization of the rational closure of LIN which follows directly from a normal form representation for regular expressions which states that each regular expression can be expressed as finite union of union-free expressions; see [13, Theorem 2].

Proposition 2. *The language classes* $\mathcal{L}_*(\mathrm{LIN})$, $\mathcal{L}_\circ(\mathrm{LIN})$, $\mathfrak{L} \cup \mathfrak{L}^R$, 2-LIN *and* $\mathcal{L}_{reg}(\mathrm{LIN})$ *are all closed under reversal, but* $\mathfrak{L}$ *and* $\mathfrak{L}^R$ *are not.*

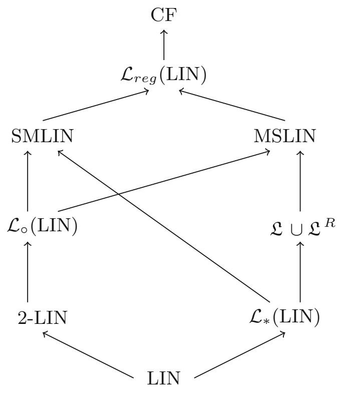
Proof. The positive closure properties follow in a straightforward inductive way from what is known about LIN and some algebraic identities.

Suppose $\mathfrak{L}$ is closed under reversal. Then $(L_1^* L_2)^R = L_3^* L_4$ for some linear languages L_3, L_4 in particular if $L_1 = \{a^n b^n \mid n \geq 1\}$ and $L_2 = \{c^m d^m \mid m \geq 1\}$. Clearly, $(L_1^* L_2)^R$ is not a linear language. Discuss some $d^j w \in L_3$, where w does not start with d. If $w \neq \lambda$, then $(d^j w)^2 \in L_3$ is not a prefix of any word in $(L_1^* L_2)^R$. Hence, $L_3 \subseteq \{d\}^*$. Consider again $d^j \in L_3$ with $j > 0$. Then, $c^j b^r a^r \in L_4$ for some $r \geq 0$. But then, also $(d^j)^2 c^j b^r a^r \in L_3^* L_4$, but $(d^j)^2 c^j b^r a^r \notin (L_1^* L_2)^R$, contradicting our assumption. $\square$

Proposition 3 [12]. *The following inclusions are true. Moreover, all are strict.*

(i) $\mathrm{LIN} \subsetneq \mathcal{L}_*(\mathrm{LIN}) \subsetneq \mathfrak{L} \cup \mathfrak{L}^R \subsetneq \mathcal{L}_{reg}(\mathrm{LIN}) \subsetneq \mathrm{CF}$.
(ii) $\mathrm{LIN} \subsetneq 2\text{-}\mathrm{LIN} \subsetneq \mathcal{L}_\circ(\mathrm{LIN}) \subsetneq \mathcal{L}_{reg}(\mathrm{LIN})$.

Proposition 4 [12]. *The following pairs of language classes are incomparable. (i)* 2-LIN *and* $\mathcal{L}_*(\mathrm{LIN})$*, (ii)* 2-LIN *and* $\mathfrak{L} \cup \mathfrak{L}^R$*, (iii)* $\mathcal{L}_\circ(\mathrm{LIN})$ *and* $\mathcal{L}_*(\mathrm{LIN})$*, (iv)* $\mathcal{L}_\circ(\mathrm{LIN})$ *and* $\mathfrak{L} \cup \mathfrak{L}^R$*.*



The inter-relationship between the closure classes of LIN stated in Propositions 3 and 4 is shown on the left. A (path of) solid arrow from A to B indicates $A \subsetneq B$ and no arrowed path between A and B tells that A and B are incomparable. We also add MSLIN $= \mathcal{L}_\circ(\mathcal{L}_*(\mathrm{LIN}))$ and SMLIN $= \mathcal{L}_*(\mathcal{L}_\circ(\mathrm{LIN}))$.
The following proposition follows directly from Theorem 2 of [13].

Proposition 5. *Let* $L \subseteq T^*$*. Then* $L \in \mathcal{L}_{reg}(\mathrm{LIN})$ *if and only if* L *is the finite union of languages from* $\mathcal{L}_{\circ,*}(\mathrm{LIN})$*.*

Let us now consider a small example that illustrates this proposition. Consider the language L described as follows.

$$L = (L_1'(L_2' \cup (L_3')^*)(L_1' \cup L_2')L_3')^*$$

for linear languages $L_1', L_2', L_3' \subseteq T^*$. Then, we find the following representation:

$$L = ((L_1'L_2'L_1'L_3')^* \circ (L_1'L_2'L_2'L_3')^* \circ (L_1'(L_3')^*L_1'L_3')^* \circ (L_1'(L_3')^*L_2'L_3')^*)^*. \quad (1)$$

Due to the previous proposition, we can focus now on expressions that have only concatenation and Kleene star as operations and whose basic elements are linear languages. Recall the well-known equivalence between expressions and (expression) trees about which we talk in the following. So, the term *subexpression* corresponds to a subtree. In this sense, leaf labels can be subexpressions. Also, we consider Kleene star as a unary operation, but concatenation can take any arity of at least two. This allows us to assume that stars and concatenation always alternate on any path in the expression tree.

In order to describe our grammar constructions that show how to generate all languages from the regular closure of LIN by appropriate GCID systems, we need to specify which of the linear grammars (associated to the leaves of the expression trees) should be simulated 'next', i.e., after finishing with the simulation of the 'current' grammar. This is formalized in the following with the notion of continuation points, reminiscent of the Glushkov transformation [7].

Assume that t is an expression tree with inner nodes labeled $*$ or $\circ$, and the leaves be labeled with numbers from $[1 \ldots k]$. For $i \in [1 \ldots k]$, we define the set of

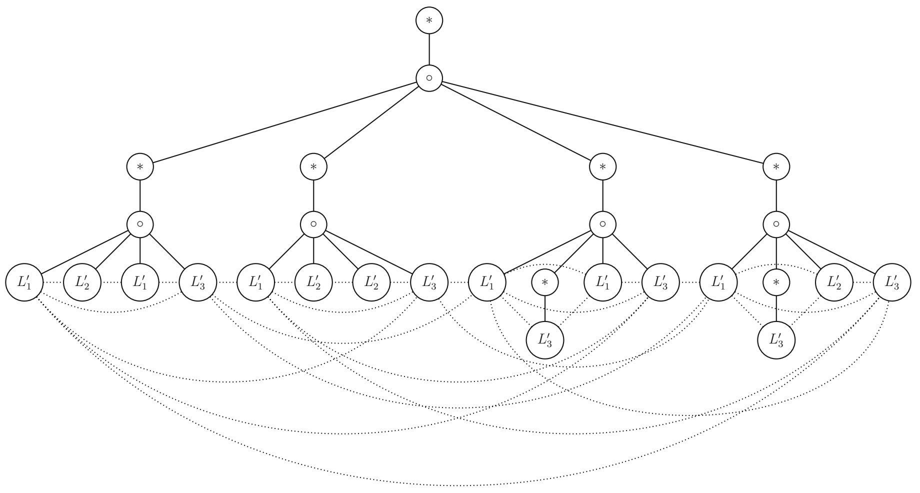


Fig. 1. The expression tree of our example; dotted lines indicate continuation points by joining leaves i and j if $j \in \text{cont}(i)$, suppressing the direction information.

continuation points $\text{cont}(i) \subseteq [1 \ldots k+1]$ as follows. Here, let $\text{subex}(i)$ denote the smallest subexpression to which i belongs, and $r(i)$ be the root label of $\text{subex}(i)$. Moreover, $\text{range}(i)$ be the subinterval of $[1 \ldots k]$ that spans from the first to the last leaf label of $\text{subex}(i)$. Slightly abusing notation, we also write $\text{range}(n)$ for the subinterval of $[1 \ldots k]$ that spans from the first to the last leaf label of the subexpression rooted at some inner node n. Hence, $\text{subex}(i) = \text{subex}(r(i))$. Inductively, we set $\text{subex}^1(i) = \text{subex}(i)$, $r^1(i) = r(i)$ and $\text{range}^1(i) = \text{range}(i)$, as well as $r^j(i) = p(r^{j-1}(i))$, where p is the parent function, $\text{subex}^j(i)$ is the subexpression rooted at $r^j(i)$, and $\text{range}^j(i)$ be the subinterval of $[1 \ldots k]$ that spans from the first to the last leaf label of $\text{subex}^j(i)$. We refer to j also as the *level*. Clearly, at some point $p(r^{j-1}(i))$ is no longer defined, as specified by the *height* $h(i)$. In the following, let $j \leq h(i)$.

– If $r^j(i) = *$ and $i = \max(\text{range}^j(i))$, then $\min(\text{range}^j(i))$ belongs to $\text{cont}(i)$. Moreover, if $j = h(i)$, we include $\max(\text{range}^j(i))+1 = k+1$ as a continuation point.
– If $r^j(i) = \circ$ and $i < \max(\text{range}^j(i))$ and either (a) $j = 1$ or (b) $r^{j-1}(i) = *$ and $i = \max(\text{range}^{j-1}(i))$, then let $s_1, \ldots, s_q$ be all the right siblings of (a) i or (b) the root of $\text{subex}^{j-1}(i)$, respectively, such that the labels of $s_1, \ldots, s_q$ are all $*$ but that of s_q, which is $\circ$, or s_q is a leaf; then, $\min(\text{range}(s_o))$ belongs to $\text{cont}(i)$ for all $1 \leq o \leq q$. As a special case, if there is no s_q with label $\circ$ (because all right siblings carry stars), then we have to continue from the beginning, with the left siblings, again from left to right, until we hit the first s'_q with label $\circ$.

Look again at our example to illustrate this definition, calling the 16 linear languages occurring in the leaves of the expression in Eq. (1) as $L_1, \ldots, L_{16}$ from left to right, also cf. Fig. 1. The following table lists the continuation points.

$i =$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$\text{cont}(i)$	2	3	4	1, 5, 9, 13	6	7	8	1, 5, 9, 13	10, 11	11	12	1, 5, 9, 13	14, 15	15	16	1, 5, 9, 13, 17

For instance, for $i = 1$, we have $r^1(1) = \circ$, as the parent operation of $L_1 = L'_1$ is concatenation, $r^2(1) = *$, as the subexpression $L_1L_2L_3L_4$ is starred, $r^3(1) = \circ$ and $r^4(1) = *$, with $h(1) = 4$. Moreover, $\text{range}^1(1) = [1 \dots 1]$, $\text{range}^2(1) = [1 \dots 4]$, $\text{range}^3(1) = [1 \dots 4]$ and $\text{range}^4(1) = [1 \dots 16]$. For the computation of the continuation points, only $r^1(1) = \circ$ is important and yields 2. The case $i = 4$ is more interesting. Again, we have $r^1(4) = \circ$, $r^2(4) = *$, $r^3(4) = \circ$ and $r^4(4) = *$, with $h(4) = 4$. However, the level $j = 1$ is no longer of interest, rather $j = 2$, which puts $1 = \min(\text{range}^2(4))$ into $\text{cont}(4)$. Moreover, considering the level $j = 3$, we get the first elements of the range of the siblings into the set of continuation points, which is $5 = \min(\text{range}(p(p(5))))$, as $p(p(5))$ describes the right sibling of $p(p(4))$, $9 = \min(\text{range}(p(p(9))))$, as $p(p(9))$ describes the right sibling of $p(p(5))$, as well as 13. The next interesting case happens if $i = 8$, as we now have to continue looking at siblings from the beginning. Finally, with $i = 9$, we see something interesting with $j = 1$, as now the starred subexpression $L^*_{10} = (L'_3)^*$ that follows $L_9 = L'_1$ can be skipped.

Proposition 6. *Let $L \subseteq T^*$. If $L \in \mathcal{L}_{*,\circ}(\text{LIN})$, given by some expression tree t, then there is a context-free grammar $G = (N, T, S'_1, P)$ with $L(G) = L$, together with some integer $k \geq 1$ counting the leaves of t, satisfying the following:*

- *N is partitioned into $N_0, N_1, \dots, N_k$, where for each $i = 1, \dots, k$, $S_i \in N_i$;*
- *$N_0 = \{S'_1, S'_2, S'_3, \dots, S'_k, S'_{k+1}\}$;*
- *P can be partitioned into $P_0, P_1, \dots, P_k$ such that $G_i = (N_i, T, S_i, P_i)$ forms a linear grammar for each $i = 1, \dots, k$;*
- *$P_0 = \{S'_i \to S_iS'_c \mid c \in \text{cont}(i)\} \cup \{S'_{k+1} \to \lambda\}$.*

If the continuation points satisfy $\text{cont}(i) = \{i + 1\}$ for all $1 \leq i \leq k$, then this gives a characterization of the language class $\mathcal{L}_\circ(\text{LIN})$.

In order to simplify some of our main results in the following sections, the following observations from [4] are helpful.

Proposition 7. [4]. *Let $\mathcal{L}$ be a language class that is closed under reversal and k, n, i', i'', m, j, j'' be non-negative integers. The following statements are true.*

1. $\text{GCID}(k; n, i', i''; m, j', j'') = [\text{GCID}(k; n, i'', i'; m, j'', j')]^R$
2. *$\mathcal{L} \subseteq \text{GCID}(k; n, i', i''; m, j', j'')$ iff $\mathcal{L} \subseteq \text{GCID}(k; n, i'', i'; m, j'', j')$*

4 Describing Closure Classes of Linear Languages

Initially, our main objective was to find how much beyond LIN GCID systems (of the four sizes stated in Proposition 1) can lead us. However, we then succeeded to provide a general result showing that if there exists $\text{GCID}_{SD(T)}$ systems of ID size $(n, i', i''; m, j'j'')$ describing LIN, then these constructions can be extended to $\text{GCID}_{SD(T)}$ systems of the same ID size at the expense of two more components to describe $\mathcal{L}_{reg}(\text{LIN})$. Unfortunately, we were not able to describe even CF with GCID systems of these four sizes and this question is left open to the reader.

Describing $\mathcal{L}_{reg}(\text{LIN})$ by GCID systems is rather an immediate consequence of Proposition 6. Here, we slightly extend the notion $\text{cont}(i)$ once more to the case when $i = 0$. This is somehow interesting when $r^{h(1)}(1) = *$ and allows to skip, for instance, to the position $k+1$ to easily incorporate the empty word.

Theorem 1. *For all integers $t, n, m \geq 1$ and $i', i'', j', j'' \geq 0$ with $i' + i'' \geq 1$, if $\text{LIN} \subseteq \text{GCID}_X(t; n, i', i''; m, j', j'')$ for $X \in \{SD, SDT\}$, then $\mathcal{L}_{reg}(\text{LIN}) \subseteq \text{GCID}_X(t+2; n, i', i''; m, j', j'')$.*

Proof. Let $L \in \mathcal{L}_{\cup,\circ,*}(\text{LIN})$ for some $L \subseteq T^*$. By Proposition 5, we can assume that L is the finite union of k languages from $\mathcal{L}_{\circ,*}(\text{LIN})$. We first show how to simulate context-free grammars that are given as in Proposition 6, using $\text{GCID}_X(t+2; n, i', i''; m, j', j'')$ for languages from $\mathcal{L}_{\circ,*}(\text{LIN})$. By using disjoint nonterminal alphabets, we get a GCID system for the finite union of such languages, as well, because we can assume that the constituent systems handle terminals properly.

Since $\text{LIN} \subseteq \text{GCID}_{SD(T)}(t; n, i', i''; m, j', j'')$, each G_i can be simulated by a simple-deleting GCID system $\Pi_i = (t, V_i, T, \{S_i\}, H_i, 1, 1, R_i)$ for $1 \leq i \leq k$, each of size $(t; n, i', i''; m, j', j'')$. We assume, without loss of generality, that $V_i \cap V_j = T$ if $1 \leq i < j \leq k$. Let us first consider the case $i' \geq 1$ and $i'' = 0$. We construct a GCID system Π for G as follows: $\Pi = (t+2, V, T, \{S_0 S'_c \mid c \in \text{cont}(0)\}, H \cup H', 1, 1, (R \setminus \hat{R}) \cup R' \cup R'')$, where

- $V = \left(\bigcup_{i=1}^{k} V_i\right) \cup \{S_0, S'_1, \ldots, S'_{k+1}\}$
- $H' = \bigcup_{i=1}^{k} \{r_i(t+1).1, r_i(t+1).2, r_i(t+2).1\} \cup \{r_{k+1}(t+1).1\}$
- $H = \bigcup_{i=1}^{k} H_i$; $R = \bigcup_{i=1}^{k} R_i$; $\hat{R} = \bigcup_{i=1}^{k} \{h_i 1.1 : (1, (\lambda, X_i, \lambda)_{del}, 1)\}$;
- $R' = \bigcup_{i=0}^{k} \{h_i 1.1 : (1, (\lambda, X_i, \lambda)_{del}, t+1) \mid X_i \to \lambda \in P_i \vee X_0 = S_0\}$
- R'' is the set with the following rules: for each $1 \leq i \leq k$ and $c \in \text{cont}(i)$,

$$\begin{aligned} r_i(t+1).1 &: (t+1, (S'_i, S_i, \lambda)_{ins}, t+2), \\ r_i(t+1).(c+1) &: (t+1, (S_i, S'_c, \lambda)_{ins}, 1) \\ r_i(t+2).1 &: (t+2, (\lambda, S'_i, \lambda)_{del}, t+1); \\ \text{Further}, r_{k+1}(t+1).1 &: (t+1, (\lambda, S'_{k+1}, \lambda)_{del}, 1) \end{aligned}$$

Since $L_i = L(G_i)$ is generated by Π_i, respectively, for $1 \leq i \leq k$, the linear rules of Π_i are simulated by rules of R_i in the first t components and there is no interference between rules of different systems Π_i and Π_j, since $V_i \cap V_j = T$ if $1 \leq i < j \leq k$.

We start with the axiom $S_0S'_c$ for some $c \in \mathrm{cont}(0)$. S_0 is deleted and in $C(t+1)$ and $C(t+2)$, a simulation of G_c is initiated: $(S_0S'_c)_1 \Rightarrow (S'_c)_{t+1} \Rightarrow (S'_cS_c)_{t+2} \Rightarrow (S_c)_{t+1} \Rightarrow (S_cS'_d)_1$ for some $d \in \mathrm{cont}(c)$. Now, a string $w_1 \in L_c$ is produced by simulating G_c in the first t components of the system Π. In general, the simulation goes from left to right. When the string $w_c \in L_c$ is produced, the terminating rule of L_c, namely $h_c.1$, takes the string to component $t+1$, where we either arrive in configuration $(w_cS'_d)_{t+1}$, and the simulation continues with producing a word according to G_d etc. The whole process ends on applying the rule $r_{k+1}(t+1).1 : (t+1, (\lambda, S'_{k+1}, \lambda)_{del}, 1)$, which deletes the nonterminal S'_{k+1}.

Conversely, any derivation within Π can be split into phases, where each *linear phase* starts and ends in the first component with a string that starts with a terminal string, followed by $S_iS'_c$ for some $c \in \mathrm{cont}(i)$ in the beginning, and by some $X_ivS'_c$ in the end of this phase, where v is some terminal string. Now, on applying $h_i1.1$, X_i gets deleted and the *transition phase* is initiated, moving a string starting with a terminal string and ending with some S'_c into $C(t+1)$. Now, apart from the special case when S'_{k+1} is the last symbol of the string, by applying rules from $C(t+1)$ or $C(t+2)$, some string is moved back to $C1$ that satisfies the conditions expressed as the beginning of a linear phase. It is now clearly seen that this alternation of linear and transition phases corresponds to generating words from L from left to right, following some concrete instantiation of the expression tree. The case when $i' = 0$ and $i'' \geq 1$ follows from Propositions 2 and 7. □

5 Reducing Components for Certain Closure Classes

In this section, we show that with GCID systems of ID size s and $t+1$ components we can describe $\mathcal{L}^2_*(\mathrm{LIN}) := \{M_1M_2 : M_1, M_2 \in \mathcal{L}_*(\mathrm{LIN})\}$. Hence, $\mathcal{L}^2_*(\mathrm{LIN}) = 2\text{-LIN} \cup (\mathfrak{L} \cup \mathfrak{L}^R) \cup \{L_1^*L_2^* : L_1, L_2 \in \mathrm{LIN}\}$. We prove the next theorem by providing three different simulations of its three subsets stated above, in the subsequent theorems.

Theorem 2. *For all integers $t, n, m \geq 1$ and $i', i'', j', j'' \geq 0$ with $i' + i'' \geq 1$ and $X \in \{SD, SDT\}$, if $\mathrm{LIN} \subseteq \mathrm{GCID}_X(t; n, i', i''; m, j', j'')$ was shown by a simple-deleting simulation, then $\mathcal{L}^2_*(\mathrm{LIN}) \in \mathrm{GCID}_X(t+1; n, i', i''; m, j', j'')$.*

Theorem 3. *For all integers $t, n, m \geq 1$ and $i', i'', j', j'' \geq 0$ with $i' + i'' \geq 1$ and $X \in \{SD, SDT\}$, if $\mathrm{LIN} \subseteq \mathrm{GCID}_X(t; n, i', i''; m, j', j'')$ was shown by a simple-deleting simulation, then $2\text{-LIN} \subseteq \mathrm{GCID}_X(t+1; n, i', i''; m, j', j'')$.*

Proof. Let $G_1 = (N_1, T, S_1, P_1)$ and $G_2 = (N_2, T, S_2, P_2)$ be linear grammars of L_1 and L_2, respectively, with $N_1 \cap N_2 = \emptyset$ and whose rules are of the form $p_i : X_i \to aY_i$, $q_i : X_i \to Y_ia$ and $h_i : X_i \to \lambda$, with $X_i, Y_i \in N_i$ for $= 1, 2$. Since $\mathrm{LIN} \subseteq \mathrm{GCID}_{SD}(t; n, i', i''; m, j', j'')$, each of the rule types p_i, q_i, h_i can be simulated by rules of a simple-deleting GCID system $\Pi_i = (t, V_i, T, \{S_i\}, H_i, 1, 1, R_i)$ for $i = 1, 2$, each of size $(t; n, i', i''; m, j', j'')$. Hence, the h_i rule type is simulated by the GCID rules $h_i1.1 : (1, (\lambda, X_i, \lambda)_{del}, 1)$. First, consider the case when $i' \geq 1$ and $i'' = 0$. We construct a GCID system $\Pi_3 = (t+1, V_1 \cup V_2 \cup \{\#\}, T, \{\lambda, S_1\#\}, H_1 \cup H_2 \cup \{r1.1, r(t+1).1\}, 1, 1, R)$ for $L(G_1)L(G_2)$, with $R = ((R_1 \cup R_2) \setminus \{h_1 1.1 : (1, (\lambda, X_1, \lambda)_{del}, 1)\}) \cup R'$, where R' has the following three rules: (i) $h_1 1.1 : (1, (\lambda, X_1, \lambda)_{del}, t+1)$, (ii) $r(t+1).1 : (t+1, (\#, S_2, \lambda)_{ins}, 1)$, (iii) $r1.1 : (1, (\lambda, \#, \lambda)_{del}, 1)$.

Starting with the axiom $S_1\#$ and using rules of R_1, a string $w_1 \in L_1$ is produced first with being the last rule applied is $h_1 1.1$. This leads to $w_1\#$ in $C(t+1)$. The only rule in $C(t+1)$ is applied which inserts S_2 after $\#$ and moves back to $C1$. Continuing with $w_1\#S_2$, $w_2 \in L(G_2)$ is generated reaching to the configuration $(w_1\#w_2)_1$ where $\#$ is deleted by $r1.1$. If $r1.1$ is applied before $h_1 1.1$, then the string is stuck at $C(t+1)$ which is not the target component.

Since 2-LIN is closed under reversal (due to Proposition 2), the case when $i' = 0, i'' \geq 1$ follows from Proposition 7. □

Theorem 4. *For all integers $t, n, m \geq 1$ and $i', i'', j', j'' \geq 0$ with $i' + i'' \geq 1$ and $X \in \{SD, SDT\}$, if* LIN $\subseteq$ GCID$_X(t; n, i', i''; m, j', j'')$ *was shown by a simple-deleting simulation, then* $\mathfrak{L} \cup \mathfrak{L}^R \subseteq$ GCID$_X(t+1; n, i', i''; m, j', j'')$.

Proof. Let $G_1 = (N_1, T, S_1, P_1)$ and $G_2 = (N_2, T, S_2, P_2)$ be linear grammars of L_1 and L_2, respectively, with $N_1 \cap N_2 = \emptyset$. Let $i' \geq 1$, $i'' = 0$ and we will now show $\mathfrak{L} \subseteq$ GCID$_{SD}(t+1; n, i', i''; m, j', j'')$. We construct a GCID system $\Pi_4 = (t+1, V_1 \cup V_2 \cup \{\#\}, T, \{\lambda, \#S_2\}, H_1 \cup H_2 \cup \{r(t+1).1, r(t+1).2\}, 1, 1, R)$ for $(L(G_1))^* L(G_2)$, with $R = ((R_1 \cup R_2) \setminus \{h_1 1.1 : (1, (\lambda, X_1, \lambda)_{del}, 1), h_2 1.1 : (1, (\lambda, X_2, \lambda)_{del}, 1)\}) \cup R'$, where R' has the following rules.

$$\begin{array}{ll} h_1 1.1 : (1, (\lambda, X_1, \lambda)_{del}, t+1) & h_2 1.1 : (1, (\lambda, X_2, \lambda)_{del}, t+1) \\ r(t+1).1 : (t+1, (\#, S_1, \lambda)_{ins}, 1) & r(t+1).2 : (t+1, (\lambda, \#, \lambda)_{del}, 1) \end{array}$$

If we start with the axiom $\#S_2$, $w_2 \in L_2$ is produced and $\#w_2$ moves to $C(t+1)$. The rules in $C(t+1)$ initiate the simulation of the rules of G_1 by inserting S_1 after $\#$ and thereafter continuing with $\#S_1 w_2$ from $C1$, the configuration $(\#w_1 w_2)_{t+1}$ is reached, with $w_1 \in L_1^*$ and $w_2 \in L_2$. Now if $r(t+1).1$ is applied, the simulation of G_1 is restarted, and after generating $w_1 \in L(G_1)$ for desired number of times, the whole derivation stops. With this observation, we conclude that Π_4 generates $(L(G_1))^* L(G_2) \in \mathfrak{L}$.

Consider the case when $i' = 1$, but we want to prove the inclusion for $\mathfrak{L}^R$. We aim at constructing a GCID system Π_4' for $L_2 L_1^*$. The simulation is identical to the one just presented except for the axiom, which is $S_2\#$ now.

The case when $i' = 0$ and $i'' \geq 1$ follows from the fact that $\mathfrak{L} \cup \mathfrak{L}^R$ is closed under reversal and by Propositions 2 and 7. □

Theorem 5. *For all integers $t, n, m \geq 1$ and $i', i'', j', j'' \geq 0$ with $i' + i'' \geq 1$ and $X \in \{SD, SDT\}$, if $L_1, L_2 \in$* LIN $\subseteq$ GCID$_X(t; n, i', i''; m, j', j'')$ *was shown by a simple-deleting simulation, then* $L_1^* L_2^* \in$ GCID$_X(t+1; n, i', i''; m, j', j'')$.

The following proof is a simple extension of Theorem 4. Hence, we give the simulating rules and refrain from explaining the working.

Proof. For $i' \geq 1$ and $i'' = 0$, we construct a GCID system $\Pi_3 = (t+1, V_1 \cup V_2 \cup \{\#_1, \#_2\}, T, \{\#_1\#_2\}, H \cup \{r_1(t+1).1, r_2(t+1).1, s_1(t+1).1, s_2(t+1).1\}, t+1, t+1, (R \setminus \hat{R}) \cup R' \cup R'')$ such that $L(\Pi_3) = L_1^* L_2^*$, where

– $\hat{R} = \{h_1 1.1 : (1, (\lambda, X_1, \lambda)_{del}, 1), h_2 1.1 : (1, (\lambda, X_2, \lambda)_{del}, 1)\}$;
– $R' = \{h_1 1.1 : (1, (\lambda, X_1, \lambda)_{del}, t+1), h_2 1.1 : (1, (\lambda, X_2, \lambda)_{del}, t+1)\}$;

– R'' is the set of the following four rules:

$r_1(t+1).1 : (t+1, (\#_1, S_1, \lambda)_{ins}, 1)$ $\quad$ $r_2(t+1).1 : (t+1, (\#_2, S_2, \lambda)_{ins}, 1)$
$s_1(t+1).1 : (t+1, (\lambda, \#_1, \lambda)_{del}, t+1)$ $\quad$ $s_2(t+1).1 : (t+1, (\lambda, \#_2, \lambda)_{del}, t+1)$

The case when $i' = 0$ and $i'' \geq 1$ follows from Propositions 2 and 7. □

Remark 1. The proof of Theorem 5 can be extended to describe $\{L_1^* L_2^* \ldots L_k^* : L_i \in \mathrm{LIN} \text{ for } 1 \leq i \leq k\}$. Consider the GCID system Π' as in Theorem 5 with alphabet and label set extended from 2 to k. Let axiom be $\#_1\#_2 \ldots \#_k$. The rules of $\hat{R}, R' \in \Pi_3$ are similarly extended to k rules and there are $2k$ rules in R''. This shows that $L_1^* L_2^* \ldots L_k^* \in \mathrm{GCID}(t+1; n, i', i''; m, j', j'')$ under the assumptions of Theorem 5. Since there is no control on the number of applications of the rule $r_i(t+1).1 : (t+1, (\#_i, S_i, \lambda)_{ins}, 1)$, we cannot enforce it to be applied exactly once; hence $L_1 L_2$ or $L_1^* L_2$ or $L_1 L_2^*$ alone cannot be produced this way.

Remark 2. By Proposition 1, LIN $\subsetneq$ GCID$(4; 1, 1, 1; 1, 0, 0)$ and by Theorem 2, $\mathcal{L}_*^2(\mathrm{LIN}) \subsetneq \mathrm{GCID}(5; 1, 1, 1; 1, 0, 0)$. By [4], RE $= \mathrm{GCID}(5; 1, 1, 1; 1, 0, 0)$. This opens the quest to prove computational incompleteness results, similarly to the conjecture of Ivanov and Verlan [9] which states that RE $\neq$ GCID(s) if $k = 2$ in $s = (k; 1, i', i''; 1, j', j'')$, with $i', i'', j', j'' \in \{0, 1\}$ and $i' + i'' + j' + j'' \leq 3$.

6 Summary and Future Challenges

Up to the present, most of the research on the descriptional complexity of (graph-controlled) insertion-deletion systems was about the limits to the resources so that we can still show that such systems are able to describe all recursively enumerable languages. Although we do not have a proof showing that the borderline that we reached is optimal, it might be an idea to look into smaller language classes now. One natural question would be to see with which resources we can still describe all context-free languages. While all results collected in this paper show, in particular, that all linear languages can be described by the corresponding resources, we put it up as a challenge to come up with non-trivial simulations of context-free grammars.

In this paper, we tried to bridge between linear and context-free languages as best as possible. Our main technical contribution is to describe these simulations in quite a general fashion, so that we can save giving similar simulations for each specific case of sizes of the systems.

Acknowledgments. Some part of the work done by the second author was during the author's visits to University of Trier, Germany in June-July and December 2016. The possibility to use some overhead money from a DFG grant to support this stay is gratefully acknowledged.

References

1. Benne, R. (ed.): RNA Editing: The Alteration of Protein Coding Sequences of RNA. Series in Molecular Biology. Ellis Horwood, Chichester (1993)
2. Fernau, H., Kuppusamy, L., Raman, I.: Descriptional complexity of graph-controlled insertion-deletion systems. In: Câmpeanu, C., Manea, F., Shallit, J. (eds.) DCFS 2016. LNCS, vol. 9777, pp. 111–125. Springer, Cham (2016). doi:10.1007/978-3-319-41114-9_9
3. Fernau, H., Kuppusamy, L., Raman, I.: Generative power of graph-controlled insertion-deletion systems with small sizes. J. Automata Lang. Comb. (2017)
4. Fernau, H., Kuppusamy, L., Raman, I.: On the computational completeness of graph-controlled insertion-deletion systems with binary sizes. Theoret. Comput. Sci. (2017). doi:http://dx.doi.org/10.1016/j.tcs.2017.01.019
5. Freund, R., Kogler, M., Rogozhin, Y., Verlan, S.: Graph-controlled insertion-deletion systems. In: McQuillan, I., Pighizzini, G. (eds.) Proceedings of Twelfth Annual Workshop on Descriptional Complexity of Formal Systems, DCFS, EPTCS, vol. 31, pp. 88–98 (2010)
6. Galiukschov, B.S.: Semicontextual grammars (in Russian). Mat. Logica i Mat. Ling., Kalinin Univ. 38–50 (1981)
7. Glushkov, V.M.: The abstract theory of automata (in Russian). Russ. Math. Surv. **16**, 1–53 (1961)
8. Haussler, D.: Insertion languages. Inf. Sci. **31**(1), 77–89 (1983)
9. 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). doi:10.1007/978-3-642-54239-8_16
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. Kutrib, M., Malcher, A.: Finite turns and the regular closure of linear context-free languages. Discrete Appl. Math. **155**(16), 2152–2164 (2007)
13. Nagy, B.: A normal form for regular expressions. In: Calude, C.S., Calude, E., Dinneen, M.J. (eds.) Supplemental Papers for DLT 2004, CDMTCS, vol. 252, University of Auckland, New Zealand, Centre for Discrete Mathematics and Theoretical Computer Science (2004)
14. Păun, G., Rozenberg, G., Salomaa, A.: DNA Computing: New Computing Paradigms. Springer, Heidelberg (1998)
15. Verlan, S.: Recent developments on insertion-deletion systems. Comput. Sci. J. Moldova **18**(2), 210–245 (2010)