RAIRO-Theor. Inf. Appl. 52 (2018) 1–21
https://doi.org/10.1051/ita/2018005

RAIRO - Theoretical Informatics and Applications
www.rairo-ita.org

ON DESCRIBING THE REGULAR CLOSURE OF THE LINEAR LANGUAGES WITH GRAPH-CONTROLLED INSERTION-DELETION SYSTEMS

Henning Fernau[1*], Lakshmanan Kuppusamy[2] and Indhumathi Raman[3]

Abstract. A graph-controlled insertion-deletion (GCID) system has several components and each component contains some insertion-deletion rules. A transition is performed by any applicable rule in the current component on a string and the resultant string is then moved to the target component specified in the rule. The language of the system is the set of all terminal strings collected in the final component. When resources are very limited (especially, when deletion is demanded to be context-free and insertion to be one-sided only), then GCID systems are not known to describe the class of recursively enumerable languages. Hence, it becomes interesting to explore the descriptional complexity of such GCID systems of small sizes with respect to language classes below RE and even below CF. To this end, we consider so-called closure classes of linear languages defined over the operations concatenation, Kleene star and union. We show that whenever GCID systems (with certain syntactical restrictions) describe all linear languages (LIN) with t components, we can extend this to GCID systems with just one more component to describe, for instance, the concatenation of two languages from the language family that can be described as the Kleene closure of linear languages. With further addition of one more component, we can extend the construction to GCID systems that describe the regular closure of LIN.

Mathematics Subject Classification. 68Q42, 68Q45, 03D05

Received December 29, 2017. Accepted May 18, 2018.

1. Introduction

The origin of insertion/deletion systems, or ins-del systems for short, comes from linguistics [9], as well from biology, as in [12]; a detailed note is given in [4].

Informally, insertion means putting a string η between the strings w_1 and w_2 to obtain $w_1\eta w_2$. Similarly, deletion means removing a substring δ from the string $w_1\delta w_2$ to obtain the string w_1w_2 from $w_1\delta w_2$. The suffixes of w_1 and the prefixes of w_2 are called left contexts and right contexts of the operation, respectively.

Keywords and phrases: Insertion-deletion systems, graph-controlled systems, descriptional complexity measures, regular closure of linear languages.

[1] Fachbereich 4 – Abteilung Informatikwissenschaften, CIRT, Universität Trier, 54286 Trier, Germany.
[2] School of Computer Science and Engineering, VIT, Vellore 632 014, India.
[3] School of Information Technology and Engineering, VIT, Vellore 632 014, India.

[*] Corresponding author: fernau@uni-trier.de

Several variants of ins-del systems have been considered in the literature, as explained in [4, 13]. Here, we focus on graph-controlled ins-del systems (abbreviated as GCID systems). Such systems were introduced in [8] and further studied in [2, 5, 6]. A GCID system contains several components; these are associated with insertion or deletion rules. The work of a GCID system can be thought of as moving the current sentential form from component to component according to the description of the system. This means that for an application of any rule (associated to some component), the sentential form has to reside in the said component. 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 process starts by putting some axiom in the initial component and then applying transitions until a terminal string appears in a final component; such terminal strings comprise the language associated to a GCID system.

The descriptional complexity measures that are usually considered for GCID systems are summarized in a quantity called *size*, denoted by $(k; n, i', i''; m, j', j'')$, where the parameters from left to right denote the following: (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, and (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*.

The traditional research question is to find the smallest possible sizes that still describe all recursively enumerable languages (denoted RE). This question is hence about the computational completeness of GCID systems of a certain size s. But what can be said if it is not known if size s allows to characterize RE? Or, possibly better, what if it is even known that size s does not yield computational completeness? In some cases of sizes, especially, when the insertion happens with one-sided contexts and the deletion happens without any contexts, it is not even clear if all context-free languages (CF for short) can be described. This less explored question is the starting point of our research here. We are mostly concerned with certain sub-families of CF, like the class of linear languages (written LIN). It is known that LIN is not closed under concatenation and Kleene closure, while CF enjoys positive closure properties. 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 [6] that if GCID systems can describe LIN with ID size s and t components (with some further syntactic constraints), then the according construction can be extended to the following results: GCID systems with ID size s and $t+1$ components suffice to describe $\mathcal{L}_*(\mathrm{LIN})$; particular cases of GCID systems with ID size s and $t+2$ components describing $\mathcal{L}_\circ(\mathrm{LIN})$ were reported. Further classes of languages between linear and context-free were studied in [7, 11]. We will address these as *super-linear* in the following.

In this paper, we generalize these results to show that even the *regular closure* (also known as the *rational closure*) of LIN, denoted as $\mathcal{L}_{reg}(\mathrm{LIN})$, can be described by GCID systems with ID size s and $t+2$ components. Recall that $\mathcal{L}_{reg}(\mathrm{LIN})$ is the smallest language class containing all linear languages and being closed under the operations concatenation, Kleene star and union. 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 normal form for $\mathcal{L}_{reg}(\mathrm{LIN})$ presented in [7].

A preliminary version of this paper has been presented at DCFS 2017; see [3]. This version not only contains all proof details but also some new results. A thorough discussion of super-linear languages (as also done in parts in [3]) can be found in [7].

2. Preliminaries

We assume that the readers are familiar with the standard notations used in formal language theory. However, we recall a few notations in the following. 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* (a finite set of symbols), 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 $\mathcal{L}$, respectively. The families of linear, context-free and recursively enumerable languages are denoted by LIN, CF and RE, respectively.

2.1. Super-linear languages

Being a sub-family of CF, also the linear languages can be described by a particular form of context-free grammars. Namely, a context-free grammar $G = (N, T, S, P)$ is called *linear* if all rules are linear, *i.e.*, their right-hand sides are from $T^* \cup T^*NT^*$. Alternatively, we can assume the following *normal form for linear grammars*: right-hand sides are from $T \cup \{\lambda\} \cup NT \cup TN$. The language class LIN is neither closed under concatenation nor under Kleene closure. This motivates us to consider classes of formal languages built from linear languages by requiring additional closure properties. This is undertaken in the following.

Let $\mathcal{L}_{op}(\mathcal{F})$ be the smallest language class containing the language family $\mathcal{F}$ and being closed under the operation *op*. We will mainly study union, *i.e.*, $op = \cup$, concatenation, *i.e.*, $op = \circ$, and Kleene star, *i.e.*, $op = *$ as operators. Moreover, we will sometimes study sets OP of these operators, meaning that $\mathcal{L}_{OP}(\mathcal{F})$ denotes the smallest language class containing $\mathcal{F}$ and being closed under all operations from OP. For simplicity, we will omit the set brackets in the subscript and write, *e.g.*, $\mathcal{L}_{\cup,\circ}(\mathcal{F})$, when $OP = \{\cup, \circ\}$. Since LIN is closed under union, it is true that $\mathcal{L}_{\cup}(\text{LIN}) = \text{LIN}$. Since LIN is not closed under concatenation and Kleene closure, the closure classes $\mathcal{L}_{\circ}(\text{LIN})$ and $\mathcal{L}_{*}(\text{LIN})$ are strict supersets of LIN.

If $L \in \mathcal{L}_{\circ}(\text{LIN})$, then $L = L_1 \circ L_2 \circ \cdots \circ L_k$ (in short $L_1 L_2 \dots L_k$) for some $k \geq 1$, where $L_i \in \text{LIN}$ for each $1 \leq i \leq k$. Fixing $k \geq 1$, we arrive at the class LIN^k, a subclass of $\mathcal{L}_{\circ}(\text{LIN})$. In particular, $\text{LIN}^2 = \text{LIN} \circ \text{LIN}$ and $\text{LIN} = \text{LIN}^1$ by definition. Clearly,

$$\mathcal{L}_{\circ}(\text{LIN}) = \bigcup_{k \geq 1} \text{LIN}^k \,.$$

We shall also discuss the class of *meta-linear* languages, which can be described as $\mathcal{L}_{\cup,\circ}(\text{LIN})$. Traditionally, one defines $k\text{-LIN} = \mathcal{L}_{\cup}(\text{LIN}^k)$. Notice that

$$\mathcal{L}_{\cup,\circ}(\text{LIN}) = \mathcal{L}_{\cup}(\mathcal{L}_{\circ}(\text{LIN})) = \bigcup_{k \geq 1} k\text{-LIN} = \bigcup_{k \geq 1} \mathcal{L}_{\cup}(\text{LIN}^k)\,.$$

Similarly, if $L \in \mathcal{L}_{*}(\text{LIN})$, then either $L \in \text{LIN}$ or $L = (L')^*$ for some linear language L'. The class $\mathfrak{L} := \{L_1^* L_2 \mid L_1, L_2 \in \text{LIN}\}$ is considered as an extension of $\mathcal{L}_{*}(\text{LIN})$ and LIN^2 in [7, 11].[1] It has a nice characterization in terms of pushdown automata with finite turns. Recall the application of the reversal operation on classes of languages. Hence, we also consider $\mathfrak{L}^R := \{L_2 L_1^* \mid L_1, L_2 \in \text{LIN}\}$, and when $\mathfrak{L}$ is not closed under reversal, it is worth to consider the union of both classes, *i.e.*, $\mathfrak{L} \cup \mathfrak{L}^R$, which coincides with $\mathcal{L}_R(\mathfrak{L})$.

Continuing to play around with the concatenation and Kleene closure operators, we have $\mathcal{L}_{\circ,*}(\text{LIN})$, the smallest language family containing LIN and being closed under concatenation and Kleene closure. Recall that $\mathcal{L}_{reg}(\text{LIN})$ is the smallest language family that contains LIN and is closed under the three regular operators: union, concatenation and Kleene closure, also written as $\mathcal{L}_{\cup,\circ,*}(\text{LIN})$ and is commonly called the regular or rational closure of LIN. All these language families in between LIN and CF will be referred to as *super-linear*. The hierarchical relationships between these super-linear language classes have been exhibited in [7, 11].

2.2. Properties of (new) super-linear language classes

In this paper, we will also come across three further super-linear language classes that, to the best of our knowledge, have not been considered previously.

- $\mathcal{L}_*^2(\text{LIN}) := \{L_1 L_2 : L_1, L_2 \in \mathcal{L}_*(\text{LIN})\}$.
  In other words, $\mathcal{L}_*^2(\text{LIN}) = \mathcal{L}_*(\text{LIN}) \circ \mathcal{L}_*(\text{LIN})$.
- $\mathcal{L}_{\cup}(\mathcal{L}_*^2(\text{LIN}))$ (as a natural extension of the class introduced above).
- $\mathbb{L} = \{L_1^* L_2^* : L_1, L_2 \in \text{LIN}\}$ (as a natural restriction of this class).

[1] In [11], $\mathfrak{L}$ was called $\mathcal{L}_*$, which we avoid due to possible confusions with our Kleene closure operator notation.

Proposition 2.1. *The following assertions are true.*

(1) $\mathcal{L}_*^2(\mathrm{LIN}) = \mathrm{LIN}^2 \cup (\mathfrak{L} \cup \mathfrak{L}^R) \cup \mathbb{L}$.
(2) For $L_1 = \{a^k b^k \mid k \geq 0\}$, $L_2 = \{c^k d^k \mid k \geq 0\}$ *and* $L_3 = \{e^k f^k \mid k \geq 0\}$, *the language* $L_1^* L_2^* L_3^*$ *does not belong to* $\mathcal{L}_\cup(\mathcal{L}_*^2(\mathrm{LIN}))$.
(3) $\mathcal{L}_\cup(\mathcal{L}_*^2(\mathrm{LIN})) \subsetneq \mathcal{L}_{reg}(\mathrm{LIN})$.

Proof.

(1) The stated decomposition follows immediately from the definition of the classes $\mathcal{L}_*^2(\mathrm{LIN})$, LIN^2, $(\mathfrak{L} \cup \mathfrak{L}^R)$, and $\mathbb{L}$.

(2) Assume for the sake of contradiction that there are linear languages $L_{i,j}$ and $k_{i,j} \in \{1, *\}$, for $i \in \{1, \ldots, n\}$, $j = 1, 2$, such that $L_1^* L_2^* L_3^* = \bigcup_{i=1}^n (L_{i,1}^{k_{i,1}} L_{i,2}^{k_{i,2}})$. Each $L_{i,j}$ is described by some linear grammar with, say, $m_{i,j}$ many nonterminals. Let $m = \max_{i=1,\ldots,n; j=1,2}\{m_{i,j}\} + 1$. For $\ell \in \{1, \ldots, n+1\}$, consider the $n+1$ strings $w_\ell = a^{\ell m} b^{\ell m} c^{\ell m} d^{\ell m} e^{\ell m} f^{\ell m}$ from $L_1^* L_2^* L_3^*$. By pigeon-hole principle, there must be some $\ell_1, \ell_2 \in \{1, \ldots, n+1\}$, $\ell_1 \neq \ell_2$, and correspondingly an $i \in \{1, \ldots, n\}$ such that $w_{\ell_1}, w_{\ell_2} \in L_{i,1}^{k_{i,1}} L_{i,2}^{k_{i,2}}$. Hence, there are words $u_{\ell_1}, u_{\ell_2} \in L_{i,1}^{k_{i,1}}$ and $v_{\ell_1}, v_{\ell_2} \in L_{i,2}^{k_{i,2}}$ such that $w_{\ell_1} = u_{\ell_1} v_{\ell_1}$ and $w_{\ell_2} = u_{\ell_2} v_{\ell_2}$. Notice that by construction, both $w_{\ell_1 \ell_2} := u_{\ell_1} v_{\ell_2}$ and $w_{\ell_2 \ell_1} := u_{\ell_2} v_{\ell_1}$ belong to $L_{i,1}^{k_{i,1}} L_{i,2}^{k_{i,2}}$. Now, we apply an interchange argument. If u_{ℓ_1} ends with the same letter with which v_{ℓ_1} starts, or if u_{ℓ_1} ends with an a and v_{ℓ_1} starts with a b, or if u_{ℓ_1} ends with a c and v_{ℓ_1} starts with a d, or if u_{ℓ_1} ends with an e and v_{ℓ_1} starts with an f, then clearly $w_{\ell_1 \ell_2}$ does not belong to $L_1^* L_2^* L_3^*$. The only remaining cases are: (i) u_{ℓ_1} ends with a b and v_{ℓ_1} starts with a c, or (ii) u_{ℓ_1} ends with a d and v_{ℓ_1} starts with an e. Both cases are completely symmetric due to the structure of the languages. Hence, assume that (ii) holds. This means that either (a) $u_{\ell_1} = a^{\ell_1 m} b^{\ell_1 m} c^{\ell_1 m} d^{\ell_1 m} \in L_{i,1}$ or (b) $u_{\ell_1} = a^{\ell_1 m} b^{\ell_1 m} c^{\ell_1 m} d^{\ell_1 m} \in L_{i,1}^*$. By the choice of m, both assumptions lead to a contradiction by pumping arguments.
Suppose that (a) holds; then, by the pumping lemma for linear languages with $p = m_{i,1}$ as the pumping constant, and by definition of m, it is clear that $p \leq m$. Hence, u_{ℓ_1} with $|u_{\ell_1}| = 4\ell_1 m \geq p$ can be factorized as $stxyz$ such that $|styz| \leq p \leq \ell_1 m$ and $|ty| \geq 1$. This implies that $t = a^{k_1}$ and $y = a^{k_2}$ for some $k_1, k_2 \geq 1$. By pumping in a t and y into u_{ℓ_1} we have $u'_{\ell_1} = a^{\ell_1 m + k_1 + k_2} b^{\ell_1 m} c^{\ell_1 m} d^{\ell_1 m} \in L_{i,1}$. Hence, $u'_{\ell_1} e^q f^q \in L_{i,1} L_{i,2}^{k_{i,2}}$ for $q = \ell_1 m$ by construction. Clearly, $u'_{\ell_1} e^q f^q$ does not belong to $L_1^* L_2^* L_3^*$ for any q, giving the desired contradiction.
Suppose that (b) holds; then $u_{\ell_1}^2 \in L_{i,1}^*$ also. Hence by construction,

$$u_{\ell_1}^2 e^{\ell_1 m} f^{\ell_1 m} = a^{\ell_1 m} b^{\ell_1 m} c^{\ell_1 m} d^{\ell_1 m} a^{\ell_1 m} b^{\ell_1 m} c^{\ell_1 m} d^{\ell_1 m} e^{\ell_1 m} f^{\ell_1 m} \in L_{i,1}^* L_{i,2}^{k_{i,2}}$$

Clearly by its structure, the string $u_{\ell_1}^2 e^{\ell_1 m} f^{\ell_1 m}$ does not lie in $L_1^* L_2^* L_3^*$, giving the desired contradiction.

(3) The inclusion follows directly from the definitions of the two languages. The language $L_1^* L_2^* L_3^*$ stated in item (2) acts as a witness language to justify the strictness of the inclusion.

□

In [7], several closure properties and characterization results for super-linear language classes have been derived. For this paper, the following ones are useful.

Proposition 2.2 ([7])**.** *The classes* LIN, LIN^2, 2-LIN, $\mathfrak{L} \cup \mathfrak{L}^R$, $\mathcal{L}_\cup(\mathfrak{L} \cup \mathfrak{L}^R)$ *and* $\mathcal{L}_{reg}(\mathrm{LIN})$ *are closed under reversal.*

In the same lines, we show that the following.

Proposition 2.3. *The classes* $\mathbb{L}$, $\mathcal{L}_\cup(\mathbb{L})$, $\mathcal{L}_*^2(\mathrm{LIN})$ *and* $\mathcal{L}_\cup(\mathcal{L}_*^2(\mathrm{LIN}))$ *are closed under reversal.*

Proof. $\mathbb{L}$ is closed under reversal, since $(L_1^* L_2^*)^R = (L_2^*)^R (L_1^*)^R = (L_2^R)^* (L_1^R)^*$, as LIN is closed under reversal. By Proposition 2.1, each $L \in \mathcal{L}_*^2(\mathrm{LIN})$ belongs to one of three classes $\mathrm{LIN}^2, (\mathfrak{L} \cup \mathfrak{L}^R), \mathbb{L}$, each of which is closed under reversal, so that $L^R \in \mathcal{L}^2_*(\mathrm{LIN})$ follows. As $(L_1 \cup L_2)^R = L_1^R \cup L_2^R$, this also entails that $\mathcal{L}_\cup(\mathcal{L}^2_*(\mathrm{LIN}))$ and $\mathcal{L}_\cup(\mathbb{L})$ are closed under reversal. □

We next provide some rewriting grammars for certain super-linear grammars.

Proposition 2.4 ([7])**.** *Let $L \subseteq T^*$. Then, $L \in \mathcal{L}_{reg}(\mathrm{LIN})$ if and only if there is a context-free grammar $G = (N, T, S, P)$ with $L(G) = L$ that satisfies the following properties.*

- *N can be partitioned into N_0 and N'.*
- *There is a right-linear grammar $G_R = (N_0, N', S, P_0)$.*
- *N' can be further partitioned into $N_1, \dots, N_k$ for some k, such that the restriction P_i of P involving symbols from $N_i \cup T$ are only linear rules, with T serving as the terminal alphabet.*
- *P can be partitioned into $P_0, P_1, \dots, P_k$.*

Notice that this characterization corresponds to a two-stage approach: First, the right linear grammar G_R is used to produce a sequence of symbols from N' that both serve as terminal symbols for G_R and as nonterminal symbols for linear grammar G_i that can be obtained from G by using rules P_i only. Here, it is not necessary but possible to insist on using $N'' \subseteq N'$ instead of N' as the terminal alphabet of G_R, such that $N'' \cap N_i = \{S_i\}$ for each $i \in [1 \dots k]$, *i.e.*, we can single out a start symbol S_i for each G_i. Clearly, the *linear rules* mentioned in the previous proposition can be assumed to be in normal form; this remark also applies to the following characterization results of [7] without further explicit mentioning its proof.

Proposition 2.5 ([7])**.** *Let $L \in T^*$. Then $L \in \mathfrak{L}$ if and only if there are two linear grammars $G_1 = (N_1, T, S_1, P_1)$ and $G_2 = (N_2, T, S_2, P_2)$ (w.l.o.g., with $N_1 \cap N_2 = \emptyset$) such that $L = L(G_1)^* L(G_2)$ if and only if there is a context-free grammar $G = (N, T, S, P)$ (based on G_1, G_2) with $L(G) = L$ such that, for $S, S' \notin N_1 \cup N_2$,*

- *$N = N_1 \cup N_2 \cup \{S, S'\}$ (partitioning of N);*
- *$P = P_1 \cup P_2 \cup \{S \to S'S_2, S' \to S'S_1\,, S' \to \lambda\}$.*

In the spirit of Proposition 2.5, we can also state:

Proposition 2.6. *Let $L \in T^*$. Then $L \in \mathbb{L}$ if and only if there are two linear grammars $G_1 = (N_1, T, S_1, P_1)$ and $G_2 = (N_2, T, S_2, P_2)$ (w.l.o.g., with $N_1 \cap N_2 = \emptyset$) such that $L = L(G_1)^* L(G_2)^*$ if and only if there is a context-free grammar $G = (N, T, S, P)$ (based on G_1, G_2) with $L(G) = L$ such that, for $S, S' \notin N_1 \cup N_2$,*

- *$N = N_1 \cup N_2 \cup \{S, S_1', S_2'\}$ (partitioning of N);*
- *$P = P_1 \cup P_2 \cup \{S \to S_1'S_2', S_1' \to S_1'S_1, S_1' \to \lambda, S_2' \to S_2'S_2, S_2' \to \lambda\}$.*

2.3. Graph-controlled insertion-deletion systems

We define graph-controlled insertion-deletion systems according to [8].

Definition 2.7. 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 $i, j \in [1 \dots k]$, i is the current component where the rule r is applied and j is the target component of where the resultant string moves.

We say that a GCID system *handles terminals properly* if terminal symbols are (i) inserted in non-empty contexts with only non-terminals and (ii) never 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*. If one of the u or v is λ for all the insertion (deletion) contexts, then we call the insertion (deletion) *one-sided*. If both $u, v = \lambda$ for every insertion (deletion) rule, then it means that the corresponding insertion (deletion) can be done freely anywhere in the string and is called *context-free* insertion (context-free deletion). Ci refers to component i. 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 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. 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 for rule p in component Ci.

A *configuration* of $\mathit{\Pi}$ is represented by $(w)_i$, where i is the number of the current component (initially i_0) and w is the current string. In that case, we also say that w has entered component Ci. We denote by $(w)_i \Rightarrow_l (w')_j$ (or simply $(w)_i \Rightarrow (w')_j$ in case the rule application is clear) if $(w')_j$ is derived from $(w)_i$ on applying a rule $l : (i, r, j)$ in R. In such a case, we also say that w' moves from Ci to Cj (after applying rule l). Let $\Rightarrow_*$ denote the reflexive transitive closure of $\Rightarrow$. We define $L(\mathit{\Pi}) = \{w \in T^* \mid (S)_{i_0} \Rightarrow_* (w)_{i_0}\}$. The *size* of $\mathit{\Pi}$ is denoted as $(k; n, i', i''; m, j', j'')$, where k is the number of components and

$$\begin{array}{ll} n = \max\{|\eta| : (i, (u, \eta, v)_{ins}, j) \in R\} & m = \max\{|\delta| : (i, (u, \delta, v)_{del}, j) \in R\} \\ i' = \max\{|u| : (i, (u, \eta, v)_{ins}, j) \in R\} & j' = \max\{|u| : (i, (u, \delta, v)_{del}, j) \in R\} \\ i'' = \max\{|v| : (i, (u, \eta, v)_{ins}, j) \in R\} & j'' = \max\{|v| : (i, (u, \delta, v)_{del}, j) \in R\} \end{array}$$

The *underlying control graph* of a graph-controlled insertion-deletion system $\mathit{\Pi}$ with k components is defined to be a graph on k nodes labelled $C1$ through Ck. There exists a directed edge from a node Ci to node Cj if and only if there exists a rule of the form (i, r, j) in R of $\mathit{\Pi}$. 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 $\mathit{\Pi}$. If this underlying undirected simple graph is a tree, then $\mathit{\Pi}$ can be viewed as an insertion-deletion P system (see [8]). In such a case, let us call a returning GCID system *tree-structured*. The language class generated by returning GCID systems of size s is denoted by GCID(s). The class of languages generated by tree-structured GCID systems of size s is denoted by $\mathrm{GCID}_T(s)$. By definition, we know that $\mathrm{GCID}_T(s) \subseteq \mathrm{GCID}(s)$. Notice that in all our constructions, the derived GCID systems will be returning. Only when necessary, we will underline this fact again by writing $\mathrm{GCID}_R(s)$ for the corresponding language class.

2.4. Simple properties of GCID

In the following proposition, we discuss a closure property that is usually considered to be easy. Yet, in order to state and prove it in a general form, we need to be careful. This also underlines the importance of several properties of GCID systems that we introduced above.

Proposition 2.8. *Let τ be some tree and s be some size measure. Let $\mathit{\Pi}_1$ and $\mathit{\Pi}_2$ be two returning GCID systems of size (at most) s with terminal alphabet Σ, whose control graphs are subtrees of τ, that handle terminals properly. Then there is a returning GCID system $\mathit{\Pi}$ with terminal alphabet Σ of size (at most) s whose control graph is a subtree of τ and that handles terminals properly, such that $L(\mathit{\Pi}) = L(\mathit{\Pi}_1) \cup L(\mathit{\Pi}_2)$.*

Proof. The construction is very simple: w.l.o.g., we can assume that the nonterminal alphabets of $\mathit{\Pi}_1$ and $\mathit{\Pi}_2$ are disjoint; their union is the nonterminal alphabet of $\mathit{\Pi}$. Now, let the axiom set A of $\mathit{\Pi}$ be simply the union of the axiom sets A_1 of Π_1 and A_2 of Π_2; likewise, the rule set associated to the components in Π are simply the unions of the rule sets associated to that component in Π_1 and in Π_2. As Π_1 and Π_2 possess the same control (super-)structure τ, this is also true for Π. Because Π_1 and Π_2 handle terminals properly, the derivations starting with axioms from A_1 cannot use rules from Π_2 and vice versa, so that the language generated by Π is $L(\Pi_1) \cup L(\Pi_2)$. It is clear that Π handles terminals properly, as its constituent systems Π_1 and Π_2 do. □

We will use the previous result in the following way.

Remark 2.9. Whenever we show that a language family LAN is contained in some language family $\mathcal{L}$ described by certain GCID systems, we will make sure that this GCID family obeys the conditions of Proposition 2.8, so that we can immediately conclude that $\mathcal{L}_\cup(\mathrm{LAN})$ is also included in the same language class $\mathcal{L}$.

We also call a returning GCID system that handles terminals properly *simple-deleting* if it contains one rule of the form $h1.1 : (1, (\lambda, Z, \lambda)_{del}, 1)$ (intended to simulate a deletion rule $h : Z \to \lambda$) and we assume that this rule is always the last one 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.

3. Examples of super-linear languages and GCID

We now provide a couple of example languages to show that even GCID systems with very restricted resources can generate quite interesting languages. These examples will also serve as witness languages to show the strictness of several inclusions between language families that we are going to prove later. Finally, we also make clear in this way how (non-trivial) GCID systems work and how to prove that GCID systems actually do what they are supposed to do.

Example 3.1. Consider the language

$$L_{\#,\$} = \{\#a^{n_1}b^{n_1}\#a^{n_2}b^{n_2}\dots\#a^{n_p}b^{n_p}\$c^{k_1}d^{k_1}\$c^{k_2}d^{k_2}\dots\$c^{k_p}d^{k_p} \mid p, n_1, \dots, n_p, k_1, \dots, k_p \geq 1\} \cup \{\lambda\}\,.$$

We will prove that $L_{\#,\$} \in (\mathrm{CF} \cap \mathrm{GCID}_{SDT}(5; 2, 1, 0; 1, 0, 0)) \setminus \mathcal{L}_{reg}(\mathrm{LIN})$.

1. The language is context-free, since the following simple type-2 rules generate $L_{\#,\$}$, starting from S: $S \to \#XS\$Y$, $S \to \lambda$, $X \to aA$, $A \to Bb$, $B \to aA$, $B \to \lambda$, $Y \to cC$, $C \to Dd$, $D \to cC$, $D \to \lambda$, where $a, b, c, d, \#, \$$ are terminals and others are non-terminals.
2. We next show that $L_{\#,\$} \notin \mathcal{L}_{reg}(\mathrm{LIN})$. Assume that $L_{\#,\$} \in \mathcal{L}_{reg}(\mathrm{LIN})$. According to [11], this means that there is a pushdown automaton A with, say, q states, using s pushdown symbols, that accepts $L_{\#,\$}$ and that satisfies that it always empties its pushdown storage completely before starting to store new symbols on it. In other words, the work of A on any input word w decomposes into phases $\phi_1, \psi_1, \phi_2, \psi_2, \dots, \phi_r, \psi_r$, where in phase ϕ_i, symbols are pushed on the pushdown, while in phase ψ_i, symbols are popped from the pushdown, and when changing between ψ_i and ϕ_{i+1}, the pushdown store is empty.
   Consider A working on a word w starting with $v = \#a^nb^m\#$. If w should belong to $L_{\#,\$}$, it must satisfy $n = m$. As n, m are arbitrarily large numbers, this can only be checked by A when using its pushdown store. According to the conditions put on A, the pushdown store must be nearly empty when finishing reading v. This means that there might be some left-over symbols, but in order to check that $m = n$, this must be a small number of symbols, say ℓ, depending on q and s, as we will see. Now, if w decomposes like $w = vv'v''$, with $v' = a^{n'}b^{m'}\#$, then a similar argument shows that, for sufficiently large n', A must use its pushdown store to check that $n' = m'$. In order to do so, A must first empty its pushdown store completely, either before or after reading the separator $\#$. Assume that A has not emptied its pushdown when reading $\#$. So, A has to continue emptying its pushdown when reading past $\#$. If $\ell > qs$, then while emptying the pushdown, reading a's from the input, some configuration must occur twice by *pigeon hole* principle. Hence, we would also accept some word w' that is obtained from w by omitting some symbols

TABLE 1. A GCID$_{SDT}(5;2,1,0;1,0,0)$ system for the language $L_{\#,\$}$ that is context-free but not in the regular closure of LIN.

Component C1		Component C5
$s_1 1.1 : (1, (\dagger_1, \#X, \lambda)_{ins}, 2)$		$s_2 5.1 : (5, (\dagger_2, \$Y, \lambda)_{ins}, 2)$
$s_2 1.1 : (1, (\lambda, \dagger_1, \lambda)_{del}, 1)$		
$s_3 1.1 : (1, (\lambda, \dagger_2, \lambda)_{del}, 1)$		
Component C2	Component C3	Component C4
$p_1 2.1 : (2, (X, p_1, \lambda)_{ins}, 3)$	$p_1 3.1 : (3, (\lambda, X, \lambda)_{del}, 2)$	$p_1 4.1 : (4, (\lambda, p_1, \lambda)_{del}, 2)$
$p_1 2.2 : (2, (p_1, aA, \lambda)_{ins}, 4)$		
$q_1 2.1 : (2, (A, q_1, \lambda)_{ins}, 3)$	$q_1 3.1 : (3, (\lambda, A, \lambda)_{del}, 2)$	$q_1 4.1 : (4, (\lambda, q_1, \lambda)_{del}, 2)$
$q_1 2.2 : (2, (q_1, Bb, \lambda)_{ins}, 4)$		
$r_1 2.1 : (2, (B, r_1, \lambda)_{ins}, 3)$	$r_1 3.1 : (3, (\lambda, B, \lambda)_{del}, 2)$	$r_1 4.1 : (4, (\lambda, r_1, \lambda)_{del}, 2)$
$r_1 2.2 : (2, (r_1, aA, \lambda)_{ins}, 4)$		
$h_1 2.1 : (2, (\lambda, B, \lambda)_{del}, 5)$		
$p_2 2.1 : (2, (Y, p_2, \lambda)_{ins}, 3)$	$p_2 3.1 : (3, (\lambda, Y, \lambda)_{del}, 2)$	$p_2 4.1 : (4, (\lambda, p_2, \lambda)_{del}, 2)$
$p_2 2.2 : (2, (p_2, cC, \lambda)_{ins}, 4)$		
$q_2 2.1 : (2, (C, q_2, \lambda)_{ins}, 3)$	$q_2 3.1 : (3, (\lambda, C, \lambda)_{del}, 2)$	$q_2 4.1 : (4, (q_2, Dd, \lambda)_{ins}, 2)$
$q_2 2.2 : (2, (\lambda, q_2, \lambda)_{del}, 4)$		
$r_2 2.1 : (2, (D, r_2, \lambda)_{ins}, 3)$	$r_2 3.1 : (3, (\lambda, D, \lambda)_{del}, 2)$	$r_2 4.1 : (4, (\lambda, r_2, \lambda)_{del}, 2)$
$r_2 2.2 : (2, (r_2, cC, \lambda)_{ins}, 4)$		
$h_2 2.1 : (2, (\lambda, D, \lambda)_{del}, 1)$		

from the second block of a's (and leaving the rest unchanged), contradicting the structure of the language $L_{\#,\$}$. We arrive at a similar contradiction assuming that A has already emptied its pushdown before reading $\#$. A similar argument proves that, upon reading the first part $\#a^{n_1}b^{n_1}\#a^{n_2}b^{n_2}\ldots\#a^{n_p}b^{n_p}\$$ of w, the pushdown store must be emptied within qs symbols to the left or right of the separators $\#, \$$. Notice that we also know that A has worked in phases $\phi_1, \psi_1, \phi_2, \psi_2, \ldots, \phi_p, \psi_p$ when processing this prefix. This also means that A cannot use its pushdown storage to memorize the number p of occurrences of the $a^n b^n$-pattern. Hence, again by pigeon hole, if p is bigger than $3q+1$, when reading the suffix $\$c^{k_1}d^{k_1}\$c^{k_2}d^{k_2}\ldots\$c^{k_p}d^{k_p}$ – with phases $\phi_{p+1}, \psi_{p+1}, \ldots, \phi_{2p}, \psi_{2p}$ as A must also check this $c^k d^k$-patterns – there must be a situation where in the sequence of states $s_p, \ldots, s_{2p}$ encountered when leaving phase $\psi_p, \ldots, \psi_{2p}$, respectively, configurations repeat. (Recall that a configuration is determined by the current input symbol, the state and the current pushdown symbol; as the pushdown store is empty in the described configurations, and as we only have three different (reasonable) input symbols, the bound follows.) This would allow to also accept words violating the structure of $L_{\#,\$}$, assuming that w is accepted. This concludes the proof of our claim.

3. We now construct a GCID system $\varPi$ of size $(5;2,1,0;1,0,0)$ to describe $L_{\#,\$}$ as follows:

$$\varPi = (5, \{\dagger_1, \dagger_2, \#, \$, X, A, B, Y, C, D, a, b, c, d\}, \{a, b, c, d, \#, \$\}, \{\dagger_1\dagger_2\}, H, 1, 1, R),$$

where the rule labels of H and the rules of R are given in Table 1. We now explain how the rules work in order to prove that $L(\varPi) = L_{\#,\$}$.

Let $L_1 = \{a^n b^n \mid n \geq 1\}$ and $L_2 = \{c^m d^m \mid m \geq 1\}$ denote two basic constituents of $L_{\#,\$}$. Starting from the axiom $\dagger_1\dagger_2$ in $C1$, we can apply $s_1 1.1$, $s_2 1.1$ or $s_3 1.1$. If we apply $s_2 1.1$ and $s_3 1.1$ together, then $\lambda \in L_{\#,\$}$ is produced. Actually, when starting with $s_2 1.1$, we have to apply $s_3 1.1$ next in order to produce some terminal string. When starting with $s_3 1.1$ instead, we might be tempted to apply $s_1 1.1$ next. It would then be possible to continue a derivation as intended when immediately starting with $s_1 1.1$, as it is explained below. However, at some point of time, the string will be stuck at $C5$ (which is not a final component) as there was no $\dagger_2$ to apply the rule $s_2 5.1$. We now move on to discuss the non-trivial (intended) rule application $s_1 1.1$ that works in phases, starting in phase $s1$.

Phase *s*1: on starting to apply $s_1 1.1$, we insert $\#X$ on the right of $\dagger_1$ and the string $\dagger_1\#X\dagger_2$ is sent to $C2$. In $C2$, using X (which acts as a start symbol for a grammar part that describes L_1), we can produce a string of L_1 in three phases $p1, q1, r1$ as follows:

Phase *p*1: Applying the rules $p_1 2.1, p_1 3.1, p_1 2.2, p_1 4.1$ in order, we have the following derivation, starting in configuration $(\dagger_1\#X\dagger_2)_2$ and applying $p_1 2.1$:

$$(\dagger_1\#Xp_1\dagger_2)_3 \Rightarrow_{p_1 3.1} (\dagger_1\#p_1\dagger_2)_2 \Rightarrow_{p_1 2.2} (\dagger_1\#p_1 aA\dagger_2)_4 \Rightarrow_{p_1 4.1} (\dagger_1\#aA\dagger_2)_2.$$

At this point, the only applicable rule in $C2$ is $q_1 2.1$, which initiates Phase $q1$.

Phase *q*1: applying the rules $q_1 2.1, q_1 3.1, q_1 2.2, q_1 4.1$ in order we have the following derivation, starting in configuration $(\dagger_1\#aA\dagger_2)_2$ and applying $q_1 2.1$:

$$(\dagger_1\#aAq_1\dagger_2)_3 \Rightarrow_{q_1 3.1} (\dagger_1\#aq_1\dagger_2)_2 \Rightarrow_{q_1 2.2} (\dagger_1\#aq_1 Bb\dagger_2)_4 \Rightarrow_{q_1 4.1} (\dagger_1\#aBb\dagger_2)_2.$$

At this point, the applicable rules in $C2$ are $r_1 2.1$ and $h_1 2.1$, which initiates Phase $r1$ and Phase $h1$, respectively.

Phase *r*1: applying the rules $r_1 2.1, r_1 3.1, r_1 2.2, r_1 4.1$ in order, we have the following derivation, starting in configuration $(\dagger_1\#aBb\dagger_2)_2$ and applying $r_1 2.1$:

$$(\dagger_1\#aBr_1 b\dagger_2)_3 \Rightarrow_{r_1 3.1} (\dagger_1\#ar_1 b\dagger_2)_2 \Rightarrow_{r_1 2.2} (\dagger_1\#ar_1 aAb\dagger_2)_4 \Rightarrow_{r_1 4.1} (\dagger_1\#aaAb\dagger_2)_2\,,$$

which is very similar to phase $p1$. Continuing to apply rules in Phases $q1$ and $r1$ alternatingly for the desired, say n_1, number of times, we obtain the configuration $(\dagger_1\#a^{n_1}Bb^{n_1}\dagger_2)_2$. To terminate this process, we move to Phase $h1$.

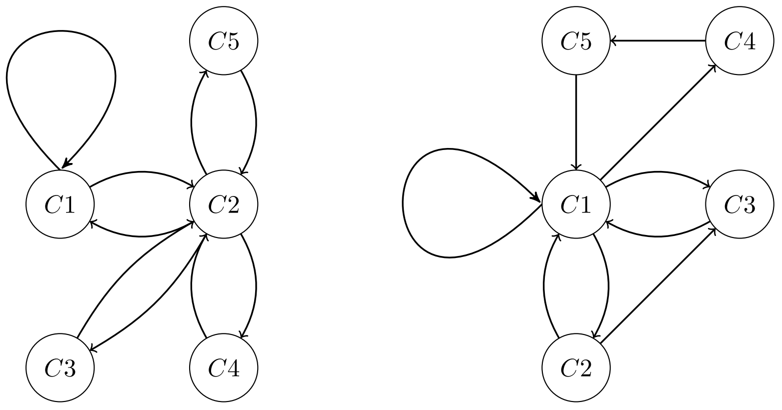
Phase *h*1: on applying the rule $h_1 2.1$ in $C2$, the nonterminal B is deleted and the resultant string $(\dagger_1\#a^{n_1}b^{n_1}\dagger_2)$is moved to $C5$. In $C5$, $\$Y$ is introduced after $\dagger_2$ by $s_2 5.1$.
Analogously to phases $p1, q1, r1$ (and again $q1$), $h1$, Y generates a string $c^{k_1}d^{k_1} \in L_2$ in $C2$ by the phases $p2, q2, r2$ (and again $q2$), $h2$. Hence, we will be having a configuration of the form $(\dagger_1\#a^{n_1}b^{n_1}\,\dagger_2\,\$c^{k_1}d^{k_1})_1$.
We highlight a main difference in the application of $h_1 2.1$ (in phase $h1$) and $h_2 2.1$. In the former case, the string is taken to $C5$ while in the latter case, the string is taken to $C1$.
Now the whole process is repeated again: applying the rules in phases $s1, p1, q1$, $r1, h1$, $s2, p2, q2$, $r2, h2$, in order, for desired number (any p) number of times, we get a configuration of the form

$$(\dagger_1\#a^{n_1}b^{n_1}\#a^{n_2}b^{n_2}\ldots\#a^{n_p}b^{n_p}\,\dagger_2\,\$c^{k_1}d^{k_1}\$c^{k_2}d^{k_2}\ldots\$c^{k_p}d^{k_p})_1\,.$$

To terminate the derivation, we apply the rules $s_2 1.1$ and $s_3 1.1$, which will delete the markers $\dagger_1$ and $\dagger_2$, respectively. With these details, one can see that $L(\mathit{\Pi}) = L_{\#,\$}$.

4. The claimed tree structure of the given GCID system can be seen by the illustration given in Figure 1a. □

Example 3.2. Consider the language $L' = \{a^n b^i c^i b^k c^k d^n \mid n, i, k \geq 0\}$. We now show that $L' \in (\mathrm{CF} \cap \mathrm{GCID}_{SD}(5; 1, 1, 0; 1, 0, 0)) \setminus \mathcal{L}_{reg}(\mathrm{LIN})$.



(a) Tree-structured control graph corresponding to rules in Table 1

(b) Control graph corresponding to rules in Table 2, with cycle $C2 \to C3 \to C1$.

FIGURE 1. Control graphs of the GCID systems that describe the context-free languages of Example 3.1 and Table 2.

1. Clearly, L' is context-free.
2. Observe that $L' \notin \text{LIN}$, which can be shown by applying the pumping lemma for linear languages as stated in [10]. If $L' \in \mathcal{L}_{reg}(\text{LIN})$, then L' would be described by substituting linear languages into a regular expression E. If E would contain a single symbol, then L' would be linear, which is not the case. If $E = F^*$ for some regular expression F, then (assuming, w.l.o.g., that a non-regular language is substituted for F) L' would equal L^* for some non-regular language L. This contradicts the structure of L'. If $E = E_1E_2$ (catenation of two regular expressions E_1 and E_2), then (excluding again trivial cases when E_1 or E_2 are substituted by regular languages) there is no way to synchronize the number of a's and d's in any string, so that L' cannot be described, as sufficiently long strings a^nwd^n (where n is bigger than the number of nonterminals of any linear language that is substituted, assuming only rules of the form $A \to bB$ or $A \to Bb$ in these grammars) can be changed to generating some word not in L', as due to the lengths of the strings, in particular the beginning grammar (corresponding to the first letter in E_1) will run into a loop by pigeon hole; this loop can be omitted (zero-pumped) to create such an unintended word. Finally, if $E = E_1 \cup E_2$, then the arguments presented above can be applied to infinite sublanguages of L'.
3. The language L' can be generated by the GCID system

$$\Pi' = (5, \{S, S_2, S_3', S_3, a, b, c, d\}, \{a, b, c, d\}, \{S\}, H, 1, 1, R)\,,$$

where the rules from R are presented in Table 2. The size of Π' is $(5; 1, 1, 0; 1, 0, 0)$. We note that $\lambda \in L'$ is derived by the following sequence of derivations.

$$\begin{aligned}&(S)_1 \Rightarrow_{h_1 1.1} (SS_2)_4 \Rightarrow_{s_1 4.1} (S_2)_5 \Rightarrow_{s_1 5.1} (S_2S_3')_1 \\ &\Rightarrow_{h_2 1.1} (S_3')_4 \Rightarrow_{s_2 4.1} (S_3'S_3)_1 \Rightarrow_{h_3 1.1} (S_3')_5 \Rightarrow_{s_3 5.1} (\lambda)_1\,.\end{aligned}$$

We now explain why $L(\Pi') = L'$. Starting from the axiom S, we can only apply $h_1 1.1$ (as discussed above), or $p_1 1.1$.

Phase p_1, starting with $p_1 1.1$. Component $C3$ only contains deletion rules. In configuration $(Sp_1)_3$, only $p_1 3.1$ and $p_1 3.2$ are applicable. Applying $p_1 3.2$ brings us back to the original configuration $(S)_1$, so that, to make progress, $p_1 3.1$ has to be applied, leading to $(p_1)_1$. Back in $C1$, only $p_1 1.2$ is applicable, leading us to $(p_1p_1')_2$. Rule $p_1 2.2$ brings us back to $(p_1)_1$, so that finally $p_1 2.1$ has to be applied, yielding the configuration

Table 2. A GCID$_{SD}(5;1,1,0;1,0,0)$ system for the language L' that is context-free but not in the regular closure of LIN.

Component C1	Component C2	Component C3
$p_1 1.1 : (1,(S,p_1,\lambda)_{ins},3)$	$p_1 2.1 : (2,(p_1,a,\lambda)_{ins},3)$	$p_1 3.1 : (3,(\lambda,S,\lambda)_{del},1)$
$p_1 1.2 : (1,(p_1,p_1',\lambda)_{ins},2)$	$p_1 2.2 : (2,(\lambda,p_1',\lambda)_{del},1)$	$p_1 3.2 : (3,(\lambda,p_1,\lambda)_{del},1)$
$p_1 1.3 : (1,(p_1',A,\lambda)_{ins},2)$		
$q_1 1.1 : (1,(A,q_1,\lambda)_{ins},3)$	$q_1 2.1 : (2,(q_1,q_1',\lambda)_{ins},3)$	$q_1 3.1 : (3,(\lambda,A,\lambda)_{del},1)$
$q_1 1.2 : (1,(q_1,d,\lambda)_{ins},2)$	$q_1 2.2 : (2,(\lambda,q_1',\lambda)_{del},1)$	$q_1 3.2 : (3,(\lambda,q_1,\lambda)_{del},1)$
$q_1 1.3 : (1,(q_1',S,\lambda)_{ins},2)$		
$h_1 1.1 : (1,(S,S_2,\lambda)_{ins},4)$		
$p_2 1.1 : (1,(S_2,p_2,\lambda)_{ins},3)$	$p_2 2.1 : (2,(p_2,b,\lambda)_{ins},3)$	$p_2 3.1 : (3,(\lambda,S_2,\lambda)_{del},1)$
$p_2 1.2 : (1,(p_2,p_2',\lambda)_{ins},2)$	$p_2 2.2 : (2,(\lambda,p_2',\lambda)_{del},1)$	$p_2 3.2 : (3,(\lambda,p_2,\lambda)_{del},1)$
$p_2 1.3 : (1,(p_2',B,\lambda)_{ins},2)$		
$q_2 1.1 : (1,(B,q_2,\lambda)_{ins},3)$	$q_2 2.1 : (2,(q_2,q_2',\lambda)_{ins},3)$	$q_2 3.1 : (3,(\lambda,B,\lambda)_{del},1)$
$q_2 1.2 : (1,(q_2,c,\lambda)_{ins},2)$	$q_2 2.2 : (2,(\lambda,q_2',\lambda)_{del},1)$	$q_2 3.2 : (3,(\lambda,q_2,\lambda)_{del},1)$
$q_2 1.3 : (1,(q_2',S_2,\lambda)_{ins},2)$		
$h_2 1.1 : (1,(\lambda,S_2,\lambda)_{del},4)$		
$p_3 1.1 : (1,(S_3,p_3,\lambda)_{ins},3)$	$p_3 2.1 : (2,(p_3,b,\lambda)_{ins},3)$	$p_3 3.1 : (3,(\lambda,S_3,\lambda)_{del},1)$
$p_3 1.2 : (1,(p_3,p_3',\lambda)_{ins},2)$	$p_3 2.2 : (2,(\lambda,p_3',\lambda)_{del},1)$	$p_3 3.2 : (3,(\lambda,p_3,\lambda)_{del},1)$
$p_3 1.3 : (1,(p_3',C,\lambda)_{ins},2)$		
$q_3 1.1 : (1,(C,q_3,\lambda)_{ins},3)$	$q_3 2.1 : (2,(q_3,q_3',\lambda)_{ins},3)$	$q_3 3.1 : (3,(\lambda,C,\lambda)_{del},1)$
$q_3 1.2 : (1,(q_3,c,\lambda)_{ins},2)$	$q_3 2.2 : (2,(\lambda,q_3',\lambda)_{del},1)$	$q_3 3.2 : (3,(\lambda,q_3,\lambda)_{del},1)$
$q_3 1.3 : (1,(q_3',S_3,\lambda)_{ins},2)$		
$h_3 1.1 : (1,(\lambda,S_3,\lambda)_{del},1)$		

Component C4	Component C5
$s_1 4.1 : (4,(\lambda,S,\lambda)_{del},5)$	$s_1 5.1 : (5,(S_2,S_3',\lambda)_{ins},1)$
$s_2 4.1 : (4,(S_3',S_3,\lambda)_{ins},5)$	$s_2 5.1 : (5,(\lambda,S_3',\lambda)_{del},1)$

$(p_1ap_1')_3$. Now, the application of $p_1 3.2$ is enforced, yielding $(ap_1')_1$. Now, only $p_1 1.3$ is applicable, leading to $(ap_1'A)_2$ and further to $(aA)_1$. This enables us to begin with the next phase $q1$.

Phase $\boldsymbol{q}_1$, starting with $\boldsymbol{q}_1$1.1, as this is the only applicable rule. Avoiding useless loops when applying $q_1 3.2$, we are led to $(aAq_1)_3 \Rightarrow (aq_1)_1$. Now, the following derivation is enforced, starting with $(aq_1)_1 \Rightarrow_{q_1 1.2} (aq_1d)_2$:

$$(aq_1d)_2 \Rightarrow_{q_1 2.1} (aq_1q_1'd)_3 \Rightarrow_{q_1 3.2} (aq_1'd)_1 \Rightarrow_{q_1 1.3} (aq_1'Sd)_2 \Rightarrow_{q_1 2.2} (aSd)_1 \,.$$

Now, we have two alternatives: either to restart Phase p_1, followed by Phase q_1, leading (after n repetitions) to $(a^nSd^n)_1$, or to change to Phase p_2 using rule $h_1 1.1$, followed by applying two more rules as an interludium. Hence, we get:

$$(a^nSd^n)_1 \Rightarrow_{h_1 1.1} (a^nSS_2d^n)_4 \Rightarrow_{s_1 4.1} (a^nS_2d^n)_5 \Rightarrow_{s_1 5.1} (a^nS_2S_3'd^n)_1 \,.$$

Instead of taking the route to Phase p_2, we could also skip this phase by directly applying $h_2 1.1$, finally leading to the possibility of having $i = 0$ as follows.

Phase $\boldsymbol{p}_2$, starting with $\boldsymbol{p}_2$1.1. Omitting loops, we apply the following derivation:

$$\begin{aligned}(a^n S_2 S_3' d^n)_1 &\Rightarrow_{p_2 1.1} (a^n S_2 p_2 S_3' d^n)_3 \Rightarrow_{p_2 3.1} (a^n p_2 S_3' d^n)_1 \Rightarrow_{p_2 1.2} (a^n p_2 p_2' S_3' d^n)_2\\ &\Rightarrow_{p_2 2.1} (a^n p_2 b p_2' S_3' d^n)_3 \Rightarrow_{p_2 3.2} (a^n b p_2' S_3' d^n)_1\\ &\Rightarrow_{p_2 1.3} (a^n b p_2' B S_3' d^n)_2 \Rightarrow_{p_2 2.2} (a^n b B S_3' d^n)_1 .\end{aligned}$$

Observe how the sequence of rule applications is enforced once it was started; also, there is no other way how to deal with configuration $(a^n S_2 S_3' d^n)_1$. Similar remarks are in order for the phases discussed in the following. Now, the next phase can begin.

Phase $\boldsymbol{q}_2$, starting with $\boldsymbol{q}_2$1.1. Omitting loops, we apply the following derivation:

$$\begin{aligned}(a^n b B S_3' d^n)_1 &\Rightarrow_{q_2 1.1} (a^n b B q_2 S_3' d^n)_3 \Rightarrow_{q_2 3.1} (a^n b q_2 S_3' d^n)_1 \Rightarrow_{q_2 1.2} (a^n b q_2 c S_3' d^n)_2\\ &\Rightarrow_{q_2 2.1} (a^n b q_2 q_2' c S_3' d^n)_3 \Rightarrow_{q_2 3.2} (a^n b q_2' c S_3' d^n)_1\\ &\Rightarrow_{q_2 1.3} (a^n b q_2' S_2 c S_3' d^n)_2 \Rightarrow_{q_2 2.2} (a^n b S_2 c S_3' d^n)_1\end{aligned}$$

Now, one can repeat Phase p_2 and q_2 i times altogether to arrive at $(a^n b^i S_2 c^i S_3' d^n)_1$. Finally, $h_2 1.1$ has to be applied, leading to a second bridge:

$$(a^n b^i S_2 c^i S_3' d^n)_1 \Rightarrow_{h_2 1.1} (a^n b^i c^i S_3' d^n)_4 \Rightarrow_{s_2 4.1} (a^n b^i c^i S_3' S_3 d^n)_5 \Rightarrow_{s_2 5.1} (a^n b^i c^i S_3 d^n)_1$$

Instead of travelling through Phase p_3, we could also skip this phase by applying $h_3 1.1$ directly, leading finally to the possibility of having $k = 0$ in the following case analysis.

Phase $\boldsymbol{p}_3$, starting with $\boldsymbol{p}_3$1.1. Omitting loops, we apply the following derivation:

$$\begin{aligned}(a^n b^i c^i S_3 d^n)_1 &\Rightarrow_{p_3 1.1} (a^n b^i c^i S_3 p_3 d^n)_3 \Rightarrow_{p_3 3.1} (a^n b^i c^i p_3 d^n)_1\\ &\Rightarrow_{p_3 1.2} (a^n b^i c^i p_3 p_3' d^n)_2 \Rightarrow_{p_3 2.1} (a^n b^i c^i p_3 b p_3' d^n)_3\\ &\Rightarrow_{p_3 3.2} (a^n b^i c^i b p_3' d^n)_1 \Rightarrow_{p_3 1.3} (a^n b^i c^i b p_3' C d^n)_2\\ &\Rightarrow_{p_3 2.2} (a^n b^i c^i b C d^n)_1\end{aligned}$$

Now, the next phase can begin.

Phase $\boldsymbol{q}_3$, starting with $\boldsymbol{q}_3$1.1. Omitting loops, we apply the following derivation:

$$\begin{aligned}(a^n b^i c^i b C d^n)_1 &\Rightarrow_{q_3 1.1} (a^n b^i c^i b C q_3 d^n)_3 \Rightarrow_{q_3 3.1} (a^n b^i c^i b q_3 d^n)_1\\ &\Rightarrow_{q_3 1.2} (a^n b^i c^i b q_3 c d^n)_2 \Rightarrow_{q_3 2.1} (a^n b^i c^i b q_3 q_3' c d^n)_3\\ &\Rightarrow_{q_3 3.2} (a^n b^i c^i b q_3' c d^n)_1 \Rightarrow_{q_3 1.3} (a^n b^i c^i b q_3' S_3 c d^n)_2\\ &\Rightarrow_{q_3 2.2} (a^n b^i c^i b S_3 c d^n)_1\end{aligned}$$

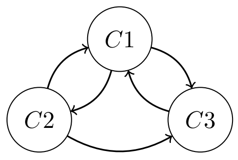
Now, one can repeat Phase p_3 and q_3, k times altogether, to arrive at the configuration $(a^n b^i c^i b^k S_3 c^k d^n)_1$. Finally, on applying $h_3 1.1$, which deletes S_3, the derivation stops at $C1$, showing $a^n b^i c^i b^k c^k d^n \in L(\Pi')$.

From the argument we gave above, it is clear that $L' = L(\Pi')$, as no malicious derivations are possible. □

Remark 3.3. The GCID system given in the last example is not tree-structured, as witnessed by Figure 1b. We do not know if this language can be described by a tree-structured system of size $(5; 1, 1, 0; 1, 0, 0)$. □

Component C1	**Component C2**	**Component C3**
$p1.1 : (1, (X, p, \lambda)_{ins}, 3)$	$p2.1 : (2, (p, a, \lambda)_{ins}, 3)$	$p3.1 : (3, (\lambda, X, \lambda)_{del}, 1)$
$p1.2 : (1, (p, p', \lambda)_{ins}, 2)$	$p2.2 : (2, (\lambda, p', \lambda)_{del}, 1)$	$p3.2 : (3, (\lambda, p, \lambda)_{del}, 1)$
$p1.3 : (1, (p', Y, \lambda)_{ins}, 2)$		
$q1.1 : (1, (X, q, \lambda)_{ins}, 3)$	$q2.1 : (2, (q, q', \lambda)_{ins}, 3)$	$q3.1 : (3, (\lambda, X, \lambda)_{del}, 1)$
$q1.2 : (1, (q, a, \lambda)_{ins}, 2)$	$q2.2 : (2, (\lambda, q', \lambda)_{del}, 1)$	$q3.2 : (3, (\lambda, q, \lambda)_{del}, 1)$
$q1.3 : (1, (q', Y, \lambda)_{ins}, 2)$		
$h1.1 : (1, (\lambda, X, \lambda)_{del}, 1)$		

(a) $\mathrm{GCID}_{SD}(3; 1, 1, 0; 1, 0, 0)$ describing LIN



(b) Control graph

FIGURE 2. How simple-deleting GCID systems with non-tree structure and of size $(3; 1, 1, 0; 1, 0, 0)$ could describe all the linear languages.

4. Describing super-linear languages by GCID systems

In order to simplify the presentation and derivation of some of our main results in the paper, the following observations are helpful.

Proposition 4.1 ([5]). *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) $\mathrm{GCID}(k; n, i', i''; m, j', j'') = [\mathrm{GCID}(k; n, i'', i'; m, j'', j')]^R$.
(2) $\mathcal{L} \subseteq \mathrm{GCID}(k; n, i', i''; m, j', j'')$ *iff* $\mathcal{L} \subseteq \mathrm{GCID}(k; n, i'', i'; m, j'', j')$.

In the following, we are building on earlier results concerning simulations of linear grammars. We summarize these results in the following proposition.

Proposition 4.2 ([1, 5]). *For any $\beta \in \{(3; 1, 1, 0; 1, 0, 0), (3; 1, 0, 1; 1, 0, 0), (3; 2, 1, 0; 1, 0, 0), (3; 2, 0, 1; 1, 0, 0)\}$, simple-deleting GCID systems of any size β can describe all linear languages. Moreover, for $\beta \in \{(3; 2, 1, 0; 1, 0, 0), (3; 2, 0, 1; 1, 0, 0)\}$, this is even possible with the control graph always being a tree.*

GCID systems of size β (as mentioned in the above proposition) are not known to characterize RE. 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.

To better explain how such a simulation might work, we show in Figures 2a and 3a how rules in the linear normal form $p : X \to aY$, $q : X \to Ya$ and $h : X \to \lambda$ could be simulated using rules of the systems $\mathrm{GCID}_{SD}(3; 1, 1, 0; 1, 0, 0)$ and $\mathrm{GCID}_{SDT}(3; 2, 1, 0; 1, 0, 0)$, respectively. For instance, a sentential form $\alpha X \beta$ in $C1$ would develop as follows when simulating an application of q rule from the table shown in Figure 2a.

$$\begin{aligned}(\alpha X \beta)_1 &\Rightarrow (\alpha X q \beta)_3 \Rightarrow (\alpha q \beta)_1 \Rightarrow (\alpha q a \beta)_2 \Rightarrow (\alpha q q' a \beta)_3 \\ &\Rightarrow (\alpha q' a \beta)_1 \Rightarrow (\alpha q' Y a \beta)_2 \Rightarrow (\alpha Y a \beta)_1\end{aligned}$$

It is also clear from the description of the working rules stated in Figure 2a that the underling control structure is not a tree; see Figure 2b. In fact, it is an open question if linear languages that can be obtained by $\mathrm{GCID}_{SDT}(3; 1, 1, 0; 1, 0, 0)$.

On the other hand, a sentential form $\alpha X \beta$ in $C1$ would develop as follows when simulating q rule from the table shown in Figure 3a.

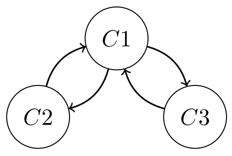
$$(\alpha X \beta)_1 \Rightarrow (\alpha X q \beta)_2 \Rightarrow (\alpha q \beta)_1 \Rightarrow (\alpha q a Y \beta)_3 \Rightarrow (\alpha a Y \beta)_1$$

Clearly, the underlying (undirected simple graph) control structure of the simulating rules presented in Figure 3a is a tree; see Figure 3b.

If we add two more components to the GCID systems stated above, then these (extended) GCID systems are shown to describe $\mathcal{L}_\circ(\mathrm{LIN})$ and $\mathcal{L}_*(\mathrm{LIN})$ in [1, 6].

Component C1	Component C2	Component C3
$p1.1 : (1, (X, p, \lambda)_{ins}, 2)$ $p1.2 : (1, (p, aY, \lambda)_{ins}, 3)$	$p2.1 : (2, (\lambda, X, \lambda)_{del}, 1)$	$p3.1 : (3, (\lambda, p, \lambda)_{del}, 1)$
$q1.1 : (1, (X, q, \lambda)_{ins}, 2)$ $q1.2 : (1, (q, Ya, \lambda)_{ins}, 3)$	$q2.1 : (2, (\lambda, X, \lambda)_{del}, 1)$	$q3.1 : (3, (\lambda, q, \lambda)_{del}, 1)$
$h1.1 : (1, (\lambda, X, \lambda)_{del}, 1)$		

(a) GCID$_{SDT}(3; 2, 1, 0; 1, 0, 0)$ describing LIN



(b) Control graph

FIGURE 3. How simple-deleting GCID systems with tree structure and of size $(3; 2, 1, 0; 1, 0, 0)$ could describe linear languages.

4.1. Describing the regular closure of the linear languages

Initially, our main objective was to find how much beyond LIN, GCID systems (of the four sizes stated in Prop. 4.2) can lead us. However, we then succeeded to provide a general result by showing the following: If there exist graph-controlled ins-del systems of ID size $(n, i', i''; m, j'j'')$ describing LIN, then these constructions can be extended to graph-controlled ins-del systems of the same ID size at the expense of two more components to describe $\mathcal{L}_{reg}(\mathrm{LIN})$.[2]

Theorem 4.3. *For all integers $t, n, m \geq 1$ and $i', i'', j', j'' \geq 0$ with $i' + i'' \geq 1$ and for $X \in \{SD, SDT\}$, if $\mathrm{LIN} \subseteq \mathrm{GCID}_X(t; n, i', i''; m, j', j'')$ was shown by a simple-deleting simulation, then $\mathcal{L}_{reg}(\mathrm{LIN}) \subseteq \mathrm{GCID}_X(t + 2; n, i', i''; m, j', j'')$ follows.*

Proof. Let $L \in \mathcal{L}_{reg}(\mathrm{LIN})$ for some $L \subseteq T^*$. By Proposition 2.4 (and the explanation that follows it), we can assume that L is described by a context-free grammar $G = (N, T, S, P)$ that basically consists of a right-linear grammar $G_R = (N_0, N'', S, P_0)$ and linear grammars $G_i = (N_i, T, S_i, P_i)$ for $i \in [1 \dots k]$. For technical reasons that should become clear soon, we rather consider $G'_i = (N'_i, T, S_i, P'_i)$, where $N'_i = N_i \cup \{\langle S_i, A\rangle \mid A \in N_0\}$ and P'_i contains, besides all rules from P_i, rules of the form $\langle S_i, A\rangle \to w$ whenever $S_i \to w \in P_i$ for some $w \in (N_i \cup T)^*$. This means that $L(G'_i) = L(G_i)$ (as the new nonterminals will never be used in terminating derivations) and $L((N'_i, T, \langle S_i, A\rangle, P'_i)) = L(G_i)$ for any $A \in N_0$.

Since $\mathrm{LIN} \subseteq \mathrm{GCID}_{SD(T)}(t; n, i', i''; m, j', j'')$, 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 graph-controlled ins-del system Π for G as follows[3]:

$$\Pi = \left(t + 2, V, T, \{S_i A' \mid S \to S_i A \in P\}, H \cup H', 1, 1, (R \setminus \hat{R}) \cup R' \cup R''\right), \text{ where}$$

- $V = \left(\bigcup_{i=1}^{k} (V_i \cup \{\langle S_i, A\rangle \mid A \in N_0\})\right) \cup N_0 \cup \{A' \mid A \in N_0\}$;
- $H' \subset \bigcup_{p \in P_0} \{r_p(t + 1).1, r_p(t + 1).2, r_p(t + 2).1\}$;
- $H = \bigcup_{i=1}^{k} H_i$; $R = \bigcup_{i=1}^{k} R_i$; $\hat{R} = \bigcup_{i=1}^{k} \{h_i 1.1 : (1, (\lambda, X_i, \lambda)_{del}, 1)\}$;
- $R' = \bigcup_{i=0}^{k} \{h_i 1.1 : (1, (\lambda, X_i, \lambda)_{del}, t + 1) \mid X_i \to \lambda \in P_i\}$;

[2]Notice that the simulations presented below differ from the ones presented in the conference version, because we found the normal form for $\mathcal{L}_{reg}(\mathrm{LIN})$ employed in this paper easier to explain.

[3]There is one subtlety with the case when $\lambda \in L(G)$: in that case, λ should be added as an axiom of Π.

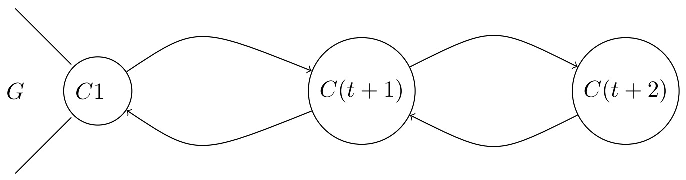


FIGURE 4. Control graph underlying the simulation of $\mathcal{L}_{reg}(\text{LIN})$ as in Theorem 4.3.

- R'' is the set with the following rules: for each $p : A \to S_iB \in P_0$, we add:

$$\begin{aligned} r_p(t+1).1 &\quad : (t+1, (A', \langle S_i, B\rangle, \lambda)_{ins}, t+2), \\ r_p(t+1).2 &\quad : (t+1, (\langle S_i, B\rangle, B', \lambda)_{ins}, 1) \\ r_p(t+2).1 &\quad : (t+2, (\lambda, A', \lambda)_{del}, t+1); \\ \text{Further, } r_q(t+1).1 &\quad : (t+1, (\lambda, A', \lambda)_{del}, 1) \text{ for } q : A \to \lambda \in P_0. \end{aligned}$$

Since $L_i = L(G_i)$ is generated by Π_i, for $1 \le i \le k$, the linear rules of Π_i are simulated by rules of R_i in the first t components. Also, there is no interference between rules of different systems Π_i and Π_j, since $V_i \cap V_j = T$ if $1 \le i < j \le k$. Moreover, the simulation could start with any symbol $\langle S_i, A\rangle$ instead of S_i itself due to the rules $\langle S_i, A\rangle \to w$ added when moving from G_i to G_i'.

We start with the axiom S_iA' for some rule $S \to S_iA$ of G_R. Now, a string $w_1 \in L(G_i)$ is produced by simulating G_i' in the first t components of the system Π. In this simulation of G_i', finally the terminating rule of L_i, namely $h_i.1.1$, takes the string to component $t+1$. There, two things might happen: either, there is a rule $q : A \to \lambda$ in G_R that we decide to simulate, then we apply $r_q(t+1).1$ and move the terminal string w_1 back to $C1$; or, we choose to simulate a rule $p : A \to S_jB$ from G_R. Then, we see the following derivation:

$$(w_1A')_{t+1} \Rightarrow (w_1A'\langle S_j, B\rangle)_{t+2} \Rightarrow (w_1\langle S_j, B\rangle)_{t+1} \Rightarrow (w_1\langle S_j, B\rangle B')_1$$

As $\langle S_j, B\rangle$ can play the same role as S_j in G_j', we can now produce a terminal word $w_2 \in L(G_j)$ by simulating G_j' within the first t components etc. By our description, it should be clear that all terminal words of G can be derived by using Π.

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_iA' or $\langle S_i, A\rangle A'$ for some $A \in N_0$ in the beginning, and by some X_ivA' 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 A' into $C(t+1)$. Now, apart from the special case when A' is going to be (correctly) deleted, some string is moved back to $C1$ that satisfies the conditions expressed as the beginning of a linear phase. This shows that every terminal string that can be generated by Π is also contained in $L(G)$.

The case when $i' = 0$ and $i'' \ge 1$ follows from Propositions 2.3 and 4.1. □

Remark 4.4. In Theorem 4.3, suppose G is the underlying control graph for the simulating rules of LIN $\subseteq$ GCID$(t; n, i', i''; m, j', j'')$. Then the underlying control graph of $\mathcal{L}_{reg}(\text{LIN}) \subseteq \text{GCID}(t; n, i', i''; m, j', j'')$ is as shown in Figure 4.

Table 3 shows how the construction of the previous proof works for the concrete case of $L_1 = \{a^nb^n \mid n \ge 0\}$ and $L_2 = \{c^md^m \mid m \ge 0\} \in \text{LIN}$. If the reader tries to generate, say, *abaabbcdcd*, (s)he will understand how the phases work.

Corollary 4.5. *Combining the results of Proposition 4.2 and Theorem 4.3, we have:*

- $\mathcal{L}_{reg}(\text{LIN}) \subsetneq \text{GCID}_{SDT}(5; 2, 1, 0; 1, 0, 0) \cap \text{GCID}_{SDT}(5; 2, 0, 1; 1, 0, 0)$.
- $\mathcal{L}_{reg}(\text{LIN} \subsetneq \text{GCID}_{SD}(5; 1, 1, 0; 1, 0, 0) \cap \text{GCID}_{SD}(5; 1, 0, 1; 1, 0, 0)$.

TABLE 3. GCID$(5;2,1,0;1,0,0)$ describing languages of $\{L_1^* L_2^* : L_1, L_2 \in \text{LIN}\} \subseteq \mathcal{L}_{reg}(\text{LIN})$ according to Theorem 4.3.

Phase	Component C1	Component C2	Component C3
$p1$	$p_1 1.1 : (1, (S_1, p_1, \lambda)_{ins}, 2)$ $p_1 1.2 : (1, (p_1, aB, \lambda)_{ins}, 3)$	$p_1 2.1 : (2, (\lambda, S_1, \lambda)_{del}, 1)$	$p_1 3.1 : (3, (\lambda, p_1, \lambda)_{del}, 1)$
$q1$	$q_1 1.1 : (1, (B, q_1, \lambda)_{ins}, 2)$ $q_1 1.2 : (1, (q_1, S_1 b, \lambda)_{ins}, 3)$	$q_1 2.1 : (2, (\lambda, B, \lambda)_{del}, 1)$	$q_1 3.1 : (3, (\lambda, q_1, \lambda)_{del}, 1)$
$p2$	$p_2 1.1 : (1, (S_2, p_2, \lambda)_{ins}, 2)$ $p_2 1.2 : (1, (p_2, cD, \lambda)_{ins}, 3)$	$p_2 2.1 : (2, (\lambda, S_2, \lambda)_{del}, 1)$	$p_2 3.1 : (3, (\lambda, p_2, \lambda)_{del}, 1)$
$q2$	$q_2 1.1 : (1, (D, q_2, \lambda)_{ins}, 2)$ $q_2 1.2 : (1, (q_2, S_2 d, \lambda)_{ins}, 3)$	$q_2 2.1 : (2, (\lambda, D, \lambda)_{del}, 1)$	$q_2 3.1 : (3, (\lambda, q_2, \lambda)_{del}, 1)$
$h1$	$h_1 1.1 : (1, (\lambda, S_1, \lambda)_{del}, 4)$		
$h2$	$h_2 1.1 : (1, (\lambda, S_2, \lambda)_{del}, 4)$		
Phase	Component C4	Component C5	
$r1$	$r_1 4.1 : (4, (S', \langle S_1, S\rangle, \lambda)_{ins}, 5)$ $r_1 4.2 : (4, (\langle S_1, S\rangle, S', \lambda)_{ins}, 1)$	$r_1 5.1 : (5, (\lambda, S', \lambda)_{del}, 4)$	
$r2$	$r_2 4.1 : (4, (S', \langle S_2, A\rangle, \lambda)_{ins}, 5)$ $r_2 4.2 : (4, (\langle S_2, A\rangle, A', \lambda)_{ins}, 1)$	$r_2 5.1 : (5, (\lambda, S', \lambda)_{del}, 4)$	
$r3$	$r_3 4.1 : (4, (A', \langle S_2, A\rangle, \lambda)_{ins}, 5)$ $r_3 4.2 : (4, (\langle S_2, A\rangle, A', \lambda)_{ins}, 1)$	$r_3 5.1 : (5, (\lambda, A', \lambda)_{del}, 4)$	
$r4$	$r_4 4.1 : (4, (\lambda, S', \lambda)_{del}, 1)$		
$r5$	$r_5 4.1 : (4, (\lambda, A', \lambda)_{del}, 1)$		

The strictness of the two subset relations stated above follow from Examples 3.1 and 3.2, respectively. □

5. Reducing components for some super-linear classes

In this section, we show that with GCID systems of ID size s and $t+1$ components we can describe $\mathcal{L}_\cup(\mathcal{L}_*^2(\text{LIN}))$, based on simulation results for LIN with GCID systems of ID size s and t components. According to Proposition 2.1, $\mathcal{L}_*^2(\text{LIN}) = \text{LIN}^2 \cup (\mathfrak{L} \cup \mathfrak{L}^R) \cup \mathbb{L}$. We first provide independent descriptions of the three stated subsets and then describe $\mathcal{L}_*^2(\text{LIN})$ by GCID systems. Using Remark 2.9, we get the desired result.

Lemma 5.1. *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 $\text{LIN} \subseteq \text{GCID}_X(t; n, i', i''; m, j', j'')$ was shown by a simple-deleting simulation, then $\text{LIN}^2 \subseteq \text{GCID}_X(t+1; n, i', i''; m, j', j'')$.*

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 $N_1 \cap N_2 = \emptyset$. We can assume that the rules are of the forms $p_i : X_i \to aY_i$, $q_i : X_i \to Y_i a$ and $h_i : X_i \to \lambda$, with $X_i, Y_i \in N_i$ for $= 1, 2$. Since $\text{LIN} \subseteq \text{GCID}_{SD}(t; n, i', i''; m, j', j'')$, each of the rule types p_i, q_i, h_i can be simulated by rules of a simple-deleting GCID system $\mathit{\Pi}_i = (t, V_i, T, \{S_i\}, H_i, 1, 1, R_i)$ for $i = 1, 2$, each of size $(t; n, i', i''; m, j', j'')$. The h_i rule type is simulated by the GCID rules $h_i 1.1 : (1, (\lambda, X_i, \lambda)_{del}, 1)$. First, consider the case when $i' \geq 1$ and $i'' = 0$. We construct a graph-controlled ins-del system

$$\Pi = (t+1, V_1 \cup V_2 \cup \{S_2'\}, T, \{\lambda, S_1S_2'\}, H_1 \cup H_2 \cup \{r1.1, r(t+1).1\}, 1, 1, R)$$

for $L(G_1)L(G_2)$, with $R = ((R_1 \cup R_2) \setminus \{h_11.1 : (1, (\lambda, X_1, \lambda)_{del}, 1)\}) \cup R'$ where R' is the set with the following three rules: (i) $h_11.1 : (1, (\lambda, X_1, \lambda)_{del}, t+1)$, (ii) $r(t+1).1 : (t+1, (S_2', S_2, \lambda)_{ins}, 1)$, (iii) $r1.1 : (1, (\lambda, S_2', \lambda)_{del}, 1)$.

The linear rules are simulated in the first t components, since L_1 and L_2 are generated by Π_1 and Π_2, respectively. There is no interference of the rules in R_1, R_2, since $N_1 \cap N_2 = \emptyset$ and the GCID systems Π_1, Π_2 handle terminals properly.

We start with the axiom S_1S_2'. From S_1 using R_1, a string $w_1 \in L_1$ is produced first and the simulation of the linear rules in G_1 ends with the application of $h_11.1 : (1, (\lambda, X_1, \lambda)_{del}, t+1) \in R'$ which leads to w_1S_2' in $C(t+1)$. The only rule in $C(t+1)$ is applied, which inserts S_2 after S_2' and moves back to $C1$. Starting with $w_1S_2'S_2$ from C_1, $w_2 \in L(G_2)$ is generated, reaching to the configuration $(w_1S_2'w_2)_1$, where S_2' is deleted by $r1.1$. Note that, in $C1$, the rule $r1.1 : (1, (\lambda, S_2', \lambda)_{del}, 1)$ can be applied at any point of the derivation. If this rule is applied before applying $h_11.1$ (*i.e.*, before producing $w_1 \in L(G_1)$), then when $h_11.1$ is applied, there will be no S_2' present in the string at $C(t+1)$ and the derivation is stuck in $C(t+1)$. Recall that $C(t+1)$ is not the target component. Hence by the preceding discussion, any string derivable by Π belongs to L_1L_2. Obviously, the resulting GCID system is returning and simple-deleting. Also, if Π_1 and Π_2 are tree-structured, then Π is tree-structured, as well. Since LIN^2 is closed under reversal; see Proposition 2.2, the case when $i' = 0, i'' \geq 1$ follows from Proposition 4.1. □

Combining the results of Remark 2.9 and Lemma 5.1, we have the following.

Corollary 5.2. *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 $2\text{-LIN} \subseteq \mathrm{GCID}_X(t+1; n, i', i''; m, j', j'')$ follows.*

Lemma 5.3. *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 $\mathrm{LIN} \subseteq \mathrm{GCID}_X(t; n, i', i''; m, j', j'')$ is shown by a simple-deleting simulation, then $\mathfrak{L} \cup \mathfrak{L}^R \subseteq \mathrm{GCID}_X(t+1; n, i', i''; m, j', j'')$ can be concluded.*

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 $N_1 \cap N_2 = \emptyset$, where the rules are of the forms $p_i : X_i \to aY_i$, $q_i : X_i \to Y_ia$ and $h_i : X_i \to \lambda$, with $X_i, Y_i \in N_i$ for $= 1, 2$. Notice that these assumptions are the basis of Proposition 2.5, so that we can take over the grammar construction G for $L(G_1)^*L(G_2)$ and consider this as a basis for the GCID system that we are going to build to describe $L(G) = L(G_1)^*L(G_2)$.

Since $\mathrm{LIN} \subseteq \mathrm{GCID}(t; n, i', i''; m, j', j'')$, for $i = 1, 2$, each of the rule types p_i, q_i, h_i can be simulated by rules of a simple-deleting GCID system Π_i of size $(t; n, i', i''; m, j', j'')$. Let $\Pi_i = (t, V_i, T, \{S_i\}, H_i, 1, 1, R_i)$. First, consider the case when $i' \geq 1$, $i'' = 0$. We now show the inclusion $\mathfrak{L} \subseteq \mathrm{GCID}_{SD}(t+1; n, i', i''; m, j', j'')$. We construct a graph-controlled ins-del system for $L(G)$ as follows.

$$\Pi = (t+1, V_1 \cup V_2 \cup \{S'\}, T, \{\lambda, S'S_2\}, H_1 \cup H_2 \cup \{r(t+1).1, r(t+1).2\}, 1, 1, R)$$

with $R = ((R_1 \cup R_2) \setminus \{h_11.1 : (1, (\lambda, X_1, \lambda)_{del}, 1), h_21.1 : (1, (\lambda, X_2, \lambda)_{del}, 1)\}) \cup R'$, where R' is the set of the following rules.

$$\begin{array}{ll} h_11.1 : (1, (\lambda, X_1, \lambda)_{del}, t+1) & h_21.1 : (1, (\lambda, X_2, \lambda)_{del}, t+1) \\ r(t+1).1 : (t+1, (S', S_1, \lambda)_{ins}, 1) & r(t+1).2 : (t+1, (\lambda, S', \lambda)_{del}, 1) \end{array}$$

The linear rules are simulated in the first t components and there is no interference of the rules of R_1 and R_2, since $N_1 \cap N_2 = \emptyset$ and the GCID systems Π_1, Π_2 handle terminals properly. We start with the axiom $S'S_2$ (notice that this corresponds to the rule $S \to S'S_2$ of grammar G), which produces $w_2 \in L_2$ and the simulation ends with the application of $h_2 1.1 \in R'$ which leads the derivation to $C(t+1)$. The rules in $C(t+1)$ initiate the simulation of the rules of G_1 by inserting S_1 after S' and thereafter continuing with $S'S_1w_2$ from $C1$, the configuration $(S'w_1w_2)_{t+1}$ is reached, with $w_1 \in L_1$ and $w_2 \in L_2$. Now if $r(t+1).1$ is applied (which corresponds to applying the rule $S' \to S'S_1$ in G), the simulation of G_1 is restarted, and after generating $w_1 \in L(G_1)$ for any desired number of times, the whole derivation stops. With this observation, we conclude that $\varPi$ generates $L(G) = (L(G_1))^* L(G_2) \in \mathfrak{L}$. Obviously, the resulting GCID system $\varPi$ is returning as well as tree-structured if $\varPi_1$ and $\varPi_2$ are.

Consider the case when $i' = 1$, but we want to prove the inclusion for $\mathfrak{L}^R$. We aim at constructing a GCID system $\varPi'$ for $L_2L_1^*$. The simulation is identical to the one just presented except for the axiom, which is S_2S' in this case.[4]

The case when $i' = 0$ and $i'' \geq 1$ follows from the fact that $\mathfrak{L} \cup \mathfrak{L}^R$ is closed under reversal; see Propositions 2.2 and 4.1. □

Combining the results of Remark 2.9 and Lemma 5.3, we have the following.

Corollary 5.4. *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 $\mathrm{LIN} \subseteq \mathrm{GCID}_X(t; n, i', i''; m, j', j'')$ is shown by a simple-deleting simulation, then $\mathcal{L}_\cup(\mathfrak{L} \cup \mathfrak{L}^R) \subseteq \mathrm{GCID}_X(t+1; n, i', i''; m, j', j'')$.*

Lemma 5.5. *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 $\mathrm{LIN} \subseteq \mathrm{GCID}_X(t; n, i', i''; m, j', j'')$ is shown by a simple-deleting simulation, then $\mathbb{L} \subseteq \mathrm{GCID}_X(t+1; n, i', i''; m, j', j'')$.*

Proof. We recall that $\mathbb{L} = \{L_1^*L_2^* \mid L_1, L_2 \in \mathrm{LIN}\}$. For $i = 1, 2$, let $G_i = (N_i, T, S_i, P_i)$ be a linear grammar of L_i and let us assume that $N_1 \cap N_2 = \emptyset$. Let the rules of P_i $(i = 1, 2)$ be of the forms $p_i : X_i \to aY_i$, $q_i : X_i \to Y_ia$ and $h_i : X_i \to \lambda$, with $X_i, Y_i \in N_i$. 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 $\varPi_i = (t, V_i, T, \{S_i\}, H_i, 1, 1, R_i)$, each of size $(t; n, i', i''; m, j', j'')$, that can handle terminals properly. The rule type h_i is simulated by the GCID rules $h_i 1.1 : (1, (\lambda, X_i, \lambda)_{del}, 1)$. First, consider the case when $i' \geq 1$ and $i'' = 0$. We construct a graph-controlled ins-del system $\varPi = (t+1, V_1 \cup V_2 \cup \{S_1', S_2'\}, T, \{S_1'S_2'\}, H \cup \{r_1(t+1).1, r_2(t+1).1, s_1(t+1).1, s_2(t+1).1\}, t+1, t+1, (R \setminus \hat{R}) \cup R' \cup R'')$ such that $L(\varPi) = L_1^*L_2^*$, 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 formed by the following four rules:

$$\begin{array}{ll} r_1(t+1).1 : (t+1, (S_1', S_1, \lambda)_{ins}, 1) & r_2(t+1).1 : (t+1, (S_2', S_2, \lambda)_{ins}, 1) \\ s_1(t+1).1 : (t+1, (\lambda, S_1', \lambda)_{del}, t+1) & s_2(t+1).1 : (t+1, (\lambda, S_2', \lambda)_{del}, t+1) \end{array}$$

Notice that this construction parallels the characterization of $\mathbb{L}$ stated in Proposition 2.6.

The linear rules of G_i, $i \in \{1, 2\}$, are simulated in the components $C1, \ldots, Ct$, since $L_1, L_2 \in \mathrm{GCID}(t; n, i', i''; m, j', j'')$. There is no interference of the rules among R_1 and R_2 since $N_1 \cap N_2 = \emptyset$. The initial and as well the final component is $C(t+1)$. Starting with $S_1'S_2'$ in $C(t+1)$, any rule in R'' can be applied. Assume that $r_1(t+1).1$ is applied. This will insert S_1 after S_1', and the resultant string $S_1'S_1S_2'$ is sent to $C1$. In $C1$, the start symbol S_1 of G_1 will start simulating the rules for L_1 and the simulation ends by applying the rule $h_1 1.1$ of R'. This will produce a configuration of the form $(S_1'w_1S_2')_{t+1}$, with $w_1 \in L_1$. Note that after the rule $h_1 1.1$ is applied, none of variables of G_1 will be present in the string and the string is back to $C(t+1)$. Repeated applications of these rules $r_1(t+1).1, \ldots, h_1 1.1$, will produce the Kleene closure part of L_1. Assuming that $r_2(t+1).1$ is applied at any point of time in the derivation when the string is available in $C(t+1)$, S_2, the start symbol of G_2, is introduced to the right of S_2' and the string is sent to C_1. In C_1, S_2 would start simulating the rules for L_2 and the simulation ends by applying the rule $h_2 1.1$ of R'. This thus generates

[4]In [7], the reader can also find a corresponding grammatical characterization of $\mathfrak{L}^R$.

TABLE 4. GCID$(4;2,1,0;1,0,0)$ describing languages of $\{L_1^*L_2^* : L_1, L_2 \in \text{LIN}\}$ according to Lemma 5.5.

Phase	Component C1	Component C2	Component C3
$p1$	$p_11.1 : (1,(S_1,p_1,\lambda)_{ins},2)$ $p_11.2 : (1,(p_1,aB,\lambda)_{ins},3)$	$p_12.1 : (2,(\lambda,S_1,\lambda)_{del},1)$	$p_13.1 : (3,(\lambda,p_1,\lambda)_{del},1)$
$q1$	$q_11.1 : (1,(B,q_1,\lambda)_{ins},2)$ $q_11.2 : (1,(q_1,S_1b,\lambda)_{ins},3)$	$q_12.1 : (2,(\lambda,B,\lambda)_{del},1)$	$q_13.1 : (3,(\lambda,q_1,\lambda)_{del},1)$
$p2$	$p_21.1 : (1,(S_2,p_2,\lambda)_{ins},2)$ $p_21.2 : (1,(p_2,cD,\lambda)_{ins},3)$	$p_22.1 : (2,(\lambda,S_2,\lambda)_{del},1)$	$p_23.1 : (3,(\lambda,p_2,\lambda)_{del},1)$
$q2$	$q_21.1 : (1,(D,q_2,\lambda)_{ins},2)$ $q_21.2 : (1,(q_2,S_2d,\lambda)_{ins},3)$	$q_22.1 : (2,(\lambda,D,\lambda)_{del},1)$	$q_23.1 : (3,(\lambda,q_2,\lambda)_{del},1)$
$h1$	$h_11.1 : (1,(\lambda,S_1,\lambda)_{del},4)$		
$h2$	$h_21.1 : (1,(\lambda,S_2,\lambda)_{del},4)$		
Phase	Component C4		
$r1$	$r_14.1 : (4,(S_1',S_1,\lambda)_{ins},1)$		
$r2$	$r_24.1 : (4,(S_2',S_2,\lambda)_{ins},1)$		
$s1$	$s_14.1 : (4,(\lambda,S_1',\lambda)_{del},4)$		
$s2$	$s_24.1 : (4,(\lambda,S_2',\lambda)_{del},4)$		

a string w_2 of L_2 after S_2'. Repeated applications of these rules $r_2(t+1).1,\ldots,h_21.1$, will produce the Kleene closure part of L_2. In $C(t+1)$, at any point of time, $S_1(t+1).1$ (or $S_2(t+1).1$) is applied, then it will delete S_1' (correspondingly, S_2'), thus the production of strings from L_1 (correspondingly, from L_2) is stopped. If both these rules $s_i(t+1).1$, $i \in \{1,2\}$, are applied in the initial stage itself, then λ is generated. With these details, it is easy to verify that the rules of Π generate $L_1^*L_2^*$. As claimed, Π has $t+1$ components. Just by re-labelling the components, the resulting GCID system can be made returning and tree-structured if Π_1 and Π_2 are. The case when $i'=0$ and $i''\geq 1$ follows from Propositions 2.3 and 4.1. □

Table 4 shows how the construction of the previous proof works for the concrete case of $L_1 = \{a^nb^n \mid n \geq 0\}$ and $L_2 = \{c^nd^n \mid n \geq 0\} \in \text{LIN}$. If the reader tries to generate, say, *abaabbcdcd*, (s)he will understand how the phases work. Combining the results of Remark 2.9 and Lemma 5.5, we have the following.

Corollary 5.6. *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 it is true that $\mathcal{L}_\cup(\mathbb{L}) \subseteq$* GCID$_X(t+1;n,i',i'';m,j',j'')$.

Remark 5.7. The proof of Lemma 5.5 can be extended to describe the language class $\mathbb{L}' = \bigcup_{k\geq 1}\{L_1^*L_2^*\ldots L_k^* : L_i \in \text{LIN for } 1 \leq i \leq k\}$. Consider the GCID system Π' as in Lemma 5.5 with alphabet and label set extended from 2 to k. Let the 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 also $L_1^*L_2^*\ldots L_k^* \in$ GCID$(t+1;n,i',i'';m,j',j'')$ under the assumptions stated therein. In particular, $L_1^*L_2^*L_3^*$ can be generated by GCID$(t+1;n,i',i'';m,j',j'')$ whenever L_i can be generated by GCID$(t;n,i',i'';m,j',j'')$ for each $i=1,2,3$.

Remark 5.8. To highlight how the reduction in the number of components happened, we invite the reader to observe the difference in the simulating rules that generate $L_1^*L_2^*$ according to Theorem 4.3 and Lemma 5.5, stated in Tables 3 and 4 respectively. We notice that in Theorem 4.3, strings of L_1^* and L_2^* are generated one after the other. However in Lemma 5.5, strings of L_1^* and L_2^* are generated simultaneously in an independent manner.

Combining the results of Lemmas 5.1, 5.3 and 5.5, we have the following main result that was claimed at the start of this section.

Theorem 5.9. *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 $\mathcal{L}_*^2(\mathrm{LIN}) \subseteq \mathrm{GCID}_X(t + 1; n, i', i''; m, j', j'')$.*

Combining Remark 2.9 with Theorem 5.9, we have the following.

Corollary 5.10. *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 $\mathcal{L}_\cup(\mathcal{L}_*^2(\mathrm{LIN})) \subseteq \mathrm{GCID}_X(t + 1; n, i', i''; m, j', j'')$.*

Corollary 5.11. *Combining Proposition 4.2 and Corollary 5.10, we conclude that:*

- $\mathcal{L}_\cup(\mathcal{L}_*^2(\mathrm{LIN})) \subsetneq \mathrm{GCID}_{SDT}(4; 2, 1, 0; 1, 0, 0) \cap \mathrm{GCID}_{SD}(4; 2, 0, 1; 1, 0, 0)$.
- $\mathcal{L}_\cup(\mathcal{L}_*^2(\mathrm{LIN})) \subsetneq \mathrm{GCID}_{SD}(4; 1, 1, 0; 1, 0, 0) \cap \mathrm{GCID}_{SD}(4; 1, 0, 1; 1, 0, 0)$.

The strictness of these two subset relations follow from Remark 5.7and (item (2) of) Proposition 2.1, as $\{a^k b^k \mid k \geq 0\}^* \{c^k d^k \mid k \geq 0\}^* \{e^k f^k \mid k \geq 0\}^*$ can serve as a witness language.

6. Summary and future challenges

In this paper, we simulate the rewriting grammars of certain super-linear languages (languages between LIN and CF) using GCID systems of sizes that are not known to describe RE (not even CF). 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. Specifically, we have proved that if a graph-controlled ins-del system of ID size s with t components can describe LIN, then there is a graph-controlled ins-del system of ID size s with $t + 2$ components that will describe $\mathcal{L}_{reg}(\mathrm{LIN})$, the regular closure of LIN. We also proved that we do not need two but only one additional component to describe $\mathcal{L}_\cup(\mathcal{L}_*^2(\mathrm{LIN}))$, a subset of $\mathcal{L}_{reg}(\mathrm{LIN})$. In fact, the class $\mathcal{L}_\cup(\mathcal{L}_*^2(\mathrm{LIN}))$ seems to have been never considered before. We feel that for simulation results as the ones presented in this paper, this class might be an interesting family of languages to look at. These simulation results might also inspire further language-theoretic investigations of the other (new) super-linear language classes introduced in this paper.

One natural challenge is to see with which resources we can still describe all context-free languages, not only super-linear languages as endeavored in this paper.

A second challenge would be to prove corresponding upper bounds, *i.e.*, we should find (classical) grammar families (not describing RE already) that can simulate certain (graph-controlled) ins-del systems with a given size bound, like the ones mentioned in this paper. This would then also rule out RE results for these systems.

Acknowledgements. Some part of the work done by the second author was during the author's visits to University Trier, Germany, in December 2016. The possibility to use some overhead money from a DFG grant to support this stay is gratefully acknowledged. The authors also thank Martin Kutrib for his comments and suggestions on super-linear languages.

References

[1] H. Fernau, L. Kuppusamy and I. Raman, Descriptional complexity of graph-controlled insertion-deletion systems, in *18th International Conference on Descriptional Complexity of Formal Systems (DCFS)*, edited by C. Câmpeanu, F. Manea, J. O. Shallit. Vol. 9777 of *LNCS*. Springer (2016) 111–125.

[2] H. Fernau, L. Kuppusamy and I. Raman, Computational completeness of path-structured graph-controlled insertion-deletion systems, in *22nd International Conference on Implementation and Application of Automata CIAA*, edited by A. Carayol, C. Nicaud. Vol. 10329 of *LNCS*. Springer (2017) 89–100.

[3] H. Fernau, L. Kuppusamy and I. Raman, Graph-controlled insertion-deletion systems generating language classes beyond linearity, in *19th IFIP WG 1.02 International Conference on Descriptional Complexity of Formal Systems DCFS*, edited by G. Pighizzini, C. Câmpeanu. Vol. 316 of *LNCS*. Springer (2017) 128–139.

[4] H. Fernau, L. Kuppusamy and I. Raman, On path-controlled insertion-deletion systems. Submitted to *Acta Inform.* (2017).

[5] H. Fernau, L. Kuppusamy and I. Raman, On the computational completeness of graph-controlled insertion-deletion systems with binary sizes. *Theor. Comput. Sci.* **682** (2017) 100–121. Special Issue on Languages and Combinatorics in Theory and Nature.

[6] H. Fernau, L. Kuppusamy and I. Raman. On the generative power of graph-controlled insertion-deletion systems with small sizes. *J. Autom. Lang. Comb.* **22** (2017) 61–92.

[7] H. Fernau, L. Kuppusamy and I. Raman, Properties of language classes between linear and context-free. Submitted to *J. Autom. Lang. Comb.* (2017).

[8] R. Freund, M. Kogler, Y. Rogozhin and S. Verlan, Graph-controlled insertion-deletion systems, in *Proceedings 12th Annual Workshop on Descriptional Complexity of Formal Systems (DCFS)*, edited by I. McQuillan and G. Pighizzini. Vol. 31 of *EPTCS* (2010) 88–98.

[9] B. S. Galiukschov, Semicontextual grammars (in Russian). *Mat. logica i mat. ling., Kalinin Univ.* (1981) 38–50 .

[10] G. Horváth and B. Nagy, Pumping lemmas for linear and nonlinear context-free languages. *Acta Univ. Sapientiae Inform.* **2** (2010) 194–209.

[11] M. Kutrib and A. Malcher, Finite turns and the regular closure of linear context-free languages. *Discret. Appl. Math.* **155** (2007) 2152–2164.

[12] Gh. Păun, G. Rozenberg and A. Salomaa, DNA Computing: New Computing Paradigms. Springer (1998).

[13] S. Verlan. Recent developments on insertion-deletion systems. *Comput. Sci. J. Mold.* **18** (2010) 210–245.