Theoretical Computer Science 682 (2017) 100–121

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

On the computational completeness of graph-controlled insertion–deletion systems with binary sizes

Henning Fernau [a,*], Lakshmanan Kuppusamy [b], Indhumathi Raman [c]

[a] Fachbereich 4 – Abteilung Informatikwissenschaften, Universität Trier, D-54286 Trier, Germany
[b] School of Computer Science and Engineering, VIT University, Vellore-632 014, India
[c] School of Information Technology and Engineering, VIT University, Vellore-632 014, India

ARTICLE INFO

Article history:
Received 21 September 2016
Received in revised form 16 December 2016
Accepted 20 January 2017
Available online 31 January 2017

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

ABSTRACT

A graph-controlled insertion–deletion (GCID) system is a regulated extension of an insertion–deletion system. Such a system has several components and each component has some insertion–deletion rules. The 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. The parameters in the size $(k; n, i', i''; m, j', j'')$ of a GCID system denote (from left to right) the maximum number of components, the maximal length of the insertion string, the maximal length of the left context for insertion, the maximal length of the right context for insertion; the last three parameters follow a similar representation with respect to deletion.

In this paper, we discuss the computational completeness of the families of GCID systems of size $(k; 1, i', i''; 1, j', j'')$ with $k \in \{3, 5\}$ and for (nearly) all values of $i', i'' j', j'' \in \{0, 1\}$. All proofs are based on the simulation of type-0 grammars given in *Special Geffert Normal Form* (SGNF). The novelty in our proof presentation is that the context-free and the non-context-free rules of the given SGNF grammar are simulated by GCID systems of different sizes and finally we combine them by stitching and overlaying to characterize the recursive enumerable languages. This proof presentation greatly simplifies and unifies the proof of such characterization results. We also connect some of the obtained GCID simulations to the domain of insertion–deletion P systems.

1. Introduction

Insertion and deletion operations often occur in DNA processing and RNA editing. During the theoretical process of mismatched annealing in DNA sequences, certain segments of strands are either inserted or deleted [28]. In the process of RNA editing, some fragments of messenger RNA are inserted or deleted [3,4]. The motivation for the insertion operation is found in [11], where this operation and its iterated variant were introduced as a generalization of concatenation and Kleene's closure. The deletion operation was first discussed in [16], where the deletion was considered as a quotient-like operation. Insertion and deletion operations together were introduced into formal language theory in [18]. The corresponding grammatical mechanism is called *insertion–deletion system* (abbreviated as ins–del system). Informally, the insertion and deletion

* Corresponding author.
E-mail addresses: Fernau@uni-trier.de (H. Fernau), klakshma@vit.ac.in (L. Kuppusamy), indhumathi.r@vit.ac.in (I. Raman).

http://dx.doi.org/10.1016/j.tcs.2017.01.019
0304-3975/

operations of an ins–del system are defined as follows: if a string η is inserted between two parts w_1 and w_2 of a string w_1w_2 to get $w_1\eta w_2$, we call the operation *insertion*, whereas if a substring δ is deleted from a string $w_1\delta w_2$ to get w_1w_2, we call the operation *deletion*. Suffixes of w_1 and prefixes of w_2 are called *contexts*.

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

Several variants of ins–del systems have been introduced in the literature, like ins–del P systems [2,21], tissue P systems with ins–del rules [24], context-free ins–del systems [25], matrix ins–del systems [23,27,22], random context and semi-conditional ins–del systems [14], etc. All the mentioned papers (as well as [17,29]) attempted to characterize the recursively enumerable languages (*i.e.*, they show *computational completeness*) using ins–del systems. We refer to the survey article [31] for details of variants of ins–del systems; this survey also discusses some proof techniques for showing computational completeness results. We also refer to the thesis [12] for various recent results on variants of ins–del systems.

One of the important variants of ins–del systems is *graph-controlled ins–del systems* (abbreviated as GCID systems) introduced in [9] and further studied in [15] and [7]. In such a system, the concept of a component is introduced and is associated with every insertion or deletion rule. The transition is performed by choosing any applicable rule from the set of rules of the current component and by moving the resultant string to the target component specified in the rule. Incidentally, when the underlying graph forms a tree structure, this system can be interpreted as an insertion–deletion P system [21], where several membranes (possibly, with nesting membrane structures) contain objects (in our case, strings) and based on the rules available in the membrane, they evolve and are sent to an adjacent membrane, as specified in the rule as its *target* membrane. For more details, see [26].

The objective of this paper is to obtain computational completeness results of the GCID systems with few components and small descriptional complexity measures of ins–del rules such that the underlying control graph is linear wherever possible. In this paper, considering all possible values of $i', i'', j', j'' \in \{0, 1\}$, we aim to find which GCID systems with k components and the underlying ins–del size $(1, i', i''; 1, j', j'')$, denoted as $\mathrm{GCID}(k; n, i', i''; m, j', j'')$, can characterize the recursively enumerable languages.

The novelty in our proof presentation is as follows. We consider a type-0 grammar in *Special Geffert Normal Form* (SGNF), which has two fundamentally different types of rules: context-free ones and non-context-free ones. We independently simulate these two types with different (parts of) graph-controlled ins–del systems that have also different properties, including the number of components and sizes. Such a part looks (formally) as a GCID Π, but we are not interested in the language generated by Π, but rather in the set of strings over V (*i.e.*, the set of sentential forms) that can be transferred into component i_f, assuming a certain (usually infinite) set of strings is fed into component i_0. So, such a GCID rather works like a transducer, translating languages over V into itself.

We then *stitch* together a simulation of context-free rules and a simulation of non-context-free rules (note that the simulating rules may have different sizes) by *overlaying* the rules of the two simulations, to obtain a characterization of the class of recursively enumerable languages. The concepts of *stitching* and *overlaying* are the hallmarks of this paper. These concepts are discussed in detail in Sec. 2.3. This approach provides a new perspective to the proof technique for proving such characterization results. Appropriate use of *rule markers* avoids malicious derivations with the new components obtained by stitching and/or overlaying.

The results of this paper improve the previous results (available in the literature) in the following ways.

1. In [9], Freund et al. showed that, in our notation, $\mathrm{GCID}(4; 1, 1, 0; 1, 1, 0)$ and $\mathrm{GCID}(4; 1, 1, 0; 1, 0, 1)$ are computationally complete and in [13], the authors proved that $\mathrm{GCID}(3; 1, 2, 0; 1, 1, 0)$ and $\mathrm{GCID}(3; 1, 1, 0; 1, 2, 0)$ are computationally complete. In this paper, we improve the above-mentioned results (i) of [9] by reducing the number of components from 4 to 3 and (ii) of [13] by reducing the length of left context of insertion/deletion from 2 to 1; see Theorem 28 and Theorem 30.
2. In [19], it has been proved that ins–del systems (that is, GCIDs with one component) with size $(2, 0, 0; 1, 1, 1)$ yield RE. If one wishes to reduce the length of the insertion string from 2 to 1 in the former result, then in this paper we prove that the number of components of a GCID system increases from 1 to 5; see Theorem 34. Reducing the components from 5 to, say 4, is left open.
3. In [28], it has been proved that an ins–del systems with size $(1, 1, 1; 2, 0, 0)$ also give RE. Wishing to reduce the length of the deletion string from 2 to 1 in this result, we prove in this paper that the number of components again increases from 1 to 5; see Theorem 33. Reducing the components from 5 is left open, again.
4. In [29], it has been proved that ins–del systems with size $(1, 1, 1; 1, 1, 1)$ characterize RE. If we desire to have one-sided context for insertion/deletion, then it is proved in [19] that the ins–del systems of size $(1, 1, 1; 1, 1, 0)$ or $(1, 1, 0; 1, 1, 1)$ cannot characterize RE. It is therefore obvious that we need at least 2 components in a graph-controlled ins–del system of sizes $(1, 1, 1; 1, 1, 0)$ and $(1, 1, 0; 1, 1, 1)$ to characterize RE. In this paper, we prove the characterization with 3 components, maintaining a linear structure of the control graph only in the former case; see Theorems 27, 28. If we further desire to have context-free insertion/deletion (that is, randomly insert/delete in the string), then the number

of components of a GCID system increases from 1 to 5, still maintaining a linear structure of the control graph; see Theorems 33, 34.

In conclusion, the most significant contributions of this paper are as follows. It is conjectured in [19] that an ins–del system with weight ω (defined by $n + i' + i'' + m + j' + j'') = 4$ is not computationally complete. However, if we allow more components (say 3 or 5) to act, then graph-controlled ins–del systems with underlying weight $\omega = 4$ or 5 all characterize RE. More specifically,

1. If $\omega = 4$, then GCID systems with 3 or 5 components describe RE.
2. If $\omega = 5$, then GCID systems with 3 components describe RE.

The results show some trade-off between the weight of a system and the number of components to arrive at computational completeness.

2. Preliminaries

We assume that the readers are familiar with the standard notations used in formal language theory. Nevertheless, we now recall a few notations here. Let $\mathbb{N}$ denote the set of positive integers, and $[1 \dots k] = \{i \in \mathbb{N} \colon 1 \leq i \leq k\}$. Given an *alphabet* (finite set) Σ, Σ^* denotes the free monoid generated by Σ. The elements of Σ^* are called *strings* or *words*; λ denotes the empty string. For a string $w \in \Sigma^*$, $|w|$ denotes the length of a string w and w^R denotes the reversal (mirror image) of w. Likewise, L^R and $\mathcal{L}^R$ are understood for languages L and language families $\mathcal{L}$. FIN, REG, LIN, CF, CS and RE denote the families of the finite, regular, linear, context-free, context-sensitive and recursively enumerable languages, respectively.

For the computational completeness results, we are using the fact that type-0 grammars in SGNF are known to characterize the RE languages.

Definition 1. ([9]) A type-0 grammar $G = (N, T, P, S)$ is said to be in *Special Geffert Normal Form*, SGNF for short, if

- N decomposes as $N = N' \cup N''$, where $N'' = \{A, B, C, D\}$ and N' contains at least the two nonterminals S and S',
- the only non-context-free rules in P are the two erasing rules $AB \to \lambda$ and $CD \to \lambda$,
- the context-free rules are of the following forms:

$$X \to Yb \text{ or } X \to bY \text{ where } X, Y \in N',\ X \neq Y,\ b \in T \cup N'',\ \text{or } S' \to \lambda.$$

The way the normal form is constructed is described in [9] and is based on [10]. Also, the derivation of a string is done in two phases. First, the context-free rules are applied repeatedly and the phase I is completed by applying the rule $S' \to \lambda$ in the derivation. In phase II, only the non-context-free erasing rules are applied repeatedly and the derivation ends. This normal form has already been used in [9,13,27] in the domain ins–del systems and their variants. The advantage of using this normal form is that as these context-free rules are more like linear type, there can be at most only one nonterminal from N' present in the derivation of G. We exploit this observation in the proofs of several lemmas.

As usual, we write $\Rightarrow_x$ to denote a single derivation step using rule x, and $\Rightarrow_G$ (or $\Rightarrow$ if no confusion arises) denotes a single derivation step using any rule of G. Then, $L(G) = \{w \in T^* \mid S \Rightarrow^* w\}$, where $\Rightarrow$ is the reflexive transitive closure of $\Rightarrow$.

2.1. Insertion–deletion systems

We now give the basic definition of insertion–deletion systems, following [18,28].

Definition 2. An *insertion–deletion system*, or *ins–del system* for short, is a construct $\gamma = (V, T, A, R)$, where V is an alphabet, $T \subseteq V$ is the terminal alphabet, A is a finite language over V, R is a finite set of triplets of the form $(u, \eta, v)_{ins}$ or $(u, \delta, v)_{del}$, where $(u, v) \in V^* \times V^*$, $\eta, \delta \in V^+$.

The pair (u, v) is called the *context*, η is called the *insertion string*, δ is called the *deletion string* and $x \in A$ is called an *axiom*. If one of the u or v is λ for all the insertion (deletion) contexts, then we call the insertion (deletion) as *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). 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 $u\delta v \to uv$.

For $x, y \in V^*$ we write $x \Rightarrow y$ if y can be obtained from x by using either an insertion rule or a deletion rule. The language generated by γ is defined by $L(\gamma) = \{w \in T^* \mid x \Rightarrow_* w,\ \text{for some } x \in A\}$ where $\Rightarrow_*$ denotes (as often with ins–del systems, also to distinguish them from derivations within more traditional types of grammars) the reflexive and transitive closure of the relation $\Rightarrow$.

2.2. Graph-controlled insertion–deletion systems

A *graph-controlled insertion–deletion system* (GCID 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,
- $A \subseteq V$ 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 deletion rule of the form $(u, \delta, v)_{del}$ and $i, j \in [1 \dots k]$.

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

A *configuration* of Π is represented by $(w)_i$, where i is the number of the current component (initially i_0) and w is the current string. In that case, we also say that w has entered component Ci. If the initial component itself is the final component, then we call the system to be a *returning* GCID system. We denote by $(w)_i \Rightarrow_l (w')_j$ or $(w')_j \Leftarrow_l (w)_i$ if $(w')_j$ is derived from $(w)_i$ on applying a rule $l : (i, r, j)$ in R. By $(w)_i \underset{l'}{\overset{l}{\rightleftarrows}} (w')_j$, we mean that $(w')_j$ is derivable from $(w)_i$ using rule l and $(w)_i$ is derivable from $(w')_j$ using rule l'. For a returning GCID system Π with initial component i_0, we also use the following derived form $\Rightarrow'$ of a derivation relation: $(w)_{i_0} \Rightarrow' (w')_{i_0}$ is true if there is a sequence of derivation steps (via $\Rightarrow$) that starts with w in Ci_0 and ends with w' in Ci_0, but no intermediate string in this derivation enters Ci_0. If $\Rightarrow'_*$ denotes the reflexive transitive closure of $\Rightarrow'$, then clearly for returning systems Π,

$$L(\Pi) = \{w \in T^* \mid (S)_{i_0} \Rightarrow'_* (w)_{i_0},\ S \in A\}.$$

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

A size $s_1 = (k_1; n_1, i'_1, i''_1; m_1, j'_1, j''_1)$ is said to be weaker than a size $s_2 = (k_2; n_2, i'_2, i''_2; m_2, j'_2, j''_2)$ if $x_1 \geq x_2$ for all $x \in \{k, n, m, i', i'', j', j''\}$.

We now discuss some examples to clarify the work of GCID systems.

Example 3. Consider the copy language $L_1 = \{ww : w \in \{a, b\}^*\}$. A returning GCID system Π_1 of size $(3; 1, 1, 0; 1, 0, 0)$, with $L_1 = L(\Pi_1)$, is described in the following. $\Pi_1 = (3, \{\#, \$, a, b\}, \{\#\$\}, \{r1.1, r1.2, r1.3, r2.1, r2.2, r2.3, r3.1, r3.2, r3.3\}, 1, 1, R)$, where the rules of R are listed below. Here, one can also see our convention of labeling rules in a way that also makes the component explicit by using the common infix 'i.' to refer to rules in Ci.

$r1.1 : (1, (\#, a, \lambda)_{ins}, 2)$	$r1.2 : (1, (\#, b, \lambda)_{ins}, 3)$	$r1.3 : (1, (\lambda, \$, \lambda)_{del}, 2)$
$r2.1 : (2, (\$, a, \lambda)_{ins}, 1)$	$r2.2 : (2, (\lambda, \#, \lambda)_{del}, 1)$	
$r3.1 : (3, (\$, b, \lambda)_{ins}, 1)$		

Starting from #$ as the axiom, if an a is inserted by rule $r1.1$ after the marker #, then in $C2$, another a is inserted after the marker $ and the string is sent back to $C1$. Similarly, if $r1.2$ is applied, this will introduce a b after # and in $C3$, one b is inserted after $. This process can be repeated and the non-terminals are deleted by rules $r1.3$ and $r2.2$. One can note that # and $ are used as markers that will help introduce the symbols at their right place. During the derivation, if $r2.2$ is applied just after $r1.2$, then it is possible to apply $r1.3$; the resultant string will however be in $C2$ which is not the target component. Hence, to successfully terminate, rules $r1.3$ and $r2.2$ have to be applied at the end. Let us explain a possible derivation by using the notation for configurations in the following:

$$(\#\$)_1 \Rightarrow (\#a\$)_2 \Rightarrow (\#a\$a)_1 \Rightarrow (\#ba\$a)_3 \Rightarrow (\#ba\$ba)_1 \Rightarrow' (\#bba\$bba)_1 \Rightarrow' (bbabba)_1$$

In general, by induction one can find that $(\#\$)_1 \Rightarrow'_* (\#w\$w)_1 \Rightarrow' (ww)_1$ for any $w \in \{a, b\}^*$. □

In [20], it is shown that the language $\{ba\}^+$ cannot be generated using ins–del systems with size $(1, 1, 0; 1, 1, 1)$ which is same as GCID$(1; 1, 1, 0; 1, 1, 1)$. The example below shows that two components are more powerful than one for systems of size $(1, y, z; 1, 1, 1)$ with $y + z \leq 1$.

Example 4. Consider the regular language $L_2 = \{ba\}^+$. L_2 is generated by the graph-controlled ins–del system $\Pi_2 = (2, \{\$, a, b\}, \{a, b\}, \{\$ba\}, \{r1.1, r1.2, r2.1\}, 1, 1, R)$, where the rules of R are: $r1.1 : (1, (\$, a, \lambda)_{ins}, 2)$, $r1.2 : (1, (\lambda, \$, \lambda)_{del}, 1)$, and $r2.1 : (2, (\$, b, \lambda)_{ins}, 1)$. Π_2 is a GCID system of size $(2; 1, 1, 0; 1, 0, 0)$. □

In [31], the following example is mentioned.

Example 5. Consider the linear language $L_3 = \{w \in \{a, b\}^* : |w|_a = |w|_b\}$. L_3 can be generated by the graph-controlled ins–del system $\Pi_3 = (2, \{a, b\}, \{a, b\}, \{\lambda\}, \{r1.1, r2.2\}, 1, 1, R)$, where the rules of R are: $r1.1 : (1, (\lambda, a, \lambda)_{ins}, 2)$ and $r2.1 : (2, (\lambda, b, \lambda)_{ins}, 1)$. Π_3 is a GCID system of size $(2; 1, 0, 0; 0, 0, 0)$. □

The (*underlying*) *control graph* of a graph-controlled insertion–deletion system Π with k components is defined to be a graph with k nodes labeled $C1$ 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 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 has a tree (for instance, a linear) structure, then Π can be viewed as an insertion–deletion P system (see [9]). Let us call a returning GCID system with k components *strictly linear* if its underlying simple undirected control graph has the edge set $\{(Ci, C(i+1))\} \mid i \in [1 \ldots k-1]\}$. Notice that this means that the corresponding directed control graph may contain arcs like $(Ci, C(i+1))$, $(C(i+1), Ci)$, as well as loops (Ci, Ci). Notice that the control graph of Π_1 from Example 3 is not strictly linear, although Π_1 is returning and the underlying undirected control graph is a path. Conversely, the two Examples 4 and 5 show strictly linear GCID systems, because in fact any returning GCID system with two components only is strictly linear.

Slightly abusing notation, the language class generateable by GCID systems of size at most σ is denoted by $\mathrm{GCID}(\sigma)$. In particular, for $k = 1$, $\mathrm{GCID}(1; n, i', i''; m, j', j'') = \mathrm{ID}(n, i', i''; m, j', j'')$. The language class generateable by strictly linear GCID systems of size σ is denoted by $\mathrm{GCID}_L(\sigma)$.

2.3. Stitching and overlaying

We consider a type-0 grammar G in Special Geffert Normal Form (SGNF) as described in Definition 1, carrying over the notation introduced there. So, G contains context-free rules of the three forms $X \to bY$, $X \to Yb$, $S' \to \lambda$ and the two non-context-free rules $AB \to \lambda$, $CD \to \lambda$. We first show how to simulate the non-context-free rules by GCID parts of certain sizes and then independently show how to simulate the context-free rules by GCID parts of certain other (possibly different) sizes. With these pieces in hand, we stitch together a simulation of non-context-free rules (say with size σ_{cs}) with a simulation of context-free rules (say with size σ_{cf}), to obtain a simulation of the SGNF rules by a GCID system of size $\max\{\sigma_{cs}, \sigma_{cf}\}$, where the maximum is applied for all respective parameters. The idea behind the stitching is as follows.

Suppose we simulate the context-free rule types $p : X \to bY$, $q : X \to Yb$ and $h : S' \to \lambda$ of SGNF by a returning GCID part with k components $(C1, C2, \ldots, Ck)$, with initial component $C1$. Similarly, suppose we simulate the non-context-free rules $f : AB \to \lambda$ and $g : CD \to \lambda$ of SGNF by a returning GCID part with j components $(C1, C(k+1), \ldots, C(k+j-1))$ and the same $C1$ as the initial component. Then we may simulate all the SGNF rules by a GCID system of $k + j$ components with $C1$ as start and final component by the following steps:

1. Start at $C1$.
2. Traverse through $C1, C2, \ldots, Ck$ to simulate the CF rules.
3. Come back to $C1$, as this is the final component.
4. Starting off from $C1$, traverse through $C1, C(k+1), \ldots, C(k+j-1)$ to simulate the non-CF rules.
5. Come back to $C1$, as this is the final component.

This will correctly simulate any type-0 grammar in SGNF, so that we can describe all of RE by such systems, assuming that the rules put together in $C1$ do not interfere with each other, but we aim at decreasing the number of components. A detailed note on this is given in Sec. 5. To achieve such savings in the number of components, we *overlay* the components of the two types of simulations. More formally, this *overlay* operation means the following. Let Π_1, Π_2, where $\Pi_j = (k_j, V_j, T, A_j, H_j, i_0, i_f, R_j)$ for $j = 1, 2$, be two GCID systems (or parts thereof), then the *overlay* of Π_1 and Π_2, written $\Pi_1 \cup \Pi_2$ for short, is given by the GCID system $\Pi = (k, V, T, A, H, i_0, i_f, R)$, where $k = \max\{k_1, k_2\}$, $V = V_1 \cup V_2$, $A = A_1 \cup A_2$, $R = R_1 \cup R_2$, and $H = H_1 \cup H_2$ (preserving the association of labels to rules in R_1 and R_2). Notice that this operation is defined only if the initial and final component and the terminal alphabet are identical in both Π_1 and Π_2, which is in particular true for returning systems. We will call the systems Π_1, Π_2 where $\Pi_1 \cup \Pi_2$ is defined, *overlayable*.

Observe that the overlay of two GCID systems basically corresponds to merging the rules associated to their components in a straightforward manner. By definition, if Π_1 and Π_2 are overlayable, then

$$L(\Pi_1) \cup L(\Pi_2) \subseteq L(\Pi_1 \cup \Pi_2)\,.$$

We will use this operation mainly to produce a complete GCID system simulating a type-0 grammar in SGNF, based on the parts simulating the non-context-free rules and the context-free rules. The set of axioms is then (always) a singleton set, consisting of the starting symbol S of the simulated grammar.

When providing simulation results, in our case, explaining how a GCID system Π can be constructed that simulates a given type-0 grammar G in SGNF, it is usually relatively easy to prove that $L(G) \subseteq L(\Pi)$ holds, as this proof just follows the intended simulation, but it is more complicated to prove that the reverse inclusion $L(G) \supseteq L(\Pi)$ is also true. This means that we have to show that there is no way to derive a word in $T^* \setminus L(G)$ by Π from any axiom. In other words, we have to rule out *malicious derivations*.

In order to be sure that no malicious derivations occur due to overlaying, we assume that the rules in P are labeled injectively with labels from the set $[1 \ldots |P|]$. We introduce a new *rule marker (symbol)* for each rule in order to tell which of the rules we are simulating. For clarity, we use the notation $p : \zeta \to \xi$ for a rule $\zeta \to \xi \in P$ with label p. Usually, the rule marker is introduced in $C1$. So, when going to the next component, this marker is present and can guard the application of rules. For instance, if $C2$ is the component to which the string is sent, a rule $(2, (\lambda, Y, p)_{ins}, 3)$ can be only applied if the rule marker p is present and this shall introduce the symbol Y that is associated with the p rule ($p : X \to bY$). Therefore, we call rules that use rule markers as a context or that delete rule markers also *guarded rules*. Moreover, we often use *derived rule markers*, mostly primed rule markers like p' or p'', that are introduced by guarded rules and can hence be also used to guard (other) rules. Using guarded rules makes it easy to argue that no malicious derivations are possible in the simulating ins–del system. This is in particular true if we can prove that no derivable string ever contains more than one rule marker. This useful property will be satisfied by all simulations using rule markers but the ones used in Theorems 28 and 30. In that case, we show, in addition, that any configuration $(w)_1$ with one rule marker in w cannot derive a terminal string w' in $C1$. We further extend the use of rule markers by sometimes requiring both a rule marker p and its primed version p' be present in the string sent to a particular component, by checking their presence in *double-guarded* rules. This proves in particular that two rules (introducing these two symbols) have been applied before. This idea is useful when insertions are context-free.

Another useful trick is that of a *dummy symbol* Δ that we sometimes introduce in order to be check the vicinity of symbols using deletions. To compare the work of the simulating ins–del system with that of the simulated type-0 grammar, we then use a homomorphism ϕ_Δ that keeps all symbols unchanged but Δ which is deleted. Sometimes, the rules related to dummy symbol help us to obtain a linear structure of the underlying control graph which otherwise is not easy to achieve.

Also, the separated treatment of the simulations of the context-free and non-context-free rules facilitates our arguments, as we can separately (and hence more clearly) argue why no malicious derivations are possible in the combined system produced by overlay.

3. Auxiliary results

It has been proved in [28] that an ins–del system $\mathrm{ID}(n, i', i''; 0, 0, 0)$ with no deletions cannot generate more that CS. It is clear that the same holds true for graph-controlled ins–del systems, as well (since without deletion rules, the rules are *monotone*). As a counterpart, we show below that if there is no insertion, then the (graph-controlled) ins–del system cannot generate more than the family of finite languages.

Theorem 6. $\mathrm{FIN} = \mathrm{GCID}(k; 0, i', i''; m, j', j'')$ *for* $k \geq 1; i', i'', j', j'', m \geq 0$.

Proof. Any finite language can be represented with the axiom set alone. Conversely, as long as the insertion string length fixed to be 0, rules can only shorten words in a derivation, so starting out from any finite language (the axiom set), we can only describe finite languages. □

To get some more interesting results, we assume in the following that the insertion and deletion lengths (n and m, respectively) are at least one.

Further, in order to simplify the proofs of some of our main results in the subsequent sections, the following observations are helpful.

Theorem 7. *For all non-negative integers* k, n, i', i'', m, j, j'', *we have that*

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

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

Corollary 8. *Let* $\mathcal{L}$ *be a language class that is closed under reversal. Then, for all non-negative integers* $k, n, i', i'', m, j', j''$, *we conclude that*

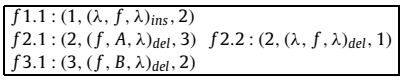

Fig. 1. Simulation of $f : AB \to \lambda$ by GCID rules of size $(3; 1, 0, 0; 1, 1, 0)$.

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

All the above results are also true for the family of insertion–deletion languages also, as ins–del grammar systems are special case of graph-controlled ins–del grammar systems when $k = 1$. A very important special case will be $\mathcal{L} = \mathrm{RE}$, as RE is closed under reversal and we are mostly interested in obtaining computational completeness results. To this end, the following proposition is also important to know.

Proposition 9. *If L can be generated by some GCID system, then $L \in \mathrm{RE}$.*

Proof. (Sketch) As the applicability of each rule of a GCID system can be easily checked by a Turing machine, it is straightforward but tedious to develop a Turing machine that, given the description of some GCID system, enumerates all words of the corresponding language L by dovetailing. □

Having the previous result in mind, from now on we only need to show how to simulate Turing machines, or equivalently type-0 grammars in SGNF, by using certain types of GCID systems in order to show that these systems characterize the class RE of recursively enumerable languages.

4. Simulation of the SGNF rules in pieces

In this section, we provide different simulations of (first) the non-context-free rules and (then) the context-free rules of a type-0 grammar in SGNF. We chose this sequence of presentation, as the simulation of the non-context-free rules tend to be easier than the simulations of the context-free ones.

4.1. Simulation of non-context-free rules

Recall that in a type-0 grammar in SGNF, the non-context-free rules are of the form $f : AB \to \lambda$ and $g : CD \to \lambda$, where $A, B, C, D \in N''$. In the following lemmas, we will only present the simulation of the rule $f : AB \to \lambda$ by GCID rules of the specified sizes. A GCID part Π of size σ_j that simulates these non-context-free rules is denoted by $\Pi_{cs}^{\sigma_j}$ and the size σ_j is sometimes also denoted as σ_{cs}^{j} in Sec. 5.1. The simulation of $g : CD \to \lambda$ is similar to the simulation of the f rule and hence omitted. For simplicity, we abbreviate $(\Rightarrow_f \cup \Rightarrow_g)$ as $\Rightarrow_{f,g}$. As all our simulations start by introducing a rule marker symbol in $C1$, we do not mention this explicitly in the following.

Lemma 10. *The non-context-free rules of a grammar G in SGNF can be simulated by a returning GCID part $\Pi = \Pi_{cs}^{\sigma_1}$ of size $\sigma_1 = (3; 1, 0, 0; 1, 1, 0)$.*

Let $w, w' \in (N \cup T)^*$. If $w \Rightarrow_{f,g} w'$ in G, then $(w)_1 \Rightarrow' (w')_1$ in Π. Moreover, if $(w)_1 \Rightarrow' (w')_1$ in Π, then $w \Rightarrow_{f,g}^* w'$.

Proof. The simulation of $f : AB \to \lambda$ is presented in Fig. 1.

We will see how the simulation works. Consider the string $\alpha AB\beta$ in $C1$. We introduce a marker f as a rule marker before A and move to $C2$. Now there is a choice of applying rule $f2.1$ or $f2.2$. In the latter case, we will delete the just inserted marker f and end up with $\alpha AB\beta$ in $C1$ (the starting point). Hence, we have to choose rule $f2.1$ eventually to move on. In this case, there is a unique sequence of rule applications in Π as follows.

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

There is only one little caveat here: It could be that we may choose to apply $f2.1$ instead of $f2.2$ in the last step. As this corresponds to first applying $f2.2$ and then immediately applying $f1.1$ again (re-inserting f in the same place), no harm can be done. Also, if $(w)_1 \Rightarrow (w')_1$, then by induction (and observing the previous derivation possibilities), this means that w' was obtained from w by repeatedly removing (contiguous) substrings AB or CD. In particular, observe that introducing f into a position such that AB is not immediately following renders further derivation steps infeasible, apart from $f2.2$ which simply undoes this choice of inserting f. Also, observe how using the rule markers f and g easily avoid any malicious derivations, as all rules in $C2$ and $C3$ are guarded. □

Remark 11. The control graph underlying the graph-controlled insertion–deletion rules from Lemma 10 is shown in Fig. 2. Its linear structure (a path on 3 vertices) is clearly visible. The labels of the edges refer to the rules as in Fig. 1. It is

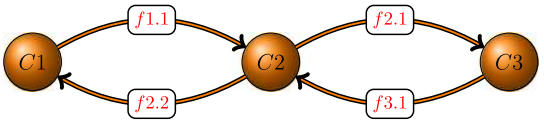

Fig. 2. Control graph corresponding to Lemma 10: Simulation of non-CF rules by GCID rules of size $(3; 1, 0, 0; 1, 1, 0)$.

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

Fig. 3. Simulation of $f : AB \to \lambda$ by GCID rules of size $(3; 1, 0, 0; 1, 0, 1)$.

$$\begin{array}{|ll|}\hline f1.1 : (1, (\lambda, f, A)_{ins}, 2) & \\ f2.1 : (2, (\lambda, A, \lambda)_{del}, 3) & f2.2 : (2, (\lambda, f'', \lambda)_{del}, 1) \\ f3.1 : (3, (B, f'', \lambda)_{ins}, 4) & f3.2 : (3, (\lambda, f', \lambda)_{del}, 2) \\ f4.1 : (4, (\lambda, B, \lambda)_{del}, 5) & f4.2 : (4, (\lambda, f, \lambda)_{del}, 3) \\ f5.1 : (5, (f, f', f'')_{ins}, 4) & \\ \hline \end{array}$$

Fig. 4. Simulation of $f : AB \to \lambda$ by GCID rules of size $(5; 1, 1, 1; 1, 0, 0)$.

understood that whenever an edge has label $fi.j$, then the label is read as $fi.j, gi.j$, since the simulation of g rules, though similar to the f rules, should also be taken care of. This convention is followed in all our control graphs. □

Lemma 12. *The non-context-free rules of a grammar G in SGNF can be simulated by a returning GCID part $\Pi = \Pi_{cs}^{\sigma_2}$ of size $\sigma_2 = (3; 1, 0, 0; 1, 0, 1)$.*

Let $w, w' \in (N \cup T)^*$. If $w \Rightarrow_{f,g} w'$ in G, then $(w)_1 \Rightarrow' (w')_1$ in Π. Moreover, if $(w)_1 \Rightarrow' (w')_1$ in Π, then $w \Rightarrow_{f,g}^* w'$.

Proof. The rules of the GCID system is symmetrical to the simulating rules given in Fig. 1. However, the rules are made explicit in Fig. 3. □

Again, the underlying control graph structure is isomorphic to the graph shown in Fig. 2 and hence is linear.

While all previous non-trivial simulations were done by rules that either introduced rule markers or were guarded, this is no longer true in the following simulation. It is relatively hard to get rid of any context in the insertions or deletions. Hence it seems to be necessary to allow for more components. We require these many components and (primed/double-primed) markers to prevent these non-context free deletion rules from interfering or being interfered by the simulation of the corresponding context-free rules of SGNF while overlaying. We will even use triple- and quadruple-primed markers in some constructions that follow.

Lemma 13. *The non-context-free rules of a grammar G in SGNF can be simulated by a returning GCID part $\Pi = \Pi_{cs}^{\sigma_3}$ of size $\sigma_3 = (5; 1, 1, 1; 1, 0, 0)$.*

Let $w, w' \in (N \cup T)^*$. Then $w \Rightarrow_{f,g} w'$ iff $(w)_1 \Rightarrow' (w')_1$ in Π.

Proof. The simulation of $f : AB \to \lambda$ is presented in Fig. 4. We now focus on explaining the simulation. $C1$ is the component where the simulation begins and ends. Assume that we need to process the string $\alpha AB\beta$. In $C1$, the marker f is introduced on the left of A and the string moves to $C2$. In $C2$, the rule $f2.2$ cannot be applied now as there is no double-primed marker f''. Thus, in $C2$, the rule $f2.1$ is applied which will delete an occurrence of A from the string. In $C3$, the rule $f3.2$ cannot be used now as there is no f' in the string, thus the rule $f3.1$ only can be applied which will introduce a marker f'' on the right of (some) B and the string moves to $C4$. In $C2$ and in $C4$ the marked A and B are (expected to be) deleted and the string moves to $C5$. If the correct occurrences of A and B are deleted, then the string will be of the form $\alpha ff''\beta$ and therefore, we can insert f' in between f and f'' using the rule $f5.1$. In other words, we can only apply $f5.1$ successfully if we had started the whole derivation like

$$(\alpha AB\beta)_1 \Rightarrow (\alpha fAB\beta)_2 \Rightarrow (\alpha fB\beta)_3 \Rightarrow (\alpha fBf''\beta)_4 \Rightarrow (\alpha ff''\beta)_5 \Rightarrow (\alpha ff'f''\beta)_4 .$$

Now, we remove the markers f, f' and f'' using the rules $f4.2$, $f3.2$ and $f.2.2$ in order and obtain the string $\alpha\beta$ in $C1$. More precisely, we get

$$(\alpha ff'f''\beta)_4 \Rightarrow (\alpha f'f''\beta)_3 \Rightarrow (\alpha f''\beta)_2 \Rightarrow (\alpha\beta)_1 .$$

Note that, if $f4.2$ is applied instead of $f4.1$ in the configuration $(\alpha fBf''\beta)_4$, then $(\alpha Bf''\beta)_3 \Rightarrow_{f3.1} (\alpha Bf''f''\beta)_4 \Rightarrow_{f4.1} (\alpha f''f''\beta)_5$ is enforced, but we cannot make any progress in $C5$, as the marker f is lost now and the derivation will be stuck in $C5$. If $f4.1$ is applied in configuration $(\alpha ff'f''\beta)_4$, then no rule in $C5$ is applicable and the derivation is stuck there again. Similarly, in configuration $(\alpha f'f''\beta)_3$, we could now wrongly apply $f3.1$ and move a string with one occurrence of

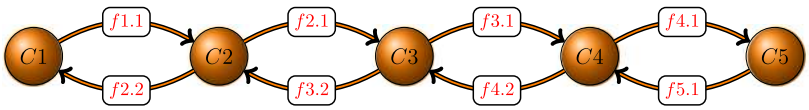

Fig. 5. Control graph corresponding to Lemma 13: Simulation of non-CF rules by GCID rules of size $(5; 1, 1, 1; 1, 0, 0)$.

$f1.1 : (1, (\lambda, f, \lambda)_{ins}, 2)$	
$f2.1 : (2, (\lambda, f', \lambda)_{ins}, 3)$	$f2.2 : (2, (\lambda, f'', \lambda)_{del}, 1)$
$f3.1 : (3, (f, B, f')_{del}, 4)$	$f3.2 : (3, (f'', f', \lambda)_{del}, 2)$
$f4.1 : (4, (\lambda, f'', \lambda)_{ins}, 5)$	$f4.2 : (4, (f'', f, f')_{del}, 3)$
$f5.1 : (5, (f'', A, f)_{del}, 4)$	

Fig. 6. Simulation of $f : AB \to \lambda$ by GCID rules of size $(5; 1, 0, 0; 1, 1, 1)$.

f' and two occurrences of f'' into $C4$, where some B has to be deleted using $f4.1$. The resulting string is moved into $C5$ and gets stuck there, as no occurrence of f is found. Finally, in configuration $(\alpha f''\beta)_2$, we might apply $f2.1$, followed by $f3.1$ and $f4.1$ (both enforced). As the resulting string that is sent to $C5$ contains no occurrence of f, the derivation is stuck again in $C5$. The crucial check that verifies the positions of the previously introduced markers f and f' is performed by an insertion operation in $C5$.

Thus, with the details provided above, we can see that AB is deleted correctly and the string $\alpha\beta$ is sent back to $C1$.

We introduce similar rules to simulate $g : CD \to \lambda$ with size $(5; 1, 1, 1; 1, 0, 0)$. As in $C1$, always a rule marker is introduced, and this is needed to process $C5$, there is no way in which both simulations can interfere. In particular, if for example $g4.1$ was used instead of $f4.1$ in a derivation that otherwise simulates the f-rule, then $f5.1$ is not applicable, so that the derivation is stuck in $C5$. □

Remark 14. The control graph underlying the graph-controlled insertion–deletion rules from Lemma 13 is shown in Fig. 5. Its linear structure (a path on 5 vertices) is clearly visible. The labels of the edges refer to the rules as in Fig. 4. □

Although this is not clear at this point of the paper, there are situations where a more complicated, seemingly weaker simulation seems to be necessary. The specific property that we later need is that we have to guarantee that both a rule from $C1$ (actually, in this case, there is only one rule per simulating rule in $C1$) and other specific marker-introducing rules from $C2$ have to be introduced before being able to successfully apply any rule from $C3$, as can be seen in the next simulation. To underline these properties, we say that the rules in $C3$ are *double-guarded*. In other words, a rule is double-guarded when two markers are contexts in the rule.

Lemma 15. *The non-context-free rules of a grammar G in SGNF can be simulated by a returning GCID part $\Pi = \Pi_{cs}^{\sigma_4}$ of size $\sigma_4 = (5; 1, 0, 0; 1, 1, 1)$.*

Let $w, w' \in w \in (N \cup T)^*$. Then $w \Rightarrow_{f,g} w'$ iff $(w)_1 \Rightarrow' (w')_1$ in Π.

Proof. The simulation of $f : AB \to \lambda$ is presented in Fig. 6 which begins at $C1$. Consider the string $\alpha AB\beta$ at $C1$. The intended sequence of simulating derivations is shown below.

$$(\alpha AB\beta)_1 \Rightarrow_{f1.1} (\alpha Af B\beta)_2 \Rightarrow_{f2.1} (\alpha Af Bf'\beta)_3 \Rightarrow_{f3.1}$$
$$(\alpha Aff'\beta)_4 \Rightarrow_{f4.1} (\alpha f''Aff'\beta)_3 \Rightarrow_{f5.1} (\alpha f''ff'\beta)_4 \Rightarrow_{f4.2}$$
$$(\alpha f''f'\beta)_3 \Rightarrow_{f3.2} (\alpha f''\beta)_2 \Rightarrow_{f2.2} (\alpha\beta)_1$$

Let us now discuss the situation $(w)_1 \Rightarrow' (w')_1$ in Π for $w \in (N \cup T)^*$. On applying the rule $f1.1$, we introduce a marker f into the string and move the resulting string w_1 to $C2$. The rule $f2.2$ cannot be applied, since there is no f'' in w_1. Hence, on applying $f2.1$, another marker f' is introduced and the resulting string w_2 is moved to $C3$. In $C3$, due to the absence of double-primed markers in w and hence in w_2, the only applicable rule is $f3.1$ which takes care of the positions of the markers f and f': f and f' are to the left and right of some occurrence of B, respectively. On deleting the sandwiched B, the resulting string w_3 is moved to $C4$. Let us describe this derivation (enforced so far) more formally now. According to what we said above, $w = \alpha' B\beta'$ for some $\alpha'\beta' \in (N \cup T)^*$.

$$(\underbrace{\alpha' B\beta'}_{w})_1 \Rightarrow_{f1.1} (\underbrace{\alpha' f B\beta'}_{w_1})_2 \Rightarrow_{f2.1} (\underbrace{\alpha' f Bf'\beta'}_{w_2})_3 \Rightarrow_{f3.1} (\underbrace{\alpha' ff'\beta'}_{w_3})_4$$

On applying the only applicable rule $f4.1$, a new marker f'' is introduced randomly and the resulting string w_4 is moved to $C5$. One may note that at this point there are three markers f, f', f'' in the string such that f' is placed after f and f'' is randomly placed. Now in $C5$, $f5.1$ can be applied only if the previously introduced f'' was placed to the left of some A, which is now deleted. Moreover, this A that is deleted was situated left to the marker f, so that this produces

$p1.1 : (1, (\lambda, p, X)_{ins}, 2)$		
$p2.1 : (2, (p, X, \lambda)_{del}, 3)$	$p2.2 : (2, (p, b, p')_{ins}, 2)$	$p2.3 : (2, (b, Y, p')_{ins}, 2)$
$p2.4 : (2, (\lambda, p, \lambda)_{del}, 3)$	$p2.5 : (2, (Y, p', \lambda)_{del}, 2)$	$p2.6 : (2, (Y, p'', \lambda)_{del}, 1)$
$p3.1 : (3, (p, p', \lambda)_{ins}, 2)$	$p3.2 : (3, (p', p'', \lambda)_{ins}, 2)$	

(a) Simulation of $p : X \to bY$

$q1.1 : (1, (\lambda, q, X)_{ins}, 2)$		
$q2.1 : (2, (q, X, \lambda)_{del}, 3)$	$q2.2 : (2, (q, Y, q')_{ins}, 2)$	$q2.3 : (2, (Y, b, q')_{ins}, 2)$
$q2.4 : (2, (\lambda, q, \lambda)_{del}, 3)$	$q2.5 : (2, (b, q', \lambda)_{del}, 2)$	$q2.6 : (2, (b, q'', \lambda)_{del}, 1)$
$q3.1 : (3, (q, q', \lambda)_{ins}, 2)$	$q3.2 : (3, (q', q'', \lambda)_{ins}, 2)$	

(b) Simulation of $q : X \to Yb$

Fig. 7. Simulation of context-free rules of SGNF by GCID rules of size $(3; 1, 1, 1; 1, 1, 0)$.

the string $w_6 = \alpha'' f'' f f' \beta'$, where $\alpha' = \alpha'' A$ which is moved to $C4$. If we now applied $f4.1$ again, then the derivation is stuck in $C5$, as f'' is immediately to the left of f. So, we have to apply $f4.2$ to continue. This produces the string $w_7 = \alpha'' f'' f' \beta'$ that is moved to $C3$. As there is no marker f anymore in the string, $f3.2$ has to be applied, producing the configuration $(\alpha'' f'' \beta')_2$. In $C2$, only $f2.2$ is applicable, leading us to $(\alpha'' \beta')_1$. In other words, we showed that inevitably $(w)_1 = (\alpha'' AB \beta')_1 \Rightarrow' (\alpha'' \beta')_1$, which corresponds to a rule application like $AB \to \lambda$. A similar reasoning applies to the simulation of $CD \to \lambda$. Finally, as rule applications are guarded, there is no danger of messing up derivations that simulate rules f and g. □

4.2. Simulation of context-free rules

Recall that in a type-0 grammar in SGNF, there are three forms of context-free rules: (i) $p : X \to bY$, (ii) $q : X \to Yb$ and (iii) $h : S' \to \lambda$, where $S', X, Y \in N'$, $X \neq Y$ and $b \in T \cup N''$. One can simulate the rule $S' \to \lambda$ directly by $(1, (\lambda, S', \lambda)_{del}, 1)$. So, (iii) will no longer be explicitly mentioned in the following. In the following lemmas, we will therefore only simulate the p and q rules by GCID rules of the specified sizes. A GCID part Π of size σ_i that simulates these context-free rules is denoted by $\Pi_{cf}^{\sigma_i'}$ and the size σ_i' is sometimes also denoted as σ_{cf}^i in Sec. 5.1. It is important to note that the initial and final component in our GCID parts is again $C1$. For simplicity, we write $\Rightarrow_{cf}$ to denote a derivation step of G due to one of the context-free rules. Apart from $(1, (\lambda, S', \lambda)_{del}, 1)$, all rules in $C1$ introduce a rule marker, which will therefore not be mentioned henceforth.

Lemma 16. *The context-free rules of a type-0 grammar G in SGNF can be simulated by a returning GCID part $\Pi = \Pi_{cf}^{\sigma_1'}$ of size $\sigma_1' = (3; 1, 1, 1; 1, 1, 0)$.*

Let $w \in (N'' \cup T)^* N' (N'' \cup T)^*$ and $w' \in (N'' \cup T)^* (N' \cup \{\lambda\}) (N'' \cup T)^*$. Then, $w \Rightarrow_{cf} w'$ iff $(w)_1 \Rightarrow' (w')_1$ in Π.

Proof. We simulate the context-free rules of a type-0 grammar G in SGNF by the rules listed in Figs. 7(a) and 7(b).

We now explain how the simulation of the rules p and q work and why no other malicious derivations are possible.

Simulation of $p : X \to bY$: Consider the string $w = \alpha X \beta$ in $C1$, with $\alpha, \beta \in (N'' \cup T)^*$ and $X \in N'$ in $C1$. By $p1.1$, a p is introduced to the left of X and in $C2$, the X is deleted. In $C2$, we can see that no other rule can be applied except $p2.4$ and if $p2.4$ is applied, then in $C3$, no rules can be applied as no marker is present in the string. Thus, after applying $p2.1$, in $C3$ the rule $p3.1$ is applied which will introduce a p' after p, sending the string $\alpha p p' \beta$ to $C2$. If $p2.4$ is applied before applying $p2.2$ and $p2.3$, the string will be stuck in $C2$ after applying $p3.2$ in $C3$, as Y is not yet introduced. Thus, in $C2$, the rules $p2.2$ and $p2.3$ are applied before applying $p2.4$, which will introduce b and Y in the string and result to $(\alpha p b Y p' \beta)_2$. As the insertion rules $p2.2$ and $p2.3$ have double-sided contexts, they cannot be applied again even if the string remains to be in $C2$. In $C2$, p is deleted by $p2.4$ and the string is sent to $C3$ where p'' is introduced on the right of p' by $p3.2$ and the resultant string $\alpha b Y p' p'' \beta$ is sent to $C2$. In $C2$, in order to apply the rule $p2.6$, first the rule $p2.5$ is applied which will delete p' and then p'' is deleted by $p2.6$.

With the details provided, we can see that the rule $p : X \to bY$ is simulated and the string $\alpha b Y \beta$ is obtained in $C1$. The simulation is given by the following sequence of rule applications.

$$(\alpha X \beta)_1 \overset{\Rightarrow_{p1.1}}{\underset{\Leftarrow_{p2.4}}{}} (\alpha p X \beta)_2 \Rightarrow_{p2.1} (\alpha p \beta)_3 \Rightarrow_{p3.1} (\alpha p p' \beta)_2 \Rightarrow_{p2.2}$$
$$(\alpha p b p' \beta)_2 \Rightarrow_{p2.3} (\alpha p b Y p' \beta)_2 \Rightarrow_{p2.4} (\alpha b Y p' \beta)_3 \Rightarrow_{p3.2}$$
$$(\alpha b Y p' p'' \beta)_2 \Rightarrow_{p2.5} (\alpha b Y p'' \beta)_2 \Rightarrow_{p2.6} (\alpha b Y \beta)_1$$

Simulation of $q : X \to Yb$: The simulation is very similar to the p-type rule simulation and hence, its explicit explanation is omitted. □

Remark 17. The control graph underlying the GCID rules from Lemma 16 is shown in Fig. 8. Its linear structure is evident. The labels of the edges refer to the rules as in Figs. 7(a) and 7(b). The labels of the edges refer to the rules as in Fig. 7(a).

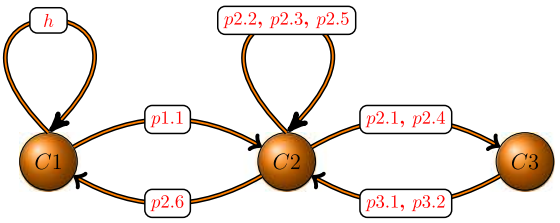

Fig. 8. A linear control graph corresponding to Lemma 16: Simulating CF rules by GCID rules of size $(3; 1, 1, 1; 1, 1, 0)$.

$p1.1: (1, (X, p, \lambda)_{ins}, 3)$ $p1.2: (1, (p, p', \lambda)_{ins}, 2)$ $p1.3: (1, (p', Y, \lambda)_{ins}, 2)$
$p2.1: (2, (p, b, \lambda)_{ins}, 3)$ $p2.2: (2, (\lambda, p', \lambda)_{del}, 1)$
$p3.1: (3, (\lambda, X, \lambda)_{del}, 1)$ $p3.2: (3, (\lambda, p, \lambda)_{del}, 1)$

(a) Simulation of $p : X \to bY$

$q1.1: (1, (X, q, \lambda)_{ins}, 3)$ $q1.2: (1, (q, b, \lambda)_{ins}, 2)$ $q1.3: (1, (q', Y, \lambda)_{ins}, 2)$
$q2.1: (2, (q, q', \lambda)_{ins}, 3)$ $q2.2: (2, (\lambda, q', \lambda)_{del}, 1)$
$q3.1: (3, (\lambda, X, \lambda)_{del}, 1)$ $q3.2: (3, (\lambda, q, \lambda)_{del}, 1)$

(b) Simulation of $q : X \to Yb$

Fig. 9. Simulation of context-free rules of SGNF by GCID rules of size $(3; 1, 1, 0; 1, 0, 0)$.

It is understood that whenever an edge has label $pi.j$, then the label is read as $pi.j, qi.j$, since the simulation of q rules, though similar to the p rules, should also be taken care of. This convention is followed in all control graphs. □

In the following lemma, we reduce the size of the simulating GCID part from $(3; 1, 1, 1; 1, 1, 0)$ or $(3; 1, 1, 0; 1, 1, 1)$ to $(3; 1, 1, 0; 1, 0, 0)$, however, compromising the linear structure of the underlying control graph.

Lemma 18. *The context-free rules of a type-0 grammar G in SGNF can be simulated by a returning GCID part $\Pi = \Pi_{cf}^{\sigma_3'}$ of size $\sigma_3' = (3; 1, 1, 0; 1, 0, 0)$.*

If $w \Rightarrow_{cf} w'$, then $(w)_1 \Rightarrow'_* (w')_1$ in Π.

Ignoring the context-free deletion rule, if $(w)_1 \Rightarrow' (w')_1 \Rightarrow' (w'')_1 \Rightarrow' (w''')_1$ in Π for some $w \in (N'' \cup T)^* N' (N'' \cup T)^*$, then $w \Rightarrow_{cf} w'''$. Moreover, then w', w'' do not contain any occurrence of N', but a (derived) rule marker instead.

Proof. The simulations of $p : X \to bY$ and $q : X \to Yb$ are presented in Figs. 9(a) and 9(b), respectively. We now focus on explaining the simulations.

Simulation of $p : X \to bY$: Consider the string $\alpha X \beta$ in $C1$. On applying rule $p1.1$, we insert p after X and get $\alpha X p \beta$ in $C3$. At this point, we have a choice of applying rule $p3.1$ or $p3.2$. In the latter case, the marker p will be deleted and we move back to the starting point. Hence we have to use rule $p3.1$ eventually to proceed. In this case, X is deleted and we move to $C1$ with $\alpha p \beta$, where p' is inserted after p and the string moves to $C2$ with $\alpha p p' \beta$. In $C2$, we can apply the rules $p2.1$ or $p2.2$. On applying $p2.2$, p' is deleted and the string $\alpha p \beta$ will be in $C1$ and we are back to the previous step. This forces us to eventually apply the rule $p2.1$. With these arguments, we simulate the rule $X \to bY$ as follows:

$$(\alpha X \beta)_1 \overset{\Rightarrow_{p1.1}}{\Leftarrow_{p3.2}} (\alpha X p \beta)_3 \Rightarrow_{p3.1} (\alpha p \beta)_1 \overset{\Rightarrow_{p1.2}}{\Leftarrow_{p2.2}} (\alpha p p' \beta)_2 \Rightarrow_{p2.1} (\alpha p b p' \beta)_3$$
$$\Rightarrow_{p3.2} (\alpha b p' \beta)_1 \Rightarrow_{p1.3} (\alpha b p' Y \beta)_2 \Rightarrow_{p2.2} (\alpha b Y \beta)_1.$$

Simulation of $q : X \to Yb$: This is done very similarly to the case of p-type rules. We only show the intended derivation below.

$$(\alpha X \beta)_1 \overset{\Rightarrow_{q1.1}}{\Leftarrow_{q3.2}} (\alpha X q \beta)_3 \Rightarrow_{q3.1} (\alpha q \beta)_1 \Rightarrow_{q1.2} (\alpha q b \beta)_2 \Rightarrow_{q2.1} (\alpha q q' b \beta)_3$$
$$\Rightarrow_{q3.2} (\alpha q' b \beta)_1 \Rightarrow_{q1.3} (\alpha q' Y b \beta)_2 \Rightarrow_{q2.2} (\alpha Y b \beta)_1.$$

It might be a tempting idea to have a derivation that employs rules from the set of p and q rules in a mixed fashion. But, once the rule marker is introduced in $C1$ next to its corresponding N' symbol, then this N' symbol gets deleted, so that no other simulation can start in $C1$. In fact, any occurrence of N' is introduced only in $C1$ again, so that no interference of different simulations is possible. Our discussion also shows the remaining statements of the lemma. □

Remark 19. The control graph of the preceding simulation is displayed in Fig. 10. The graph is non-linear due to the arc from $C3$ to $C1$. □

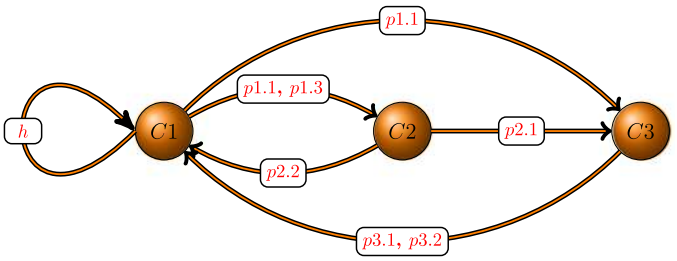

Fig. 10. Non-linear control graph corresponding to Lemma 18; Simulation of context-free rules by GCID rules of size (3; 1, 1, 0; 1, 0, 0).

$p1.1 : (1, (\lambda, p, X)_{ins}, 2)$	
$p2.1 : (2, (X, p', \lambda)_{ins}, 3)$	$p2.2 : (2, (\lambda, p, \lambda)_{del}, 1)$
$p3.1 : (3, (\lambda, X, \lambda)_{del}, 4)$	$p3.2 : (3, (\lambda, p', \lambda)_{del}, 2)$
$p4.1 : (4, (p, b, p')_{ins}, 4)$	$p4.2 : (4, (b, Y, p')_{ins}, 3)$

(a) Simulation of $p : X \to bY$

$q1.1 : (1, (\lambda, q, X)_{ins}, 2)$	
$q2.1 : (2, (X, q', \lambda)_{ins}, 3)$	$q2.2 : (2, (\lambda, q, \lambda)_{del}, 1)$
$q3.1 : (3, (\lambda, X, \lambda)_{del}, 4)$	$q3.2 : (3, (\lambda, q', \lambda)_{del}, 2)$
$q4.1 : (4, (q, Y, q')_{ins}, 4)$	$q4.2 : (4, (Y, b, q')_{ins}, 3)$

(b) Simulation of $q : X \to Yb$

Fig. 11. Simulation of context-free rules of SGNF by GCID rules of size (4; 1, 1, 1; 1, 0, 0).

In the following lemma, it seems to be necessary to allow for one more component, since it is relatively hard to get rid of any context in the deletions.

Lemma 20. *The context-free rules of a type-0 grammar G in SGNF can be simulated by a returning GCID part $\Pi = \Pi_{cf}^{\sigma_4'}$ of size $\sigma_4' = (4; 1, 1, 1; 1, 0, 0)$.*

Let $w \in (N'' \cup T)^* N' (N'' \cup T)^*$, $w' \in (N'' \cup T)^* (N' \cup \{\lambda\}) (N'' \cup T)^*$.

Then, $w \Rightarrow_{cf} w'$ iff $(w)_1 \Rightarrow' (w')_1$ in Π.

Proof. The rules $p : X \to bY$ and $q : X \to Yb$ are simulated by the rules shown in Figs. 11(a) and 11(b). We now explain the simulation.

Simulation of $p : X \to bY$: Consider the string $w = \alpha X \beta$ in $C1$. Applying the rule $p1.1$ will introduce p to the left of X and the resultant string is sent to $C2$. In $C2$, both the rules are possible to apply and applying $p2.2$ will delete the previously introduced p and thus the original string w is sent back to $C1$. Hence, the rule $p2.1$ should be applied at some point of time which will introduce p'; thus the string $\alpha p X p' \beta$ is sent to $C3$. In $C3$, if the rule $p3.2$ is applied, then the string will have no change again as the introduced marker p' is deleted and the string is sent back to $C2$. Thus, X will be eventually deleted by the rule $p3.1$ and the resultant string $\alpha p p' \beta$ is sent to $C4$. In $C4$, the only applicable rule is $p4.1$, as to apply $p4.2$, there should be a nonterminal from N', in particular, $Y \in N'$ and the corresponding marker p' should be present in the string. The application of the rule $p4.1$ will introduce a b in between p, p' and the resultant string $\alpha p b p' \beta$ shall remain in $C4$ and the rule cannot be applied again, because p is no longer immediately preceding p'. Now, the rule $p4.2$ can be applied, which will introduce a Y in between b, p' and the string $\alpha p b Y p' \beta$ is sent to $C3$, where p' is deleted using the rule $p3.2$. In $C3$, if $r3.1$ is applied, for some other rule like $r : Y \to Zb'$, $Z \in N'$, $b' \in N'' \cup T$, then the derivation is now oscillating between $C4$ and $C3$, because the substring pp' is no longer available, so that $p4.1$ is not applicable, but $p4.2$ is, which would re-introduce the N'-symbol Y that got previously deleted by $r3.1$.

So, in $C3$ only the rule $p3.2$ should finally be applied which will delete p' and the resultant string $\alpha Y b p' \beta$ is sent to $C2$. In $C2$, the marker p is meant to be deleted by the rule $p2.2$ and the resultant string $\alpha Y b \beta$ is sent to $C1$. Should we, instead, apply $r2.1$, for some other rule like $r : Y \to Zb'$, $Z \in N', b' \in N'' \cup T$, then the string $\alpha p Y r' b \beta$ would be sent to $C3$; moving now $\alpha p Y b \beta$ back to $C2$ (applying $r3.2$) would not change the flow of our argument. However, sending $\alpha p r' b \beta$ to $C4$ (applying $r3.1$) will make the derivation stuck in $C4$.

Thus, the rule $p : X \to Yb$ is simulated correctly and the intended derivation is given below.

$$(\alpha X \beta)_1 \Rightarrow_{p1.1} (\alpha p X \beta)_2 \Rightarrow_{p2.1} (\alpha p X p' \beta)_3 \Rightarrow_{p3.1} (\alpha p p' \beta)_4 \Rightarrow_{p4.1} (\alpha p b p' \beta)_4$$
$$\Rightarrow_{p4.2} (\alpha p b Y p' \beta)_3 \Rightarrow_{p3.2} (\alpha p b Y \beta)_2 \Rightarrow_{p2.2} (\alpha b Y \beta)_1.$$

Simulation of $q : X \to Yb$: This is very similar to the simulation of the p-type rule and we omit explanatory details, except one detail. When arriving in the configuration $(\alpha q Y b q' \beta)_3$, passing into $C4$ by some rule deleting Y will immediately get stuck.

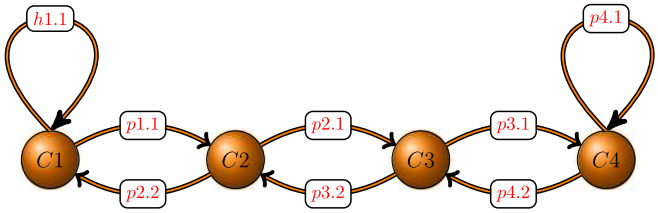

Fig. 12. A control graph corresponding to Lemma 20: Simulation of CF rules by GCID rules of size $(4; 1, 1, 1; 1, 0, 0)$.

$p1.1 : (1, (\lambda, p, \lambda)_{ins}, 2)$	$p1.2 : (1, (\lambda, \Delta, \lambda)_{ins}, 1)$
$p2.1 : (2, (\lambda, p', \lambda)_{ins}, 3)$	$p2.2 : (2, (\lambda, p', b)_{del}, 1)$
$p3.1 : (3, (p', X, p)_{del}, 4)$	$p3.2 : (3, (b, p, Y)_{del}, 2)$
$p4.1 : (4, (\lambda, Y, \lambda)_{ins}, 5)$	$p4.2 : (4, (\lambda, b, \lambda)_{ins}, 3)$
$p5.1 : (5, (p, \Delta, Y)_{del}, 4)$	

(a) Simulation of $p : X \to bY$

$q1.1 : (1, (\lambda, q, \lambda)_{ins}, 2)$	$q1.2 : (1, (\lambda, \Delta, \lambda)_{ins}, 1)$
$q2.1 : (2, (\lambda, q', \lambda)_{ins}, 3)$	$q2.2 : (2, (b, q', \lambda)_{del}, 1)$
$q3.1 : (3, (q, X, q')_{del}, 4)$	$q3.2 : (3, (Y, q, b)_{del}, 2)$
$q4.1 : (4, (\lambda, Y, \lambda)_{ins}, 5)$	$q4.2 : (4, (\lambda, b, \lambda)_{ins}, 3)$
$q5.1 : (5, (Y, \Delta, q)_{del}, 4)$	

(b) Simulation of $q : X \to Yb$

Fig. 13. Simulation of context-free rules of SGNF by GCID rules of size $(5; 1, 0, 0; 1, 1, 1)$.

Finally, observe that due to the introduction of different rule markers in $C1$, no confusion between the two kinds of rule simulations is possible. More precisely, consider two rules of the form $p : X \to bY$ and $q : X' \to Y'b'$ with $X' = X$. By symmetry, we will only discuss about starting the derivation with $p1.1$. Arguments about starting the derivation with $q1.1$ follow similarly. Consider a string $w = \alpha X \beta$. Applying $p1.1$ on w, we get $w_1 = \alpha p X \beta$. Deviating from the usual application of $p2.1$ or $p2.2$, we apply $q2.1$ on w_1 which yields $w_2 = \alpha p X q' \beta$ at $C3$. At this point, there is a choice of applying either $p3.1 = q3.1$ or $q3.2$ on w_2. In the former case, the nonterminal X is deleted and the resultant string $w_3 = \alpha p q' \beta$ enters $C4$. No rule of $C4$ is applicable due to the absence of a nonterminal and of p', q. In the latter case, the marker q' is deleted in w_2 which yields $\alpha p X \beta = w_1$. Hence, mixed derivations either get stuck up at $C4$ or get back to some string present in the intended derivation sequence. □

Remark 21. The control graph of the preceding simulation is shown in Fig. 12. Its corresponding underlying undirected simple graph is a path on four nodes and hence linear. □

We now present a symmetric simulation of context-free insertion however with one more extra component (of size $(5; 1, 0, 0; 1, 1, 1)$). It is a bit tricky in the sense that we make use of a dummy symbol Δ, as discussed in Sec. 2.

Lemma 22. *The context-free rules of a type-0 grammar G in SGNF can be simulated by a returning GCID part $\Pi = \Pi_{cf}^{\sigma_5'}$ of size $\sigma_5' = (5; 1, 0, 0; 1, 1, 1)$.*

If $w \Rightarrow_{cf} w'$, then $(w)_1 \Rightarrow' (w')_1$ in Π.

If $(w)_1 \Rightarrow' (w')_1 \Rightarrow' (w'')_1$ in Π for some $w \in (N'' \cup T)^* N' (N'' \cup T)^*$ with $\left|\{w, w', w''\}\right| = 3$, then $w = \phi_\Delta(w')$, and $w \Rightarrow_{cf} w''$ or $w = \phi_\Delta(w'')$.

Proof. The simulation of $p : X \to bY$ and $q : X \to Yb$ is presented in Figs. 13(a) and 13(b), respectively.

We now focus on explaining the simulations.

Simulation of the rule $p : X \to Yb$: Consider the string $w = \alpha X \beta$ in $C1$. In the beginning, we apply $p1.1$ or $p1.2$. Assume that $p1.2$ is applied some times which will randomly introduce some Δ in the string. Later, we will see that the most useful thing is to introduce one Δ immediately after X. Then, the rule $p1.1$ is applied which will introduce a p anywhere in the string and the string is then moved to $C2$. The rule $p2.2$ cannot be applied as there is no p' present. Thus, in $C2$, an application of the rule $p2.1$ will introduce a p' in the string and the string is moved to $C3$. As we will later understand, the string that can survive future derivation steps is now $w' = \alpha p' X p \Delta \beta$, with possibly more occurrences of Δ inserted into the string.

In $C3$, an application of $p3.2$ on w' would possibly delete p (think of having introduced some r in $C1$ corresponding to a rule with left-hand side Y, this marker r can now be deleted, or, alternatively, think of having started with $v = \alpha' Y \beta'$ instead of w); now, applying $p2.2$ would bring us back to our origin in $C1$ with the same string we started with, possibly decorated with some additional occurrences of Δ. However, if we now apply $p2.1$ (or any other such rule $r2.1$), then we are stuck in $C3$, as we have now two occurrences of primed rule labels, but no occurrence of an un-primed rule label, so that neither $p3.1$ nor $p3.2$ are applicable.

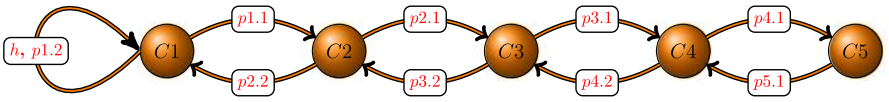

Fig. 14. Control graph corresponding to Lemma 22: Simulating the CF rules by GCID rules of size $(5; 1, 0, 0; 1, 1, 1)$.

This results in the application of $p3.1$ on w' which ensures that the previously introduced p and p' are actually placed to the side of X and the $X \in N'$ is deleted. With the resultant string $\alpha p' p \Delta \beta$ (following previous discussions), in $C4$, if $p4.2$ is applied before the rule $p4.1$ is applied, then in $C3$, no rule will be applicable, as there is no nonterminal in the string. Thus, in $C4$ with the string $\alpha p' p \Delta \beta$, the Y corresponding to the p-rule is introduced by $p4.1$. In $C5$, the rule $p5.1$ can be applied only if the dummy symbol Δ has been introduced by $p1.2$ and is placed after p.

The deletion of Δ ensures that the previously introduced Y is placed after $p\Delta$. The deletion of Δ will bring the string $\alpha p' p Y \beta$ to $C4$ and if $p4.1$ is applied again, then the string will be stuck in $C5$ as after p, the nonterminal Y is present and Δ cannot be found to delete it.

This means that if Δ has been introduced, say, twice by $p1.2$, then this additional Δ is placed somewhere else and not between p and Y. Thus, in $C4$, $p4.2$ must be applied to the configuration $(\alpha p' p Y \beta)_4$. This will introduce a b and the rule $p3.2$ ensures that the previously introduced b corresponds to this p rule and is placed after the primed label p', preceding the rule marker p. This will enable to delete p and in $C2$, p' is also deleted by $p2.2$.

It is possible that we can have several copies of the dummy symbol Δ in the string which can be removed later when calling some rule $r5.1$ again, simulating a context-free rule r, or the derivation will not terminate, which is of course innocuous, as it would have been possible to generate the correct number of Δ in $C1$ in another derivation.

Simulation of $q : X \to Yb$: The simulation of this rule is similar to the simulation of the p-type rule explained above, and hence we are not discussing it here. We only provide the intended simulation:

$$\begin{aligned}&(\alpha \Delta X \beta)_1 \Rightarrow_{q1.1} (\alpha \Delta q X \beta)_2 \Rightarrow_{q2.1} (\alpha \Delta q X q' \beta)_3\\ &\Rightarrow_{q3.1} (\alpha \Delta q q' \beta)_4 \Rightarrow_{p4.1} (\alpha Y \Delta q q' \beta)_5 \Rightarrow_{p5.1} (\alpha Y q q' \beta)_4\\ &\Rightarrow_{q4.2} (\alpha Y q b q' \beta)_3 \Rightarrow_{q3.2} (\alpha Y b q' \beta)_2 \Rightarrow_{q2.2} (\alpha Y b \beta)_1\end{aligned}$$

There is also no way to start a derivation with a p-type rule simulation and then switch to the q-type rule simulation, or vice versa, in any terminating derivation of Π, as the rules in $C3$ that allow to pass to $C4$ are double-guarded and the correct introductions of Y and b (by the context-free insertions in $C4$) are later checked by guarded deletions.

Finally consider the situation $(w)_1 \Rightarrow' (w')_1 \Rightarrow' (w'')_1$ in Π for some $w \in (N'' \cup T)^* N' (N'' \cup T)^*$ with $\left|\{w, w', w''\}\right| = 3$. By our previous considerations, we cannot first apply some rule $r1.1$, as such a derivation is stuck, and we cannot move a string w' into $C1$ again. So, we need to apply some rule $r1.2$ once or twice in the beginning. If we apply it once, then w' and w only differ by one occurrence of Δ in w', but $\phi_\Delta(w') = w$. If we apply $r1.2$ a second time, then $\phi_\Delta(w'') = w$ holds. Otherwise, we apply $r1.1$ to w', follow the simulation as described above, in particular deleting the only occurrence of Δ with $r5.1$, so that w'' satisfies $w \Rightarrow_{cf} w''$. □

Remark 23. The control graph of the simulation rules given in Figs. 13(a) and 13(b) is shown in Fig. 14. □

Remark 24. A GCID system of size as specified in the previous Lemmas (16 through 22) simulates the rules $p : X \to bY$, $q : X \to Yb$ and $h : S' \to \lambda$ where $X,\ Y,\ S' \in N'$ and $b \in T \cup N''$. In particular, if $b \in T$, then the GCID system will simulate linear rules. So, $\mathrm{LIN} \subsetneq \mathrm{GCID}(\alpha)$ where α is one of the sizes specified in Lemmas 16 through 22. The strictness of the containment follows from Example 5. □

5. Computational completeness of GCID systems

In this section, we prove several computational completeness results for graph-controlled ins–del systems of size $(k; 1, i', i''; 1, j', j'')$ for all values in the range $i', i'', j', j'' \in \{0, 1\}$, with $k \in \{3, 4, 5\}$ components. S. Ivanov and S. Verlan [13] have conjectured that such GCID systems with only two components are no better than ins–del systems to characterize RE, as two components do not provide sufficient control. We were also unable to find any computational completeness results for such GCID systems with two components only. However, it would be very interesting to prove the mentioned conjecture, as it would automatically imply that our results for 3 components are optimal.

We achieve the computational completeness results of GCID systems with 3 to 5 components by stitching together the ingredients that we explained in Sec. 4. In order to be more parsimonious regarding the overall number of components, we will *overlay* the components defined above for the non-context-free and for the context-free rules. At this point, the reader may refer to Sec. 2.3 to recall the concept of overlaying. Further, the strict linearity of the control graph is preserved by overlaying. Formally, this yields the following proposition.

Proposition 25. *Let Π_1 and Π_2 be two strictly linear overlayable GCID systems. Then, $\Pi_1 \cup \Pi_2$ is also strictly linear.*

Table 1
The stitching components (simulations of context-free and non-context-free rules) and resulting sizes of GCID systems yielding RE.

σ_{cf}^i	GCID(σ_{cf}^i) can simulate CF rules of SGNF:	σ_{cs}^j	GCID(σ_{cs}^j) can simulate non-CF rules of SGNF:	Size δ_{ij} of resulting GCID system, where $\delta_{ij} := \sigma_{cf}^i \oplus \sigma_{cs}^j$	Why does RE equal GCID(δ_{ij})?	Control graph is linear?
σ_{cf}^1	Lemma 16	σ_{cs}^1	Lemma 10	(3; 1, 1, 1; 1, 1, 0)	Theorem 27	Yes
σ_{cf}^2	Lemma 18	σ_{cs}^1	Lemma 10	(3; 1, 1, 0; 1, 1, 0)	Theorem 28	No
σ_{cf}^2	Lemma 18	σ_{cs}^2	Lemma 12	(3; 1, 1, 0; 1, 0, 1)	Theorem 30	No
σ_{cf}^3	Lemma 20	σ_{cs}^3	Lemma 13	(5; 1, 1, 1; 1, 0, 0)	Theorem 33	Yes
σ_{cf}^4	Lemma 22	σ_{cs}^4	Lemma 15	(5; 1, 0, 0; 1, 1, 1)	Theorem 34	Yes

5.1. RE results

From Lemmas 16 through 22, we notice that the context-free rules $p : X \to bY$, $q : X \to Yb$ and $h : S' \to \lambda$ of SGNF are simulated by GCID rules of the following sizes:

$$\sigma_{cf}^1 = (3; 1, 1, 1; 1, 1, 0) \quad \sigma_{cf}^2 = (3; 1, 1, 0; 1, 0, 0)$$
$$\sigma_{cf}^3 = (4; 1, 1, 1; 1, 0, 0) \quad \sigma_{cf}^4 = (5; 1, 0, 0; 1, 1, 1)$$

We note that the sizes σ_{cf}^1 and σ_{cf}^3 are weaker than σ_{cf}^2. However, the underlying control graph of a GCID part of the first two sizes that simulates the context-free rules of SGNF has a linear structure, this is not so with the GCID of size σ_{cf}^2.

From Lemmas 10 through 13, we notice that the deletion rules $f : AB \to \lambda$ and $g : CD \to \lambda$ of SGNF are simulated by GCID rules of the following sizes:

$$\sigma_{cs}^1 = (3; 1, 0, 0; 1, 1, 0) \quad \sigma_{cs}^2 = (3; 1, 0, 0; 1, 0, 1)$$
$$\sigma_{cs}^3 = (5; 1, 1, 1; 1, 0, 0) \quad \sigma_{cs}^4 = (5; 1, 0, 0; 1, 1, 1)$$

We now define a map $\oplus$ as follows: Let $s_1 = (k_1; n_1, i_1', i_1''; m_1, j_1', j_1'')$ and $s_2 = (k_2; n_2, i_2', i_2''; m_1, j_2', j_2'')$ be two sizes of GCID system (parts), then define $s_1 \oplus s_2 := (k; n, i', i''; m, j', j'')$ where

$$k = \max\{k_1, k_2\} \quad n = \max\{n_1, n_2\} \quad i' = \max\{i_1', i_2'\} \quad i'' = \max\{i_1'', i_2''\}$$
$$m = \max\{m_1, m_2\} \quad j' = \max\{j_1', j_2'\} \quad j'' = \max\{j_1'', j_2''\}$$

We choose a σ_{cf}^i $(1 \le i \le 4)$ and a σ_{cs}^j $(1 \le j \le 4)$ and we prove in the following theorems that GCID($\sigma_{cf}^i \oplus \sigma_{cs}^j$) equals RE. In Table 1, we explicitly specify which σ_{cf}^i and σ_{cs}^j go into the stitching of the Theorems 27 through 34.

Since we stitch together the two parts by overlay, the following observation is important.

Proposition 26. *Let Π_1 and Π_2 be two overlayable GCID systems of sizes s_1 and s_2, respectively. Then, $\Pi_1 \cup \Pi_2$ has size $s_1 \oplus s_2$.*

We are now ready to prove our main results. The salient points in the proof of all our following RE results are as follows.

- We consider a type-0 grammar $G = (N, T, S, P)$ in SGNF.
- The rules of P are uniquely labeled with $[1 \ldots |P|]$, which will be used as markers. Sometimes, we also use primed or double-primed markers.
- In order to prove that RE = GCID(δ_{ij}) with k components where $\delta_{ij} = \sigma_{cf}^i \oplus \sigma_{cs}^j$, we construct a graph-controlled insertion–deletion system $\Pi = (k, V, T, \{S\}, H, 1, 1, R)$ such that $L(\Pi) = L(G)$ as follows.
  - Build the GCID part $\Pi_{cs}^{\sigma_j}$ where $\sigma_j = \sigma_{cs}^j$, according to the Lemma cited against σ_{cs}^j.
  - Build the GCID part $\Pi_{cf}^{\sigma_i'}$ where $\sigma_i' = \sigma_{cf}^i$, according to the Lemma cited against σ_{cf}^i.
  - Set $\Pi := \Pi_{cf}^{\sigma_i'} \cup \Pi_{cs}^{\sigma_j}$ by overlay.
  - By Proposition 26, Π is of size $\sigma_{cf}^i \oplus \sigma_{cs}^j = \delta_{ij}$.
- The alphabet of Π satisfies $V \subset N \cup T \cup \{p, p', p'' : p \in [1 \ldots |P|]\}$. The exact definition of the alphabet V will individually follow by the descriptions of the rules.
- The set of rules R (of Π) can be seen as the union of the GCID rules simulating (i) the context-free rules, (ii) the context-free deletion rule and (iii) the non-context-free rules.
- It is known that any word $w \in T^*$ derivable in G can be derived in two phases: first, only context-free rules are applied, starting with S and yielding w' (the first phase ends by simulating $S' \to \lambda$) and then, only non-context-free rules are applied. By the appropriate Lemmas showing the simulation of the stitching components, we know that $(S)_1 \Rightarrow_*' (w')_1 \Rightarrow_*' (w)_1$. This proves that $L(G) \subseteq L(\Pi)$.

Component C1	Component C2	Component C3
$p1.1 : (1, (\lambda, p, X)_{ins}, 2)$	$p2.1 : (2, (p, X, \lambda)_{del}, 3)$ $p2.2 : (2, (p, b, p')_{ins}, 2)$ $p2.3 : (2, (b, Y, p')_{ins}, 2)$ $p2.4 : (2, (\lambda, p, \lambda)_{del}, 3)$ $p2.5 : (2, (Y, p', \lambda)_{del}, 2)$ $p2.6 : (2, (Y, p'', \lambda)_{del}, 1)$	$p3.1 : (3, (p, p', \lambda)_{ins}, 2)$ $p3.2 : (3, (p', p'', \lambda)_{ins}, 2)$
$q1.1 : (1, (\lambda, q, X)_{ins}, 2)$	$q2.1 : (2, (q, X, \lambda)_{del}, 3)$ $q2.2 : (2, (q, Y, q')_{ins}, 2)$ $q2.3 : (2, (Y, b, q')_{ins}, 2)$ $q2.4 : (2, (\lambda, q, \lambda)_{del}, 3)$ $q2.5 : (2, (b, q', \lambda)_{del}, 2)$ $q2.6 : (2, (b, q'', \lambda)_{del}, 1)$	$q3.1 : (3, (q, q', \lambda)_{ins}, 2)$ $q3.2 : (3, (q', q'', \lambda)_{ins}, 2)$
$f1.1 : (1, (\lambda, f, \lambda)_{ins}, 2)$	$f2.1 : (2, (f, A, \lambda)_{del}, 3)$ $f2.2 : (2, (\lambda, f, \lambda)_{del}, 1)$	$f3.1 : (3, (f, B, \lambda)_{del}, 2)$

Fig. 15. GCID rules of size $(3; 1, 1, 1; 1, 1, 0)$ characterizing RE; see Figs. 7(a), 7(b) and 1.

- To prove the converse ($L(\Pi) \subseteq L(G)$), we consider some derivation $(S)_1 \Rightarrow'_* (w)_1$ in Π and show the following.
  - Decompose the above derivation into $(w_i)_1 \Rightarrow' (w_{i+1})_1$, such that $S = w_0$, $w = w_m$, *i.e.*, the chosen derivation moves exactly the strings $w_0, \ldots, w_m$ into component $C1$.
  - Markers are attached when applying rules in $C1$, apart from the (only) context-free deletion rule $h1.1$.
  - The h rule (context-free deletion) simulation cannot be mixed-up with other symbols as S' is a unique symbol from N'.
  - The introduction of rule markers was the reason why the simulations of two non-context-free deletion rules could not interfere with each other in the rules of $\Pi_{cs}^{\sigma_j}$.
  - Similarly, rule markers are also used in the context-free case to avoid the interference with each other in the rules of $\Pi_{cf}^{\sigma'_i}$.
  - *Mixed derivations* (derivations that use the rules of context-free simulation and non-context-free simulation in a mixed fashion) do not produce any unintended/malicious strings.
- In each of the following theorems, we just present the overlay table showing the rules of R in Π and only discuss why mixed derivations (do not) yield (un)intended strings. The correctness of the derivations using just the (non-)context-free rules of $\Pi_{cf}^{\sigma'_i}$ ($\Pi_{cs}^{\sigma_j}$) are shown in the respective lemmas.

We will use the features described in the previous items in the proofs presented in the following without further mentioning. To clarify our constructions, we will however always summarize the different ins–del rules in a table. As the rules simulating $f : AB \to \lambda$ and $g : CD \to \lambda$ are identical up to (uniquely) renaming f (and primed versions thereof) by g, A by C and B by D, we will not list the rules for g; nor, we will list the simulation rule for $h : S' \to \lambda$, as it is always the same.

Theorem 27. $\mathrm{GCID}_L(3; 1, 1, 1; 1, 1, 0) = \mathrm{GCID}_L(3; 1, 1, 1; 1, 0, 1) = \mathrm{RE}$.

This result is already proved in [7]; however, the proof was not approached with the concept of stitching and overlaying. Besides, most importantly, some of the rules discussed in [7] are modified here in view of *guarding* them. Thus, the proof of the following result is different from the proof discussed in [7].

Notice that this theorem will be further improved in Theorem 28. However, this result is also useful, as the underlying control graph has a linear structure and hence this result can be interpreted in terms of P systems. Recall that $\mathrm{GCID}(1; 1, 1, 1; 1, 1, 1)$ equals RE; see [29]. If one desires to have one-sided context for deletion in this system, then this is achieved with three components in this theorem.

Proof. The rules of $\Pi_{cf}^{\sigma_1}$ and $\Pi_{cs}^{\sigma_1}$ given in Figs. 7(a), 7(b) and 1 are stitched together along with $h1.1 : (1, (\lambda, S', \lambda)_{del}, 1)$ and the overlay table showing the stitched rules is presented in Fig. 15.

As said earlier, we now only discuss why mixed derivation do not yield bad strings. Every rule in $C2$ and $C3$ can be applied only if the appropriate marker is present. Also, all rules in $C3$ are double guarded. In other words, once the context-free rule simulation has started, it cannot continue with $C2$-rules stemming from the non-context-free rule simulation and vice versa. This shows that no malicious derivations are possible, so that each derivation (sub)sequence of $(w_i)_1 \Rightarrow' (w_{i+1})_1$ corresponds either to a non-context-free or to a context-free rule application according to Lemmas 16 and 10.

Corollary 8 shows that also GCID systems of size $(3; 1, 1, 1; 1, 0, 1)$ are computationally complete. From the rules given in Fig. 15, one may verify that the underlying control graph of the above simulation is isomorphic to the graph shown in Fig. 8 which is linear. This fact agrees with Proposition 25. □

Up to now, all rules that we used to show computational completeness results were guarded. This was even true (somehow) in the rather complicated simulation of the previous theorem, as the rules $r2.1$ and $r3.1$ (from simulating context-free

Component $C1$	Component $C2$	Component $C3$
$p1.1 : (1, (X, p, \lambda)_{ins}, 3)$	$p2.1 : (2, (p, b, \lambda)_{ins}, 3)$	$\boxed{p3.1 : (3, (\lambda, X, \lambda)_{del}, 1)}$
$p1.2 : (1, (p, p', \lambda)_{ins}, 2)$	$p2.2 : (2, (\lambda, p', \lambda)_{del}, 1)$	$p3.2 : (3, (\lambda, p, \lambda)_{del}, 1)$
$p1.3 : (1, (p', Y, \lambda)_{ins}, 2)$		
$q1.1 : (1, (X, q, \lambda)_{ins}, 3)$	$q2.1 : (2, (q, q', \lambda)_{ins}, 3)$	$\boxed{q3.1 : (3, (\lambda, X, \lambda)_{del}, 1)}$
$q1.2 : (1, (q, b, \lambda)_{ins}, 2)$	$q2.2 : (2, (\lambda, q', \lambda)_{del}, 1)$	$q3.2 : (3, (\lambda, q, \lambda)_{del}, 1)$
$q1.3 : (1, (q', Y, \lambda)_{ins}, 2)$		
$f1.1 : (1, (\lambda, f, \lambda)_{ins}, 2)$	$f2.1 : (2, (f, A, \lambda)_{del}, 3)$	$f3.1 : (3, (f, B, \lambda)_{del}, 2)$
	$f2.2 : (2, (\lambda, f, \lambda)_{del}, 1)$	

Fig. 16. GCID rules of size $(3; 1, 1, 0; 1, 1, 0)$ characterizing RE; see Figs. 9(a), 9(b) and 1.

rules r) could only be successfully applied when matched with the rule marker r in $C4$. This will change in the following simulations. For clarity, we will indicate dangerous unguarded rules by framing them. The following two theorem improves the result of Theorem 27 by further making the context of insertion one-sided.

Theorem 28. $\mathrm{GCID}(3; 1, 1, 0; 1, 1, 0) = \mathrm{GCID}(3; 1, 0, 1; 1, 0, 1) = \mathrm{RE}$.

This result improves a result of [9] where 4 components were used in a GCID system of size $(1, 1, 0; 1, 1, 0)$ to characterize RE and a result of [13]: $\mathrm{GCID}(3; 1, 2, 0; 1, 1, 0) = \mathrm{RE}$ and $\mathrm{GCID}(3; 1, 1, 0; 1, 2, 0) = \mathrm{RE}$. However, now we no longer obtain a linear structure of the control graph.

Proof. The set of rules R (of Π) is defined as the union of the GCID rules simulating the context-free rules as explained in Fig. 9(a) and Fig. 9(b), as well as the context-free deletion rule, plus the non-context-free rules listed in Fig. 1. We collect all the rules of R in Fig. 16.

We now discuss the correctness of the rules. No rule in $C2$ can be applied if the appropriate marker is not present, because all rules are guarded. This also shows that, once the context-free rule simulation has started, it cannot continue with $C2$-rules stemming from the non-context-free rule simulation, nor vice versa. In $C3$, the framed deletion rules $p3.1$ or $q3.1$ could be applied while simulating a non-context-free rule. Namely, they are the only rules that are not guarded. However, there is no way to continue such a derivation, because in $C1$, we will be left-out with the marker f which cannot be erased. Note that $f2.2$ cannot be used to remove such left-over rule markers, because, in order to be able to apply this rule, the system has to apply first apply $f1.1$, which introduces one more rule marker. Also, once (the only) occurrence of a symbol from N' was deleted from the string (by an unintended use of $p3.1$ or $q3.1$), the rules simulating context-free rules are no longer applicable. This shows that no malicious derivations are possible, so that each derivation (sub)sequence of $(w_i)_1 \Rightarrow'_* (w_{i+1})_1$ corresponds either to a non-context-free or to (parts of) a context-free rule application according to Lemmas 18 and 10.

The second part, $\mathrm{GCID}(3; 1, 0, 1; 1, 0, 1) = \mathrm{RE}$, follows from Corollary 8. □

Remark 29. The underlying graph of the previous simulation is not linear, since the graph underlying the simulation of its context-free rules is not linear (see Remark 19). □

Theorem 30. $\mathrm{GCID}(3; 1, 1, 0; 1, 0, 1) = \mathrm{GCID}(3; 1, 0, 1; 1, 1, 0) = \mathrm{RE}$.

This result improves a result of [9] where 4 components were used in a GCID system of size $(1, 1, 0; 1, 0, 1)$ to characterize RE.

Proof. The set of rules R (of Π) is the union of the GCID rules simulating the context-free rules as explained in Fig. 9(a) and Fig. 9(b), as well as the context-free deletion rule, plus the non-context-free rules listed in Fig. 3. We collect all rules in Fig. 17.

As the formal reasoning is completely analogous to the one given in the proof of Theorem 28, we omit it here. □

Remark 31. As in the previous case, the underlying graph of the previous simulation is not linear (also see Remark 19).

Combining Theorems 28 and 30, we have the following.

Corollary 32. $\mathrm{GCID}(3; 1, i', i''; 1, j', j'') = \mathrm{RE}$ *for all* $i', i'', j', j'' \in \{0, 1\}$ *with* $i' + i'' = 1$ *and* $j' + j'' = 1$.

It is known that $\mathrm{GCID}(1; 1, 1, 1; 1, 1, 1) = \mathrm{RE}$ (by [29]). If one desires to have context-free deletions only, then this is achieved using 5 components in the following theorem. In the subsequent theorem (Theorem 34), we prove that even if one desires to have a context-free insertion instead of deletion, then 5 components can be used to achieve this. These results indicate that characterizing RE with context-free deletion/insertion is highly non-trivial.

Component C1	Component C2	Component C3
$p1.1 : (1, (X, p, \lambda)_{ins}, 3)$	$p2.1 : (2, (p, b, \lambda)_{ins}, 3)$	$\boxed{p3.1 : (3, (\lambda, X, \lambda)_{del}, 1)}$
$p1.2 : (1, (p, p', \lambda)_{ins}, 2)$	$p2.2 : (2, (\lambda, p', \lambda)_{del}, 1)$	$p3.2 : (3, (\lambda, p, \lambda)_{del}, 1)$
$p1.3 : (1, (p', Y, \lambda)_{ins}, 2)$		
$q1.1 : (1, (X, q, \lambda)_{ins}, 3)$	$q2.1 : (2, (q, q', \lambda)_{ins}, 3)$	$\boxed{q3.1 : (3, (\lambda, X, \lambda)_{del}, 1)}$
$q1.2 : (1, (q, b, \lambda)_{ins}, 2)$	$q2.2 : (2, (\lambda, q', \lambda)_{del}, 1)$	$q3.2 : (3, (\lambda, q, \lambda)_{del}, 1)$
$q1.3 : (1, (q', Y, \lambda)_{ins}, 2)$		
$f1.1 : (1, (\lambda, f, \lambda)_{ins}, 2)$	$f2.1 : (2, (\lambda, B, f)_{del}, 3)$	$f3.1 : (3, (\lambda, A, f)_{del}, 2)$
	$f2.2 : (2, (\lambda, f, \lambda)_{del}, 1)$	

Fig. 17. GCID rules of size $(3; 1, 1, 0; 1, 0, 1)$ characterizing RE; see Figs. 9(a), 9(b) and 3.

Component C1	Component C2	Component C3
$p1.1 : (1, (\lambda, p, X)_{ins}, 2)$	$p2.1 : (2, (X, p', \lambda)_{ins}, 3)$	$\boxed{p3.1 : (3, (\lambda, X, \lambda)_{del}, 4)}$
	$p2.2 : (2, (\lambda, p, \lambda)_{del}, 1)$	$p3.2 : (3, (\lambda, p', \lambda)_{del}, 2)$
$q1.1 : (1, (\lambda, q, X)_{ins}, 2)$	$q2.1 : (2, (X, q', \lambda)_{ins}, 3)$	$\boxed{q3.1 : (3, (\lambda, X, \lambda)_{del}, 4)}$
	$q2.2 : (2, (\lambda, q, \lambda)_{del}, 1)$	$q3.2 : (3, (\lambda, q', \lambda)_{del}, 2)$
$f1.1 : (1, (\lambda, f, A)_{ins}, 2)$	$f2.1 : (2, (\lambda, A, \lambda)_{del}, 3)$	$f3.1 : (3, (B, f'', \lambda)_{ins}, 4)$
	$f2.2 : (2, (\lambda, f'', \lambda)_{del}, 1)$	$f3.2 : (3, (\lambda, f', \lambda)_{del}, 2)$
Component C4	**Component C5**	
$p4.1 : (4, (p, b, p')_{ins}, 4)$		
$p4.2 : (4, (b, Y, p')_{ins}, 3)$		
$q4.1 : (4, (q, Y, q')_{ins}, 4)$		
$q4.2 : (4, (Y, b, q')_{ins}, 3)$		
$f4.1 : (4, (\lambda, B, \lambda)_{del}, 5)$	$f5.1 : (5, (f, f', f'')_{ins}, 4)$	
$f4.2 : (4, (\lambda, f, \lambda)_{del}, 3)$		

Fig. 18. GCID rules of size $(5; 1, 1, 1; 1, 0, 0)$ characterizing RE; see Figs. 11(a), 11(b) and 4.

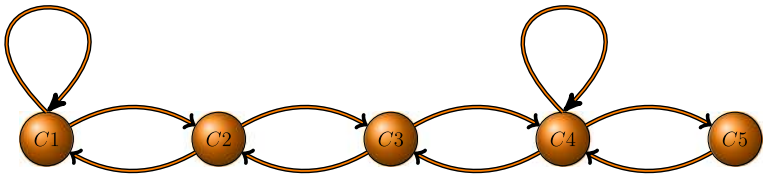

Fig. 19. A control graph corresponding to Theorem 33: $\mathrm{RE} = \mathrm{GCID}_L(5; 1, 1, 1; 1, 0, 0)$.

Theorem 33. $\mathrm{GCID}_L(5; 1, 1, 1; 1, 0, 0) = \mathrm{RE}$.

Proof. The set of rules R (of Π) is the union of the GCID rules simulating the context-free rules as explained in Fig. 11(a) and Fig. 11(b), as well as the context-free deletion rule, plus the non-context-free rules listed in Fig. 4, see Fig. 18 for a summary.

In the following we discuss only about the mixed derivations and why it cannot produce terminal strings. Once a rule in $C1$ is applied (except $h1.1$), one can continue the derivation with the sequence of rules $f.2.1$, $p3.1$ and $f4.1$. If this happens during the process of simulating the context-free rules, when the resultant string reaches $C5$, there will be no f or f'' marker to apply the rule $f5.1$ and the string will be stuck in $C5$.

If such a sequence of rules is applied during the simulation of non-context-free deletion rules, then the resultant string will again be stuck in $C5$, as there is no double-primed marker f'' present in the string. Note that the introduction of a double-primed marker is only possible in $C3$ with the rules $f3.1$ but, instead the rule $p3.1$ or $q3.1$ has been applied. Thus, we can see that the simulation of context-free rules and non-context-free rules are non-interfering. This shows that no malicious derivations are possible, so that each derivation (sub)sequence of $(w_i)_1 \Rightarrow' (w_{i+1})_1$ corresponds either to a non-context-free or to a context-free rule application according to Lemmas 20 and 13.

The underlying graph of the above simulation is as shown in Fig. 19. Note that the label responsible for an edge in the graph is not indicated to avoid cumbersome texts. The corresponding underlying undirected simple graph is a path on 5 vertices, a linear structure. □

Theorem 34. $\mathrm{GCID}_L(5; 1, 0, 0; 1, 1, 1) = \mathrm{RE}$.

Component C1	Component C2	Component C3
$p1.1 : (1, (\lambda, p, \lambda)_{ins}, 2)$	$p2.1 : (2, (\lambda, p', \lambda)_{ins}, 3)$	$p3.1 : (3, (p', X, p)_{del}, 4)$
$p1.2 : (1, (\lambda, \Delta, \lambda)_{ins}, 1)$	$p2.2 : (2, (\lambda, p', b)_{del}, 1)$	$p3.2 : (3, (b, p, Y)_{del}, 2)$
$q1.1 : (1, (\lambda, q, \lambda)_{ins}, 2)$	$q2.1 : (2, (\lambda, q', \lambda)_{ins}, 3)$	$q3.1 : (3, (q, X, q')_{ins}, 4)$
$q1.2 : (1, (\lambda, \Delta, \lambda)_{ins}, 1)$	$q2.2 : (2, (b, q', \lambda)_{del}, 1)$	$q3.2 : (3, (Y, q, b)_{del}, 2)$
$f1.1 : (1, (\lambda, f, \lambda)_{ins}, 2)$	$f2.1 : (2, (\lambda, f', \lambda)_{ins}, 3)$	$f3.1 : (3, (f, B, f')_{del}, 4)$
	$f2.2 : (2, (\lambda, f'', \lambda)_{del}, 1)$	$f3.2 : (3, (f'', f', \lambda)_{del}, 2)$
Component C4	**Component C5**	
$p4.1 : (4, (\lambda, Y, \lambda)_{ins}, 5)$	$p5.1 : (5, (p, \Delta, Y)_{del}, 4)$	
$p4.2 : (4, (\lambda, b, \lambda)_{ins}, 3)$		
$q4.1 : (4, (\lambda, Y, \lambda)_{ins}, 5)$	$q5.1 : (5, (Y, \Delta, q)_{del}, 4)$	
$q4.2 : (4, (\lambda, b, \lambda)_{ins}, 3)$		
$f4.1 : (4, (\lambda, f'', \lambda)_{ins}, 5)$	$f5.1 : (5, (f'', A, f)_{del}, 4)$	
$f4.2 : (4, (f'', f, f')_{del}, 3)$		

Fig. 20. GCID rules of size $(5; 1, 0, 0; 1, 1, 1)$ characterizing RE; see Figs. 13(a), 13(b) and 6.

Proof. The set of rules R (of Π) is the union of the GCID rules simulating the context-free rules as explained in Fig. 13(a) and Fig. 13(b), as well as the context-free deletion rule, plus the non-context-free rules listed in Fig. 6. These rules are collected in Fig. 20.

Notice that according to Lemma 22, especially the last statement, quite a number of dummy symbols can be present in some string $(w_i)_1$, but this fact would not interfere at all with derivations simulating the context-free rules.

There is one caveat here. The introduction of the primed rule marker in $r2.1$ for a context-free rule r is free of context. However, as all rules introducing a primed marker lead to component $C3$ and there, only double-guarded rules are used, we cannot continue beyond $C3$ if we had started (say) with $f1.1$ and had continued (say) with $p2.1$.

The reader might have wondered why we do not run into problems with the (framed) unguarded rules in this simulation. However, notice that the only way to get into $C4$ or $C5$ is via $C3$. As these rules are double-guarded, they require a previous application of the according rules in $C1$ and $C2$. Assume first that we had started a simulation of a context-free rule, say, of type p, hence introducing the markers p and p', but neither f nor f', into the string that is moved to $C4$. Clearly, $f4.2$ is not applicable now. If we use $f4.1$, then we are stuck in $C5$, as $X \neq Y$, so that $p5.1$ is not applicable. Secondly, if we had started a simulation of a non-context-free rule, say, rule f, we would have introduced the markers f and f', but no markers of context-free rules. If we now used, say, $p4.1$ instead of $f4.1$, then we cannot continue in $C5$, as the symbols p and f'' required in the rules of $C5$ are not present. Likewise, if we applied $p4.2$, then we are back to $C3$, where all rules are double-guarded, which means that none of the rules is applicable to our current string.

This shows that no malicious derivations are possible, so that each derivation (sub)sequence of $(w_i)_1 \Rightarrow' (w_{i+1})_1$ corresponds either to a non-context-free or to (part of) a context-free rule application according to Lemmas 22 and 15. □

Remark 35. For all i, $1 \leq i \leq 5$, we have $\mathrm{LIN} \subsetneq \mathrm{GCID}(\sigma^i_{cf})$ by Remark 24. Further, from the theorems proved in this section, we may note that the system $\mathrm{GCID}(\sigma^i_{cf})$ will itself generate RE (*i.e.*, far beyond LIN) if $i = 1, 2, 5$. We achieved this by providing some σ_{cs} weaker than σ^i_{cf} such that $\sigma_{cs} \oplus \sigma^i_{cf} = \sigma^i_{cf}$. Whether this is possible for the other cases remains open. In particular, from Remark 24, it is clear that we can simulate LIN using $\mathrm{GCID}(\sigma^3_{cf})$ (see proposition below), but we were unable to characterize RE with this size. □

Proposition 36. $\mathrm{LIN} \subsetneq \mathrm{GCID}(3; 1, 1, 0; 1, 0, 0) \cap \mathrm{GCID}(3; 1, 0, 1; 1, 0, 0)$.

Proof. According to Lemma 18 and Remark 36, GCID systems of size $(3; 1, 1, 0; 1, 0, 0)$ generate more than LIN, thus proving the first part of the statement. The second part follows from Corollary 8, as LIN is closed under reversal. □

6. Computational completeness of ins–del P systems

In this section, we connect our obtained results in Sec. 5.1 with ins–del P systems. We use $\mathrm{ELSP}_k(\mathrm{INS}_n^{i',i''}\mathrm{DEL}_m^{j',j''})$ (as used by Gh. Păun in [26]) to represent the family of languages generated by ins–del P systems of k membranes and size (n, i', i'', m, j', j''), where the size parameters have the same meaning as in GCID systems. At this point, we recall that if the underlying undirected simple graph of a GCID system establishes a tree (or in particular a linear) structure, then this GCID system can also be seen as an ins–del P system. On the other hand, in a P system, there is no specific initial membrane where the computation begins since the membranes evolve in a maximally parallel way. But artificially, if we make the axiom in each membrane (except one) to be empty, then this exceptional membrane can be viewed as a initial membrane to begin with and such a system works in the same way as a GCID system where the membranes of a P system correspond to the components of a GCID system. It is known that the following ins–del P systems are computationally complete.

Table 2
Analysis of the generative power of GCID system of sizes $(k; 1, i', i''; 1, j', j'')$.

ω	Size of the system $(k; 1, i', i''; 1, j', j'')$	Value of k	Language class relation	Reference	Remark	Control graph linear?
2	$(k; 1, 0, 0; 1, 0, 0)$	1	$\subsetneq$ REG	[30,19]		
2	$(k; 1, 0, 0; 1, 0, 0)$	≥ 2	$\subseteq$ MAT	[2]	MAT $\subsetneq$ RE	
3	$(k; 1, 0, 0; 1, 1, 0)$	–	OPEN	–		
3	$(k; 1, 0, 0; 1, 0, 1)$	–	OPEN	–		
4	$(k; 1, 0, 0; 1, 1, 1)$	5	$=$ RE	Theorem 34		Yes
3	$(k; 1, 1, 0; 1, 0, 0)$	3	$\supsetneq$ LIN	Proposition 36		No
4	$(k; 1, 1, 0; 1, 1, 0)$	3	$=$ RE	Theorem 28	Improves [9,13]	No
4	$(k; 1, 1, 0; 1, 0, 1)$	3	$=$ RE	Theorem 30	Improves [9]	No
5	$(k; 1, 1, 0; 1, 1, 1)$	3	$=$ RE	Theorem 28	$\neq$ RE if $k = 1$ [19]	No
3	$(k; 1, 0, 1; 1, 0, 0)$	3	$\supsetneq$ LIN	Proposition 36		No
4	$(k; 1, 0, 1; 1, 1, 0)$	3	$=$ RE	Theorem 30		No
4	$(k; 1, 0, 1; 1, 0, 1)$	3	$=$ RE	Theorem 28		No
5	$(k; 1, 0, 1; 1, 1, 1)$	3	$=$ RE	Theorem 28		No
4	$(k; 1, 1, 1; 1, 0, 0)$	5	$=$ RE	Theorem 33		Yes
5	$(k; 1, 1, 1; 1, 1, 0)$	3	$=$ RE	Theorem 28	$\neq$ RE if $k = 1$ [19]	Yes
5	$(k; 1, 1, 1; 1, 0, 1)$	3	$=$ RE	Theorem 28		Yes
6	$(k; 1, 1, 1; 1, 1, 1)$	1	$=$ RE	[29]		Yes

- [9] $\mathrm{ELSP}_4(\mathrm{INS}_1^{1,0}\mathrm{DEL}_2^{0,0})$, $\mathrm{ELSP}_4(\mathrm{INS}_2^{0,0}\mathrm{DEL}_1^{1,0})$
- [13] $\mathrm{ELSP}_3(\mathrm{INS}_1^{2,0}\mathrm{DEL}_1^{1,0})$, $\mathrm{ELSP}_3(\mathrm{INS}_1^{1,0}\mathrm{DEL}_1^{2,0})$

From Table 1, one may note that the underlying control graph of the GCID systems (characterizing RE) of Theorems 27, 33 and 34 has a linear structure. In other words, we have the following proposition.

Proposition 37. *The results of the Theorems 27, 33 and 34 correspond, respectively, to the following computational completeness results of ins–del P systems.*

1. $\mathrm{ELSP}_3(\mathrm{INS}_1^{1,1}\mathrm{DEL}_1^{j',j''}) = \mathrm{RE}$ *for* $j', j'' \in \{0, 1\}$ *and* $j' + j'' = 1$,
2. $\mathrm{ELSP}_5(\mathrm{INS}_1^{1,1}\mathrm{DEL}_1^{0,0}) = \mathrm{RE}$,
3. $\mathrm{ELSP}_5(\mathrm{INS}_1^{0,0}\mathrm{DEL}_1^{1,1}) = \mathrm{RE}$.

Remark 38. In a P system, the output membrane is either the elementary membrane (*i.e.*, one of the innermost membranes) or the out-of-skin membrane (*i.e.*, the environment). If we restrict the output membrane to be the same as the initial membrane (the membrane where the computation starts), then such a P system may correspond to returning GCID systems discussed throughout this paper. This observation might lead to more investigations on *linear* P systems in the future; also see [1].

7. Conclusions and open problems

In this article, we have investigated the computational completeness of graph-controlled insertion–deletion systems of size $(k; 1, i', i''; 1, j', j'')$ for all values of $i', i'', j', j'' \in \{0, 1\}$, with $k \in \{3, 5\}$. We have collected our results in Table 2. The following points can be easily observed from the table.

- Out of these 16 types of GCID systems, it is found that 11 types characterize RE, one is weaker than REG $((1; 1, 0, 0; 1, 0, 0))$ and two contain LIN. The power of other remaining 2 system types (namely $(k; 1, 0, 0; 1, 1, 0)$ and $(k; 1, 0, 0; 1, 0, 1)$) is left open. Note that (if the simulated family is closed under reversal) the generative power of the former will tell something about the generative power of the latter and vice versa, by Corollary 8.
- Comparing $\omega = n + i' + i'' + m + j' + j''$ (specified in the first column) and k (the number of components) given in the third column, we may observe the following points.
  - For $\omega = 2$, the according GCID systems either characterize the class of finite languages (for the non-binary size $(0, 0, 0; 2, 0, 0)$), or are only capable of generating context-sensitive languages (for the size $(2, 0, 0; 0, 0, 0)$), or are included in the class MAT of languages that can be described by context-free matrix grammars without appearance checking (according to [2, Theorem 4], for the size $(1, 0, 0; 1, 0, 0)$). Hence, in no case, such GCID systems can be computationally complete, nor is the class of languages that can be described by any GCID systems of weight two equal to RE.
  - If $\omega = 3$, then whether a GCID system with weight ω characterizes RE is still open.
  - If $\omega = 4$ and if the contexts of the insertion/deletion are one-sided, then GCID systems require 3 components to describe RE, assuming that the mentioned conjecture of S. Ivanov and S. Verlan [13] is correct, as this implies that

such systems with only two components are not computationally complete. Under this conjecture, our results are hence best possible for this type of GCID systems.
- If $\omega = 4$ and if the insertion/deletion are performed in a context-free manner, then GCID systems need 5 components to describe RE in our simulations with underlying control graph being linear.
- If $\omega = 5$ and if the insertion/deletion context is one-sided, then GCID systems require 3 components to describe RE, assuming that the mentioned conjecture of S. Ivanov and S. Verlan [13] is correct. Under this conjecture, our results are hence optimal for this type of GCID systems. If deletion is one-sided, then underlying control graph is linear. However it is not the case when insertion is one-sided.

From this, note that k does not decrease if we increase ω or vice versa. The results of this paper do not provide any relationship between ω and k. Instead of looking at the total weight, we believe that comparing individual size parameters to the number of components could shed even more light upon the contribution of graph control to the power of insertion and deletion.

It might be interesting to note that the classical definition of a *grammar controlled by a bicolored digraph*, see [5], differs in one aspect from the graph-controlled insertion–deletion systems as considered here: namely, the latter systems allow individual transitions to the next components, while in the former grammars, these transitions are given by the edges of the control graph, so that all rules in one node (component) have to transit to the same next node. It is therefore interesting to study correspondingly defined insertion–deletion systems controlled by a bicolored digraph. This is also discussed in [6,12].

As matrix insertion–deletion system is a particular case of graph-controlled insertion–deletion system and already some computationally completeness results of the former has been discussed in literature (see, for example, [8,27]), it would be interesting to study (in an approach similar to Sec. 6) what descriptional complexity results of GCID systems correspond to matrix ins–del systems and vice versa.

We are currently studying two aspects of GCID ins–del systems:

- What is the power of these systems if we insist on linear or tree structures of the control graph?
- In the case that we cannot prove that GCID systems of certain sizes characterize RE, can we say more about their power by comparing them with other language families known from the literature?

Acknowledgements

Some part of the work done by the second author was during the author's visits to the University of Trier, Germany, in June–July and December, 2016. The possibility to use some overhead money from a DFG grant FE 560 6/1 to finance this visit is gratefully acknowledged. We gratefully acknowledge Serghei Verlan and Rudolf Freund for their comments and feedback on earlier versions of this paper. The skillful work of the referees of *Theoretical Computer Science* added a lot to the quality of this paper, as well.

References

[1] A. Alhazov, R. Freund, S. Ivanov, Length P systems, Fund. Inform. 134 (1–2) (2014) 17–37.
[2] A. Alhazov, A. Krassovitskiy, Y. Rogozhin, S. Verlan, P systems with minimal insertion and deletion, Theoret. Comput. Sci. 412 (1–2) (2011) 136–144.
[3] R. Benne (Ed.), RNA Editing: The Alteration of Protein Coding Sequences of RNA, Series in Molecular Biology, Ellis Horwood, Chichester, UK, 1993.
[4] F. Biegler, M.J. Burrell, M. Daley, Regulated RNA rewriting: modelling RNA editing with guided insertion, Theoret. Comput. Sci. 387 (2) (2007) 103–112.
[5] J. Dassow, Gh. Păun, Regulated Rewriting in Formal Language Theory, EATCS Monographs in Theoretical Computer Science, vol. 18, Springer, 1989.
[6] H. Fernau, An essay on general grammars, J. Autom. Lang. Comb. 21 (2016) 69–92.
[7] H. Fernau, L. Kuppusamy, I. Raman, Descriptional complexity of graph-controlled insertion–deletion systems, in: C. Câmpeanu, F. Manea, J.O. Shallit (Eds.), Descriptional Complexity of Formal Systems, 18th International Conference, DCFS, in: LNCS, vol. 9777, Springer, 2016, pp. 111–125.
[8] H. Fernau, L. Kuppusamy, I. Raman, Generative power of matrix insertion–deletion systems with context-free insertion or deletion, in: M. Amos, A. Condon (Eds.), Unconventional Computation and Natural Computation Conference, UCNC, in: LNCS, vol. 9726, Springer, 2016, pp. 35–48.
[9] R. Freund, M. Kogler, Y. Rogozhin, S. Verlan, Graph-controlled insertion–deletion systems, in: I. McQuillan, G. Pighizzini (Eds.), Proceedings Twelfth Annual Workshop on Descriptional Complexity of Formal Systems, DCFS, in: EPTCS, vol. 31, 2010, pp. 88–98.
[10] V. Geffert, Normal forms for phrase-structure grammars, RAIRO Theor. Inform. Appl. 25 (1991) 473–498.
[11] D. Haussler, Insertion languages, Inform. Sci. 31 (1) (1983) 77–89.
[12] S. Ivanov, On the Power and Universality of Biologically-Inspired Models of Computation, PhD Thesis, Université de Paris-Est, France, 2014.
[13] S. Ivanov, S. Verlan, About one-sided one-symbol insertion–deletion P systems, in: A. Alhazov, S. Cojocaru, M. Gheorghe, Y. Rogozhin, G. Rozenberg, A. Salomaa (Eds.), Membrane Computing – 14th International Conference, CMC 2013, in: LNCS, vol. 8340, Springer, 2014, pp. 225–237.
[14] S. Ivanov, S. Verlan, Random context and semi-conditional insertion–deletion systems, Fund. Inform. 138 (2015) 127–144.
[15] S. Ivanov, S. Verlan, Universality of graph-controlled leftist insertion–deletion systems with two states, in: J. Durand-Lose, B. Nagy (Eds.), Machines, Computations, and Universality – 7th International Conference, MCU, in: LNCS, vol. 9288, Springer, 2015, pp. 79–93.
[16] L. Kari, On Insertion and Deletion in Formal Languages, PhD Thesis, University of Turku, Finland, 1991.
[17] L. Kari, Gh. Păun, G. Thierrin, S. Yu, At the crossroads of DNA computing and formal languages: characterizing recursively enumerable languages using insertion–deletion systems, in: H. Rubin, D.H. Wood (Eds.), DNA Based Computers III, in: DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 48, 1999, pp. 329–338.
[18] L. Kari, G. Thierrin, Contextual insertions/deletions and computability, Inform. and Comput. 131 (1) (1996) 47–61.
[19] A. Krassovitskiy, Y. Rogozhin, S. Verlan, Further results on insertion–deletion systems with one-sided contexts, in: C. Martín-Vide, F. Otto, H. Fernau (Eds.), Language and Automata Theory and Applications, Second International Conference, LATA, in: LNCS, vol. 5196, Springer, 2008, pp. 333–344.
[20] A. Krassovitskiy, Y. Rogozhin, S. Verlan, Computational power of insertion–deletion (P) systems with rules of size two, Nat. Comput. 10 (2011) 835–852.

[21] S.N. Krishna, R. Rama, Insertion–deletion P systems, in: N. Jonoska, N.C. Seeman (Eds.), DNA Computing, 7th International Workshop on DNA-Based Computers, 2001, Revised Papers, in: LNCS, vol. 2340, Springer, 2002, pp. 360–370.
[22] L. Kuppusamy, A. Mahendran, Modelling DNA and RNA secondary structures using matrix insertion–deletion systems, Int. J. Appl. Math. Comput. Sci. 26 (1) (2016) 245–258.
[23] L. Kuppusamy, A. Mahendran, S.N. Krishna, Matrix insertion–deletion systems for bio-molecular structures, in: R. Natarajan, A.K. Ojo (Eds.), Distributed Computing and Internet Technology – 7th International Conference, ICDCIT, in: LNCS, vol. 6536, Springer, 2011, pp. 301–312.
[24] L. Kuppusamy, R. Rama, On the power of tissue P systems with insertion and deletion rules, in: Pre-Proc. of Workshop on Membrane Computing, Report RGML, vol. 28, Univ. Tarragona, Spain, 2003, pp. 304–318.
[25] M. Margenstern, Gh. Păun, Y. Rogozhin, S. Verlan, Context-free insertion–deletion systems, Theoret. Comput. Sci. 330 (2) (2005) 339–348.
[26] Gh. Păun, Membrane Computing: An Introduction, Springer, 2002.
[27] I. Petre, S. Verlan, Matrix insertion–deletion systems, Theoret. Comput. Sci. 456 (2012) 80–88.
[28] Gh. Păun, G. Rozenberg, A. Salomaa, DNA Computing: New Computing Paradigms, Springer, 1998.
[29] A. Takahara, T. Yokomori, On the computational power of insertion–deletion systems, Nat. Comput. 2 (4) (2003) 321–336.
[30] S. Verlan, On minimal context-free insertion–deletion systems, J. Autom. Lang. Comb. 12 (1–2) (2007) 317–328.
[31] S. Verlan, Recent developments on insertion–deletion systems, Comput. Sci. J. Moldova 18 (2) (2010) 210–245.