Acta Informatica (2019) 56:35–59
https://doi.org/10.1007/s00236-018-0312-2

ORIGINAL ARTICLE

On path-controlled insertion–deletion systems

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

Received: 4 September 2017 / Accepted: 10 January 2018 / Published online: 5 February 2018

Abstract A graph-controlled insertion–deletion system is a regulated extension of an insertion–deletion system. It has several components and each component contains some insertion–deletion rules. These components are the vertices of a directed control graph. A transition is performed by any applicable rule in the current component on a string and the resultant string is then moved to the target component specified in the rule. This also describes the arcs of the control graph. Starting from an axiom in the initial component, strings thus move through the control graph. The language of the system is the set of all terminal strings collected in the final component. In this paper, we investigate a variant of the main question in this area: which combinations of size parameters (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; plus three similar restrictions with respect to deletion) are sufficient to maintain computational completeness of such restricted systems under the additional restriction that the (undirected) control graph is a path? Notice that these results also bear consequences for the domain of insertion–deletion P systems, improving on a number of previous results from the literature, concerning in particular the number of components (membranes) that are necessary for computational completeness results.

✉ Indhumathi Raman
indhumathi.r@vit.ac.in

Henning Fernau
fernau@uni-trier.de

Lakshmanan Kuppusamy
klakshma@vit.ac.in

[1] Fachbereich 4 – Abteilung Informatikwissenschaften, CIRT, Universität Trier, 54286 Trier, Germany

[2] School of Computer Science and Engineering, VIT University, Vellore 632 014, India

[3] School of Information Technology and Engineering, VIT University, Vellore 632 014, India

1 Introduction

The motivation for insertion–deletion system comes from both linguistics [26] (see [29] as a textbook on this topic) and also from biology, more specifically from DNA processing and RNA editing. In particular, in the theoretical process of mismatched annealing of DNA sequences, certain segments of the strands are either inserted or deleted [31]. Likewise, in RNA editing, some fragments of messenger RNA are inserted or deleted [3,4]. Another mathematical motivation for insertion operations can be seen in [15], where this operation and its iterated variant were introduced as a generalization of concatenation and Kleene's closure. The deletion operation was first studied in [18] from a formal language perspective, where the deletion is defined as a quotient-like operation that does not necessarily happen at the right end of the string. Insertion and deletion operations together were introduced in a formal language theoretic fashion in [20]. The corresponding grammatical mechanism is called *insertion–deletion system* (abbreviated as ins–del system). Informally, 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*. Several variants of ins–del systems have been considered in the literature, like ins–del P systems [2,23], tissue P systems with ins–del rules [25], context-free ins–del systems [27], matrix ins–del systems [10,24,32], random context and semi-conditional ins–del systems [17], etc. All the mentioned papers (as well as [19,33]) characterized the recursively enumerable languages (i.e., they show *computational completeness*) using ins–del systems. We refer to the survey article [34] for details of variants of ins–del systems; this survey also discusses some proof techniques for showing computationally completeness results.

One of the important variants of ins–del systems is *graph-controlled ins–del systems* (abbreviated as GCID systems), introduced in [13] and further studied in [16]. In such a system, the concept of *components* is introduced, which are associated with insertion or deletion rules. The transition is performed by choosing any applicable rule from the set of rules of the current component and by moving the resultant string to the target component specified in the rule in order to continue processing it.

If the underlying graph of a GCID system establishes a path structure (loops, multiple edges and directions are ignored), then such a GCID system can be seen as a special form of a *P system*, namely, an *ins–del P system*. As P systems (a model for *membrane computing*) draw their origins from modeling computations of biological systems, considering insertions and deletions in this context is particularly meaningful. There is one small technical issue, namely, in a P system, usually there is no specific initial membrane where the computation begins, since the membranes evolve in a so-called maximally parallel way. However, if the collection of axioms in each membrane, except one membrane is empty, then this exceptional membrane can be viewed as an initial membrane to begin with, so that 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. For more details, see [30].

The mentioned connections motivate to study GCID systems. Much research has then be devoted to restricting the computational resources as far as possible while still maintaining computational completeness. To be more concrete, typical questions are: To what extent can we limit the length of contexts and/or the insertion or deletion strings in a rule? How many components are needed with the limited size? Are there kind of trade-offs between these questions? All this is formalized in the following.

In this paper, our objective is to find the sizes $(k; n, i', i''; m, j', j'')$ with which GCID systems can (still) describe the recursively enumerable languages, with the control graphs of the systems being paths so that these results will correspond to computational completeness

results of particular ins–del P sytems, as noted above. The descriptional complexity of a GCID system is measured by its size $(k; n, i', i''; m, j', j'')$ where the parameters from left to right denote (i) number of components, (ii) the maximal length of the insertion string, (iii) the maximal length of the left context used in insertion rules, (iv) the maximal length of the right context used in insertion rules and the last three parameters follow a similar representation as (ii), (iii), (iv) with respect to deletion instead of insertion. The generative power of GCID systems for insertion/deletion lengths satisfying $n + m \in \{2, 3\}$ has also been studied in [11,12,16]. However, the control graph is not a path for many cases, so that they cannot be also viewed as ins–del P systems.

The main objective of this paper is to characterize recursively enumerable languages (denoted as RE) by size-bounded GCID systems, whose underlying (undirected) control graph is a path. This objective can be seen as a sort of *syntactic restriction* on GCID systems, on top of the usually considered numerical values limiting the descriptional complexity. We are interested in the question which type of resources of path-structured GCID (as well as of membrane computing, i.e., P) systems are sufficient to characterize RE. We prove that GCID system of sizes $(k; n, i', i''; 1, j', j'')$ with $i', i'', j', j'' \in \{0, 1\}$, and (i) $k = 3,\ n = 1,\ i' + i'' = 1,\ j' + j'' = 2$, (ii) $k = 3,\ n = 2,\ i' + i'' = 1,\ j' + j'' = 1$, (iii) $k = 4,\ n = 2,\ i' + i'' = 0,\ j' + j'' = 1$, (iv) $k = 4,\ n = 1,\ i' + i'' = 1,\ j' + j'' = 1$, all characterize RE with a path as a control graph. Previously, such results were only known for GCID systems with arbitrary control graphs [11]. Our results may also revive interest in the conjecture of Ivanov and Verlan [16] which states that RE $\neq$ GCID(s) if $k = 2$ in $s = (k; 1, i', i''; 1, j', j'')$, with $i', i'', j', j'' \in \{0, 1\}$ and $i' + i'' + j' + j'' \leq 3$. In the same situation, this statement is known to be true if $k = 1$. If the conjecture were true, then our results for $k = 3$ would be optimal. We will make this and other open problems explicit throughout the paper. Also, at the end of this paper, the reader can find tables summarizing the state-of-the-art in this area.

This study could also raise interest in considering other forms of syntactic restrictions, for instance, cycles (somehow reminiscent of time-variant control [5]) or stars and similar special trees, or variants of our notion of paths.

A preliminary version of this paper appeared in [8]; this version not only contains all proof details but also some new results.

2 Preliminaries

We assume that the readers are familiar with the standard notations used in formal language theory. Here, we recall a few notations for the understanding of the paper. 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 $w \in \Sigma^*$, $|w|$ is the length of w and w^R denotes the reversal (mirror image) of w. L^R and $\mathcal{L}^R$ are also understood for languages L and language families $\mathcal{L}$, collecting all reversals of words from L and all reversals of languages from $\mathcal{L}$, respectively. We also make use of the *shuffle operation* $\sqcup\!\sqcup$ to describe the effect of insertions at a random position in the string.[1] For the computational completeness results, we are using

[1] The *shuffle operation*, denoted by $\sqcup\!\sqcup$, is defined recursively by

$$(au \sqcup\!\sqcup bv) = a(u \sqcup\!\sqcup bv) \cup b(au \sqcup\!\sqcup v)$$

and $(u \sqcup\!\sqcup \lambda) = (\lambda \sqcup\!\sqcup u) = \{u\}$, where $u, v \in \Sigma^*$ and $a, b \in \Sigma$.

as our main tool the fact that type-0 grammars[2] in Special Geffert Normal Form (SGNF) are known to characterize the recursively enumerable languages.

Definition 1 A type-0 grammar $G = (N, T, P, S)$ is said to be in SGNF if

- N decomposes as $N = N' \cup N''$, where $N'' = \{A_1, B_1, A_2, B_2\}$ and N' contains at least the two nonterminals S and S';
- the only non-context-free rules in P are $AB \to \lambda$, where $AB \in \{A_1B_1, A_2B_2\}$;
- the context-free rules are of the form (i) $S' \to \lambda$, or (ii) $X \to Y_1Y_2$, where $X \in N'$ and $Y_1Y_2 \in ((N'\backslash\{X\})(T \cup N'')) \cup ((T \cup N'')(N'\backslash\{X\}))$.

The way the normal form is constructed is described in [13], based on [14]. We assume, without loss of generality, in this paper that the context-free rules $r : X \to Y_1Y_2$ either satisfy $Y_1 \in \{A_1, A_2\}$ and $Y_2 \in N'$, or $Y_1 \in N'$ and $Y_2 \in \{B_1, B_2\} \cup T$. This also means that the derivation in G undergoes two phases: in phase I, only context-free rules are applied. This phase ends with applying the context-free deletion rule $S' \to \lambda$. Then, only non-context-free deletion rules are applied in phase II. Notice that the symbol from N', as long as present, separates A_1 and A_2 from B_1 and B_2; this prevents a premature start of phase II. We write $\Rightarrow_r$ to denote a single derivation step using rule r, 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 Graph-controlled insertion–deletion systems

Definition 2 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,
- $A \subset V^*$ is a finite set of axioms,
- H is a set of labels associated (in a one-to-one manner) to the rules in R,
- $i_0 \in [1 \ldots k]$ is the initial component,
- $i_f \in [1 \ldots k]$ is the final component, and
- R is a finite set of rules of the form $l : (i, r, j)$, where l is the label of the rule, r is an insertion rule of the form $(u, \eta, v)_I$ or a deletion rule of the form $(u, \delta, v)_D$, where $(u, v) \in V^* \times V^*$, $\eta, \delta \in V^+$ and $i, j \in [1 \ldots k]$.

If the initial component itself is the final component, then we call the system *returning*. 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). We write rules in R in the form $l : (i, r, j)$, where $l \in H$ is the label associated to the rule. Often, the component is part of the label name. This will also (implicitly) define H. We shall omit the label l of the rule wherever it is not necessary for the discussion.

[2] A type-0 grammar G is usually specified by a quadruple (N, T, P, S) consisting of a nonterminal alphabet N, a terminal alphabet T, a finite set of (production) rules P and a start symbol $S \in N$. Rules are written in the form $\alpha \to \beta, \alpha, \beta \in (N \cup T)^*$. This defines a rewrite relation $\Rightarrow_G \subseteq (N \cup T)^* \times (N \cup T)^*$, with $u \Rightarrow_G v$ if v is obtained from u by replacing the subword α by β, for some $\alpha \to \beta \in P$. The reflexive transitive closure $\Rightarrow_G^*$ can be used to define the semantics of G—the language of G—collecting all $w \in T^*$ with $S \Rightarrow_G^* w$.

We now describe how GCID systems work. Applying an insertion rule of the form $(u, \eta, v)_I$ means that the string η is inserted between u and v; this corresponds to the rewriting rule $uv \to u\eta v$. Similarly, applying a deletion rule of the form $(u, \delta, v)_D$ means that the string δ is deleted between u and v; this corresponds to the rewriting rule $u\delta v \to uv$. A *configuration* of Π is represented by $(w)_i$, where $i \in [1 \dots k]$ is the number of the current component and $w \in V^*$ is the current string. In that case, we also say that w has entered component Ci. We write $(w)_i \Rightarrow_l (w')_j$ or $(w')_j \Leftarrow_l (w)_i$ if there is a rule $l : (i, r, j)$ in R, and w' is obtained by applying the insertion or deletion rule r to w. By $(w)_i \overset{\Rightarrow_l}{\Leftarrow_{l'}} (w')_j$, we mean that $(w')_j$ is derivable from $(w)_i$ using rule l and $(w)_i$ is derivable from $(w')_j$ using rule l'. For brevity, we write $(w)_i \Rightarrow (w')_j$ if there is some rule l in R such that $(w)_i \Rightarrow_l (w')_j$. To avoid confusion with traditional grammars, we write $\Rightarrow_*$ for the transitive reflexive closure of $\Rightarrow$ between configurations. The language $L(\Pi)$ generated by Π is defined as $\{w \in T^* \mid (x)_{i_0} \Rightarrow_* (w)_{i_f} \text{ for some } x \in A\}$. For returning GCID systems Π with initial component $C1$, we also write $(w)_1 \Rightarrow' (w')_1$, meaning that there is a sequence of derivation steps $(w)_1 \Rightarrow (w_1)_{c_1} \Rightarrow \cdots \Rightarrow (w_k)_{c_k} \Rightarrow (w')_1$ such that, for all $i \in \{1, \dots, k\}$, $c_i \neq 1$.

The *underlying control graph* of a graph-controlled insertion–deletion system Π with k components is defined to be a graph with k nodes labelled $C1$ through Ck and there exists a directed edge from a node Ci to node Cj 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 there exists a rule of the form (i, r_1, j) or (j, r_2, i) in R of Π (hence, loops and multi-edges are excluded). We call a returning GCID system with k components *path-structured* if its underlying undirected control graph has the edge set $\{\{Ci, C(i+1)\} \mid i \in [1 \dots k-1]\}$. Incidentally, when the underlying control graph forms a path or tree structure, this system can be interpreted as an insertion–deletion P system [23], 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 (in parallel) and are sent to adjacent membranes, as specified in the rule as its *target* membrane. For more details, see [30]. We also mention the paper [1] on length P systems, where a linear membrane structure with multiset objects in the form of vectors/numbers is considered.

Let us mention one technicality with our definition: a GCID on three components $C1, C2, C3$ with (un)directed arcs between $C1$ and $C2$, as well as between $C2$ and $C3$, would qualify as path-structured if it is returning with respect to any initial (and at the same time, final) component, while when starting with $C2$, the whole computation looks star-like rather than path-like. This observation might motivate further studies on returning path-structured GCID systems with initial component $C1$. Some but not all of our simulations provide simulations by path-structured GCID systems following this more restricted notion.

The *descriptional complexity* of a GCID system is measured by its *size* $s = (k; n, i', i''; m, j', j'')$, where the parameters represent resource bounds as given in Table 1. Slightly abusing notation, the language class generated by GCID systems of size s is denoted by $\mathrm{GCID}(s)$ and the class of languages describable by path-structured GCID systems of size s is denoted by $\mathrm{GCID}_P(s)$.

A first thought might be that the path-structure is not restricting the power of GCID systems at all. For instance, assume that we have a rule $(3, (a, x, b)_I, 1)$ (from $C3$ directly going to $C1$ without going via $C2$) that breaches some otherwise path-structured GCID system. Adding a new symbol A, one might try to substitute this rule by the three rules $(3, (a, A, b)_I, 2)$, $(2, (A, x, b)_I, 1)$ and $(1, (a, A, x)_D, 1)$. However, there are at least two major concerns against this strategy:

Table 1 Size $(k; n, i', i''; m, j', j'')$ of a GCID system

k = the number of components	
$n = \max\{\|\eta\|\colon (i, (u, \eta, v)_I, j) \in R\}$	$m = \max\{\|\delta\|\colon (i, (u, \delta, v)_D, j) \in R\}$
$i' = \max\{\|u\|\colon (i, (u, \eta, v)_I, j) \in R\}$	$j' = \max\{\|u\|\colon (i, (u, \delta, v)_D, j) \in R\}$
$i'' = \max\{\|v\|\colon (i, (u, \eta, v)_I, j) \in R\}$	$j'' = \max\{\|v\|\colon (i, (u, \delta, v)_D, j) \in R\}$

- In all interesting situations considered in this paper, we did not allow both left and right contexts for insertion and deletion strings. In other words, we did not consider the size $(n, 1, 1; m, 1, 1)$ in this paper. Note that ins–del system with this size $(n, m \geq 1)$ describes RE.
- Especially when trying to do the same simulation with less contexts, there is a danger that, instead of following the three rules in order (as intended), the derivation might be interrupted by applying other rules. These unintended derivations might be malicious, in the sense of producing terminal words that do not belong to the language of the simulated grammar. Of course, this problem depends on the concrete GCID system, but it shows the difficulties of this approach in general.

In fact, the danger of having malicious derivations in certain simulations will be the main concern in the proofs that follow. As such derivations must be ruled out, quite detailed induction proofs are given in the next section that compiles the main results of this paper.

3 Computational completeness

In this section, we will demonstrate our computational completeness claims by explaining simulations of type-0 grammars in SGNF, as done in previously published constructions. One of the main techniques will be the introduction of specific symbols called *markers*, associated to each rule that is to be simulated, so that we can argue that a certain sequence of rule applications have to be followed by the simulating GCID system, plus taking care of the positions where the rule applications take place.

Let us look at a simple example first, taken from a more involved construction discussed below. In order to simulate the context-sensitive deletion rule $f_1 : A_1B_1 \to \lambda$ from the given SGNF grammar, Table 2 suggests to have rule $f_1 1.1 : (1, (\lambda, f_1, \lambda)_I, 2)$ in the first component. Here we can also see some conventions that we follow to label the simulating rules: $f_1 1.1$ is the first rule of component $C1$ that is responsible for simulating the original rule f_1. This rule $f_1 1.1$ inserts the rule marker f_1 anywhere in the current string, which is then moved to $C2$. If now $f_1 2.2$ is applied (i.e., the second rule of component $C2$ that is responsible for simulating f_1), then the rule marker f_1 is deleted, so this operation would simply undo the previous insertion operation. To see progress, $f_1 2.1$ should be applied. The corresponding insertion rule $(f_1, A_1, B_1)_D$ is deleting some A_1 left to some B_1, but notice that after a successful application of this rule, f_1 will be situated left of B_1, which is exactly the situation expected when moving the string into $C3$; the rule $(f_1, B_1, \lambda)_D$ is only applicable when, in the original string f_1 was placed immediately to the left of the substring A_1B_1, which has now been deleted (or, in a sense, replaced by f_1). We also say that the applications of rules $f_1 2.1$ and $f_1 3.1$ are *guarded* by the occurrence of the rule marker f_1. Notice that after successfully deleting A_1B_1 in the simulation, we can now either delete f_1 from the

Table 2 Path-structured GCID systems of size (3; 1, 1, 0; 1, 1, 1) simulating type-0 grammars G in SGNF

Component C1	Component C2	Component C3
$r1.1 : (1, (X, r, \lambda)_I, 2)$	$r2.1 : (2, (\lambda, X, r)_D, 1)$	$r3.1 : (3, (r', Y_1, \lambda)_I, 2)$
$r1.2 : (1, (r, r', \lambda)_I, 2)$	$r2.2 : (2, (\lambda, r, r')_D, 1)$	
$r1.3 : (1, (r', \Delta, \lambda)_I, 1)$	$r2.3.c : (2, (Y_2, \Delta, c)_D, 3)$	
$r1.4 : (1, (r', Y_2, \lambda)_I, 2)$	$r2.4.c' : (2, (c', r', Y_1)_D, 1)$	
$f1.1 : (1, (\lambda, f, \lambda)_I, 2)$	$f2.1 : (2, (f, A, B)_D, 3)$	$f3.1 : (3, (f, B, \lambda)_D, 2)$
	$f2.2 : (2, (\lambda, f, \lambda)_D, 1)$	
$h1.1 : (1, (\lambda, S', \lambda)_D, 1)$		
$\kappa 1.1 : (1, (\lambda, \kappa, \lambda)_D, 1)$		
$\kappa' 1.1 : (1, (\lambda, \kappa', \lambda)_D, 1)$		

In the table, $c' \in N'' \cup T \cup \{\kappa'\}$, $c \in N'' \cup T \cup \{\kappa\}$, f, r are rule markers, while Δ is a dummy symbol that was not part of the alphabet of G

string and move it back to $C1$ in order to finish the simulation, or simply re-start by applying $f_1 2.1$ if this is possible.

To simplify the presentation and proofs of our further results, the following observations from [11] are used.

Proposition 1 [11] *Let k, n, i', i'', m, j, j'' be non-negative integers. The following statements are true.*

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

3.1 GCID systems with insertion and deletion length one

In [33], 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 [21,28] that 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 [11], we characterized RE by path-structured GCID systems of size (3; 1, 1, 1; 1, 1, 0). Also, in [16], it was shown that $\text{GCID}_P(3; 1, 2, 0; 1, 1, 0) = \text{RE}$ and $\text{GCID}_P(3; 1, 1, 0; 1, 2, 0) = \text{RE}$. We now complement these results.

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

Before giving the full proof, we sketch some basic features of the construction, as displayed in Table 2. (i) We make use of a dummy symbol Δ to rectify a control graph structure that might be otherwise not a path, as if $r1.3$ is not available, then, $r2.3.c$ could be replaced by $r1.4 : (1, (r', Y_2, \lambda)_I, 3)$, directly moving to $C3$. (Here, symbols r and r' serve as markers, specific to the simulation of the rule labeled r.) (ii) We use explicit left-end and right-end markers κ' and κ, respectively, to rule out some premature rule applications in $C2$. (iii) We return to $C1$ several times when simulating the application of a context-free rule r. This makes the proof that no malicious derivations are possible rather delicate, as many cases have to be considered.

At a first glance, the reader might wonder that the simulation should be straightforward (as initially thought by the authors themselves, as there are that many resources available). However, this is not the case. The problem is that any rule of a component could be applied whenever a string enters that component. Since insertion is only left-context-sensitive, the insertion string can be adjoined any number of times on the right of this context, similar to context-free insertion. This issue is handled by inserting some markers and then inserting Y_1 and Y_2 (from rule $X \to Y_1Y_2$) after the markers. We have to be careful, since a back-and-forth transition may insert many Y_1's and/or Y_2's after the marker.

Proof Consider a type-0 grammar $G = (N, T, P, S)$ in SGNF as in Def. 1. The rules of P are assumed to be labelled bijectively with labels from the set $[1 \ldots |P|]$. We construct a graph-controlled insertion–deletion system Π as follows such that $L(\Pi) = L(G)$: $\Pi = (3, V, T, \{\kappa' S \kappa\}, H, 1, 1, R)$. The alphabet of Π is $V \subset N \cup T \cup \{r, r' : r \in [1 \ldots |P|]\} \cup \{\kappa', \kappa\}$. The simulation rules for different cases are specified in Table 2, which completes the description of R and V. This table should be read as follows.

- If G contains a context-free rule of the form $r = X \to Y_1Y_2$ (with the special linearity properties derived from G being in SGNF), then r is simulated by the rules whose labels are prefixed with r.
- Similarly, the two genuinely context-sensitive rules $A_1B_1 \to \lambda$ and $A_2B_2 \to \lambda$ are simulated as described by the GCID rules prefixed with f.
- $h.1.1$ takes care of simulating the only context-free deletion rule, and $\kappa 1.1$ and $\kappa' 1.1$ take care of the boundary markers.

Clearly, Π has size $(3; 1, 1, 0; 1, 1, 1)$. With the rules of Table 2, we prove $L(G) \subseteq L(\Pi)$ by showing how the different types of rules are simulated. Let us look into the context-free rules first. The simulation of the deletion rule h is obvious and hence omitted. Applying some rule $r : X \to Y_1Y_2$, with $X \in N'$, to $w = \alpha X \beta$, where $\alpha, \beta \in (N'' \cup T)^*$, yields $w' = \alpha Y_1 Y_2 \beta$ in G. We assume that $\alpha' c' = \kappa' \alpha$ and $c\beta' = \beta\kappa$ where $\alpha'\kappa, \kappa' c\kappa, \kappa' c'\kappa, \kappa'\beta' \in \{\kappa'\}(N'' \cup T)^*\{\kappa\}$. The stated equality is important in the following sense: if $\kappa'\alpha = \kappa'\alpha_1\alpha_2 \ldots \alpha_n$ then we set $\alpha' = \kappa'\alpha_1\alpha_2 \ldots \alpha_{n-1}$ and $c' = \alpha_n$. If $\alpha = \lambda$, then $c' = \kappa'$. Hence $c' \in N'' \cup T \cup \{\kappa'\}$. Similarly, when $\beta\kappa = \beta_1\beta_2 \ldots \beta_m\kappa$, we set $\beta' = \beta_2\beta_3 \ldots \beta_m\kappa$ and $c = \beta_1$. If $\beta = \lambda$, then $c = \kappa$. So $c \in N'' \cup T \cup \{\kappa\}$. The main aim to assume the equality is to single out the last symbol of $\kappa'\alpha$ and the first symbol of $\beta\kappa$ and use the singled out symbol as left or right context in the rules $r2.3.c$ or $r2.4.c'$; see Table 2.

In Π, we can find the following simulation:

$$
\begin{array}{llll}
(\kappa' w \kappa)_1 & \Rightarrow_{r1.1} (\alpha' c' X r c \beta')_2 & \Rightarrow_{r2.1} (\alpha' c' r c \beta')_1 & \Rightarrow_{r1.2} (\alpha' c' r r' c \beta')_2 \\
& \Rightarrow_{r2.2} (\alpha' c' r' c \beta')_1 & \Rightarrow_{r1.3} (\alpha' c' r' \Delta c \beta')_1 & \Rightarrow_{r1.4} (\alpha' c' r' Y_2 \Delta c \beta')_2 \\
& \Rightarrow_{r2.3.c} (\alpha' c' r' Y_2 c \beta')_3 & \Rightarrow_{r3.1} (\alpha' c' r' Y_1 Y_2 c \beta')_2 & \Rightarrow_{r2.4.c'} (\alpha' c' Y_1 Y_2 c \beta')_1 \\
& = \quad (\kappa' \alpha Y_1 Y_2 \beta \kappa)_1 & = \quad (\kappa' w' \kappa)_1 \,. &
\end{array}
$$

For the non-context-free case, the simulation of $f : AB \to \lambda$ is straightforward; hence, details are omitted. By induction, this proves that whenever $S \Rightarrow^* w$ in G, then there is a derivation $(\kappa' S \kappa)_1 \Rightarrow'_* (\kappa' w \kappa)_1$ in Π, and finally $(\kappa' w \kappa)_1 \Rightarrow' (w)_1$ is possible.

We now show $L(\Pi) \subseteq L(G)$; this is crucial since it proves that Π not only produces intended strings but neither produces any unintended ones.

Conversely, consider a configuration $(w)_1$, with $(\kappa' S \kappa)_1 \Rightarrow'_* (w)_1$. We assume now that w starts with κ' and ends with κ, and that these are the only occurrences of these special letters in w. As no rules ever introduce these letters, there cannot be any other occurrences of these letters in w. If w contains no occurrence of κ or κ', still further derivations might be

possible, as the only purpose of these two special letters is to allow for tests as in rules $r2.3$ and $r2.4$. The only danger is that such derivations might get stuck when starting with strings w that do not contain κ or κ', but this causes no problems, as there is always the possibility to circumvent this blocking by keeping κ and κ' in until the very end.

We now discuss five situations for w and prove in each case that, whenever $(w)_1 \Rightarrow' (w')_1$, then w' satisfies one of these five situations, or from $(w')_1$ no final configuration can be reached. Inductively, the claim follows.

As $S \in N'$, the base case $\kappa' S \kappa$ is covered in case (iii) presented below.

(i) Assume that w contains one occurrence of r' (the primed marker of some context-free rule r), but no occurrence of unprimed markers of context-free rules, and no occurrence of any nonterminal from N', neither an occurrence of Δ. Hence, $w = \kappa' \alpha r' \beta \kappa$ for appropriate strings $\alpha, \beta \in (N'' \cup T)^*$.

Then, the rules (i.a) $r1.3$, (i.b) $r1.4$, as well as the simulation initiation rules like (i.c) $f1.1$ are applicable. Let us discuss these possibilities now.

Subcase (i.c): If $f1.1$ is applied, then, say, f is introduced to the right of some occurrence of A. In $C2$, one can then try to apply (i.c.1) $f2.1$, (i.c.2) $f2, 2$, or (i.c.3) $r2.4.c'$ for an appropriate c'. However, as we are still simulating phase I of G, B cannot be to the right of A, so that Subcase (i.c.1) cannot occur.

Subcase (i.c.2) simply undoes the effect of previously applying $f1.1$, so that we can ignore its discussion. In Subcase (i.c.3), we are back in $C1$ with a string that contains no symbols from N', nor any variants of context-free rule markers, nor any Δ, but one non-context-free rule marker. We will discuss this in Case (v) below, proving that such a derivation cannot terminate.

Subcase (i.b): If we apply $r1.4$ to w immediately, we are trapped in $C2$.

Subcase (i.a): we apply $r1.3$ first once. Now, we are in a very similar situation as before, but one Δ is added to the right of r'. This means that continuing with $f1.1$ will get stuck again in $C2$. In order to make progress, we should finally apply $r1.4$. Now, we are in the configuration $(\kappa' \alpha r' Y_2 \Delta^n \beta \kappa)_2$ for some $n \geq 1$. As $Y_1 \neq Y_2$, $r2.4.c'$ is not applicable for any c', so the derivation is stuck in $C2$. If we apply $r.2.3.c$, then we can only proceed if $n = 1$, which means that we applied $r1.3$ exactly once before. Hence, $(\kappa' \alpha r' Y_2 \Delta \beta \kappa)_2 \Rightarrow (\kappa' \alpha r' Y_2 \beta \kappa)_3 \Rightarrow (\kappa' \alpha r' Y_1 Y_2 \beta \kappa)_2 \Rightarrow (\kappa' \alpha Y_1 Y_2 \beta \kappa)_1$ is enforced. This corresponds to the intended derivation; the assumed occurrence of r' in the string was replaced by $Y_1 Y_2$; this corresponds to the situation of Case (iii).

(ii) Assume that w contains one occurrence of r (the unprimed marker of some context-free rule r), but no occurrence of primed markers of context-free rules, and no occurrence of any nonterminal from N', neither an occurrence of Δ. Hence, $w = \kappa' \alpha r \beta \kappa$ for appropriate strings $\alpha, \beta \in (N'' \cup T)^*$.

Similarly as discussed in Case (i), trying to start a simulation of some non-context-free rule gets stuck in $C2$, in particular, as we are simulating phase I of G and there is no nonterminal from N' in the current string. Hence, we are now forced to apply $r1.2$. This means that in $C2$, we have to apply $r2.2$, leading us to $(w')_1$ with $w' = \alpha r' \beta$, a situation already discussed in Case (i).

(iii) Assume that w contains one occurrence $X \in N'$, but no occurrence of unprimed or primed markers of context-free rules, and no occurrence of Δ. Hence, $w = \kappa' \alpha X \beta \kappa$ for appropriate strings $\alpha, \beta \in (N'' \cup T)^*$.

As we are still simulating phase I of G, we are now forced to apply $r1.1$ or simulate the context-free deletion rule (which gives a trivial discussion that is omitted; the important point is that this switches to phase II of the simulation of G). This means that in $C2$, we have to

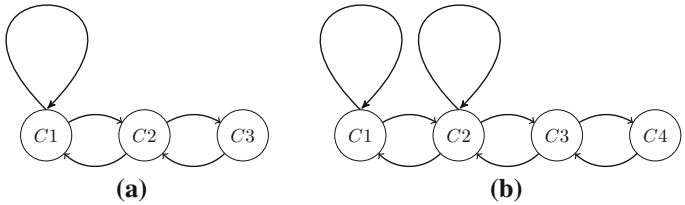

Fig. 1 Control graphs underlying the GCID systems (characterizing RE) in this paper. **a** Control graph of Theorems 1, 5, 6, **b** control graph of Theorems 2, 3

apply $r2.1$, leading us to $(w')_1$ with $w' = \kappa'\alpha r\beta\kappa$ for some context-free rule $r : X \to Y_1Y_2$, a situation already pondered in Case (ii).
(iv) Assume that $w \in \{\kappa'\}(N'' \cup T)^*\{\kappa\}$. Now, it is straightforward to analyze that we have to follow the simulation of one of the non-context-free deletion rules, or finally apply the rule deleting the special symbols κ, κ'.
(v) Assume that w contains no primed or unprimed markers of context-free rules, nor a symbol from N', nor any Δ but contains a non-context-free rule marker. This means we have to apply some rule $f1.1$, but although this might successfully simulate a non-context-free deletion rule, it will bring us back to $C1$ with a non-context-free rule marker in the string. Hence, we are back in Case (v), so that this type of derivation can never terminate.

The second claim follows by Proposition 1. The underlying graph of the simulation is shown in Fig. 1a. The corresponding undirected graph is a path and hence the presented GCID system is path-structured. □

In [13], it was shown that GCID systems of sizes (4; 1, 1, 0; 1, 1, 0) and (4; 1, 1, 0; 1, 0, 1) describe RE, with the underlying control graph not being a path. In [11], the number of components was reduced from 4 to 3, however, with the underlying graph still not being a path. In the next two theorems we characterize RE by path-structured GCID systems of sizes (4; 1, 1, 0; 1, 1, 0) and (4; 1, 1, 0; 1, 0, 1). The former result also complements an earlier result of [16], which stated that $\mathrm{GCID}_P(3; 1, 2, 0; 1, 1, 0) = \mathrm{GCID}_P(3; 1, 1, 0; 1, 2, 0) =$ RE. We trade-off the number of components against the length of the left context of the insertion/deletion.

Theorem 2 $\mathrm{RE} = \mathrm{GCID}_P(4; 1, 1, 0; 1, 1, 0) = \mathrm{GCID}_P(4; 1, 0, 1; 1, 0, 1)$.

Proof Consider a type-0 grammar $G = (N, T, P, S)$ in SGNF as in Def. 1. The rules of P are assumed to be labelled bijectively with labels from the set $[1 \ldots |P|]$. We construct a graph-controlled ins–del system Π as follows such that $L(\Pi) = L(G)$. The alphabet of Π is $V \subset N \cup T \cup \{p, p', p'', p''' : p \in [1 \ldots |P|]\} \cup \{\kappa\}$. The set of rules R (of Π) is defined as shown in Table 3. The four columns of Table 3 correspond to the four components of Π. The rows correspond to the simulation of $r : X \to Y_1Y_2$, $f : AB \to \lambda$ and of the context-free deletion rule $h : S' \to \lambda$. The last row deletes the left-end marker κ introduced in the axiom.

Clearly, Π has size (4; 1, 1, 0; 1, 1, 0). We now prove that $L(G) \subseteq L(\Pi)$. To this end, we show that if $w \Rightarrow w'$ in G, with $w, w' \in (N \cup T)^*$, then $(\kappa w)_1 \Rightarrow' (\kappa w')_1$ according to Π. From this fact, the claim follows by a simple induction argument, because finally the left-end marker κ can be removed. As the claim is evident for rule h, we only need to discuss $w \Rightarrow w'$ due to using a context-free rule (Case CF) or due to using a non-context-free rule (Case $\overline{\mathrm{CF}}$).

Table 3 GCID rules of size (4; 1, 1, 0; 1, 1, 0) with axiom κS and $c \in N'' \cup T \cup \{\kappa\}$

Component C1	Component C2	Component C3	Component C4
$r1.1 : (1, (\lambda, r, \lambda)_I, 2)$	$r2.1 : (2, (X, r', \lambda)_I, 3)$	$r3.1 : (3, (r, X, \lambda)_D, 4)$	$r4.1 : (4, (r', r'', \lambda)_I, 3)$
	$r2.2 : (2, (r, r', \lambda)_D, 2)$	$r3.2 : (3, (r'', r''', \lambda)_I, 2)$	$r4.2 : (4, (\lambda, r, \lambda)_D, 3)$
	$r2.3 : (2, (r, r'', \lambda)_D, 3)$	$r3.3 : (3, (r''', Y_2, \lambda)_I, 4)$	
	$r2.4.c : (2, (c, r''', \lambda)_D, 1)$	$r3.4 : (3, (r''', Y_1, \lambda)_I, 2)$	
$f1.1 : (1, (\lambda, f, \lambda)_I, 2)$	$f2.1 : (2, (A, f', \lambda)_I, 3)$	$f3.1 : (3, (f', B, \lambda)_D, 4)$	$f4.1 : (4, (f, A, \lambda)_D, 3)$
	$f2.2 : (2, (\lambda, f, \lambda)_D, 1)$	$f3.2 : (3, (f, f', \lambda)_D, 2)$	
$h1.1 : (1, (\lambda, S', \lambda)_D, 1)$			
$\kappa 1.1 : (1, (\lambda, \kappa, \lambda)_D, 1)$			

Case CF: The intended simulation works as follows:

$$\begin{aligned}(\kappa\alpha X\beta)_1 &\Rightarrow_{r1.1} (\kappa\alpha r X\beta)_2 \quad \Rightarrow_{r2.1} (\kappa\alpha r X r'\beta)_3 \quad \Rightarrow_{r3.1} (\kappa\alpha r r'\beta)_4\\ &\Rightarrow_{r4.1} (\kappa\alpha r r' r''\beta)_3 \quad \Rightarrow_{r3.2} (\kappa\alpha r r' r'' r'''\beta)_2 \Rightarrow_{r2.2} (\kappa\alpha r r'' r'''\beta)_2\\ &\Rightarrow_{r2.3} (\kappa\alpha r r'''\beta)_3 \quad \Rightarrow_{r3.3} (\kappa\alpha r r''' Y_2\beta)_4 \quad \Rightarrow_{r4.2} (\kappa\alpha r r''' Y_2\beta)_3\\ &\Rightarrow_{r3.4} (\kappa\alpha r''' Y_1 Y_2\beta)_2 \Rightarrow_{r2.4.c} (\kappa\alpha Y_1 Y_2\beta)_1.\end{aligned}$$

Here, c is the last symbol of $\kappa\alpha$, possibly κ.
Case $\overline{\text{CF}}$: Let us consider $f : AB \to \lambda$. This means that $w = \alpha AB\beta$ and $w' = \alpha\beta$ for some $\alpha, \beta \in (N \cup T)^*$. Within Π, this can be simulated as follows.

$$\begin{aligned}(\kappa w)_1 = \quad & (\kappa\alpha AB\beta)_1 \quad \Rightarrow_{f1.1} (\kappa\alpha f AB\beta)_2 \Rightarrow_{f2.1} (\kappa\alpha f Af' B\beta)_3\\ &\Rightarrow_{f3.1} (\kappa\alpha f Af'\beta)_4 \Rightarrow_{f4.1} (\kappa\alpha f f'\beta)_3 \quad \Rightarrow_{f3.2} (\kappa\alpha f\beta)_2\\ &\Rightarrow_{f2.2} (\kappa w')_1.\end{aligned}$$

The converse inclusion $L(\Pi) \subseteq L(G)$ is following an inductive argument as in the previous theorem. Let us consider a configuration $(w)_1$. As $\kappa 1.1$ can be applied at any time in such a configuration, but as the only purpose of having this new symbol κ is to make sure that there is always a symbol to the left of the current deletion position, we can assume from now on that w starts with κ, and as no rule will ever introduce κ, we can also assume that this is the only place where κ occurs in w. Hence, we will not discuss $\kappa 1.1$ any further from now on.

We will now discuss $(w)_1 \Rightarrow' (w')_1$ for different cases for w, proving that, whenever the derivation may lead to some final configuration, then w' is falling into one of the cases that we discussed, which proves the validity of the overall discussion by induction.
(i) If w contains no occurrence of any symbol from N' and no (variants of) rule markers, i.e., $w \in \{\kappa\}(N'' \cup T)^*$, then applying $r1.1$ to w for any applicable context-free rule r will see no continuation of the derivation in $C2$. Therefore, we have to use rule $f1.1$. Due to the fact that all rules are guarded in $C2$, $(w)_1 \Rightarrow_{f1.1} (w_1)_2 \Rightarrow (w_2)_s$ either means that $s = 1$ and $w_2 = w$ (by applying $f2.2$), which means that we observe no progress, or we used $f2.1$ so that $s = 3$. In the latter case, we can continue with $f3.1$ only in $C3$, and in $C4$, we have to pick $f4.1$. By doing so, we have verified the following selections on inserting f and f' into w before: (a) f has been inserted to the left of an occurrence of A; (b) f' has been inserted in between an occurrence of A and an occurrence of B. Upon returning to $C3$, we are in a configuration $(\hat{w})_3$ where, in comparison to w, $\hat{w}$ is obtained by applying the two rewriting rules $A \to f$ and $B \to f'$. We could now either apply $f3.1$ (i.b.1) or $f3.2$ (i.b.2). In the former case (i.b.1), we actually perform a loop that, generalizing what we said above,

leads to deleting a subword A^n to the right of f, as well as a subword B^n to the right of f', upon re-entering $C3$. If now $f3.2$ is applied, then this verifies that in fact a subword of the form $A^n B^n$ for some $n \geq 1$ was deleted from w. Notice that this also captures Case (i.b.2). Now, we return to $C1$ with a string that satisfies the conditions of string w_1 discussed at the beginning of Case (i), so that we now either return to $C1$ with a string w' that was obtained from w by deleting the subword $A^n B^n$, or we restart the derivation as discussed with $(w_2)_3$ above.

(ii) and (iii): If w contains exactly on occurrence of a symbol from N', i.e., $w \in \{\kappa\}(N'' \cup T)^* N'(N'' \cup T)^*$, then again we have the opportunity to apply some rule $r1.1$, belonging to some context-free rule r (Case (ii)), or to apply some rule designed to start the simulation of a non-context-free rule, say, of f (Case (iii)).

In Case (ii), it might be that the context-free deletion rule is directly simulated, i.e., $S' \in N'$ is deleted. In that case, $(w)_1 \Rightarrow (w')_1$ directly corresponds to some rule application of G, i.e., $w \Rightarrow w'$ in G is true. Otherwise, $(w)_1 \Rightarrow (w_1)_2$, and a rule marker r is randomly inserted into w, i.e., $w_1 \in r \sqcup\!\sqcup w$. This potential source of nondeterministic development is clarified within $C2$ and $C3$. As all rules are guarded, we can continue only if some $X \in N'$ occurs in w that is the left-hand side of some context-free rule $\hat{r}$. So, for some $\alpha \in \{\kappa\}(N'' \cup T)^*$ and $\beta \in (N'' \cup T)^*$, $w = \alpha X \beta$, and if $(w_1)_2 \Rightarrow (w_2)_3$, then $w_2 \in r \sqcup\!\sqcup \alpha X \hat{r}' \beta$. Again, checking all rules in $C3$ reveals that only $r3.1$ can be applicable, i.e., we now know that $w_2 = \alpha r X \hat{r}' \beta$, and if we apply $r3.1$, then the resultant string $w_3 = \alpha r \hat{r}' \beta$ moves to $C4$. If we now applied $r4.2$, then the derivation is stuck in $C3$, as no rule in that component can deal with a primed marker of a context-free rule. Hence, we have to apply $\hat{r}4.1$. Hence, $w_4 = \alpha r \hat{r}' \hat{r}'' \beta$ moves back to $C3$. Only one rule is applicable here, the one that can handle double-primed markers, so that $(w_4)_3 \Rightarrow (w_5)_2$ means we have applied $\hat{r}3.2$. Hence, $w_5 = \alpha r \hat{r}' \hat{r}'' \hat{r}''' \beta$. In particular as $\hat{r}''$ is immediately to the left of $\hat{r}'''$, rule $\hat{r}2.4.c$ is not applicable for any c. So, the only applicable rule is $r2.2$, assuming that $\hat{r} = r$. (The case $\hat{r} \neq r$ has no continuation.) Hence, $w_5 = \alpha r r' r'' r''' \beta$, and $(w_5)_2 \Rightarrow (w_6)_2 \Rightarrow (w_7)_3$ with $w_6 = \alpha r r'' r''' \beta$ and $w_7 = \alpha r r''' \beta$ is the only possible continuation. Now, assume that $r : X \to Y_1 Y_2$ is the context-free rule whose markers occur in w_7. Only $r3.3$ is applicable now, followed by $r4.2$ and $r3.4$. This means we observe $(w_7)_3 \Rightarrow (w_8)_4 \Rightarrow (w_9)_3 \Rightarrow (w_{10})_2$, with $w_{10} = \alpha r''' Y_1 Y_2 \beta$. Thus, $(w_{10})_2 \Rightarrow (w')_1$ such that $w \Rightarrow w'$ in G by applying r.

In Case (iii), we can follow a similar reasoning as in Case (i), which means that we would faithfully simulate applications of context-sensitive deletion rules, except for one possible deviation: upon reaching $C2$, we might now apply $r2.1$ for some suitable context-free rule r. This can happen each time when we arrive at $C2$, so we have to separately discuss these possibilities. The first such opportunity is given if we start with, say, $f1.1$ and if we apply $r2.1$ next for some context-free rule $r : X \to Y_1 Y_2$, where $X \in N'$ is occurring in w. This results in $(w)_1 \Rightarrow (w_1)_2 \Rightarrow (w_2)_3$, where $w_2 \in f \sqcup\!\sqcup \alpha X r' \beta$, assuming $w = \alpha X \beta$. It is easy to check that this derivation gets stuck in $C3$. The second possibility to deviate from (i) is when we return to $C2$ later. As discussed in Case (i), the effect of a derivation using f-simulation rules only, ending up in $C2$ and starting in configuration $(w)_1$ amounts in replacing some subword $A^n B^n$, for any $n \geq 0$, by the rule marker f. Notice that the case $n = 0$ corresponds to a random insertion of f, a case that was already discussed above. This derivation gets stuck in $C3$ for each n when trying to apply $r2.1$ instead of $f2.1$ or $f2.2$.

Hence, in each of the cases (i), (ii), (iii), whenever $(w)_1 \Rightarrow (w')_1$, then $w \Rightarrow^* w'$ in G. Also, if we start with a string w satisfying one of the conditions described in these cases, then w' satisfies one of these cases, as well. As Case (ii) corresponds to the start situation, we can conclude by induction that, whenever $(\kappa S)_1 \Rightarrow_* (v)_1$ for some $v \in T^*$, then $S \Rightarrow^* w$

Table 4 GCID rules of size (4; 1, 1, 0; 1, 0, 1) simulating a type-0 grammar in SGNF

Component C1	Component C2	Component C3	Component C4
$r1.1 : (1, (X, r, \lambda)_I, 2)$	$r2.1 : (2, (\lambda, X, r)_D, 1)$	$r3.1 : (3, (\lambda, r'', Y_2)_D, 4)$	$r4.1 : (4, (r', r''', \lambda)_I, 3)$
$r1.2 : (1, (r, r', \lambda)_I, 2)$	$r2.2 : (2, (\lambda, r, r')_D, 1)$	$r3.2 : (3, (r''', Y_1, \lambda)_I, 2)$	
$r1.3 : (1, (r', r'', \lambda)_I, 2)$	$r2.3 : (2, (r'', Y_2, \lambda)_I, 3)$		
	$r2.4 : (2, (\lambda, r''', Y_1)_D, 2)$		
	$r2.5 : (2, (\lambda, r', Y_1)_D, 1)$		
$f1.1 : (1, (\lambda, f, \lambda)_I, 2)$	$f2.1 : (2, (A, f', \lambda)_I, 3)$	$f3.1 : (3, (\lambda, A, f')_D, 4)$	$f4.1 : (4, (\lambda, B, f)_D, 3)$
	$f2.2 : (2, (\lambda, f, \lambda)_D, 1)$	$f3.2 : (3, (\lambda, f', f)_D, 2)$	
$h1.1 : (1, (\lambda, S', \lambda)_D, 1)$			

in G. The second claim follows by Proposition 1. The underlying graph of the simulation is shown in Fig. 1b. □

Theorem 3 $\mathrm{RE} = \mathrm{GCID}_P(4; 1, 1, 0; 1, 0, 1) = \mathrm{GCID}_P(4; 1, 0, 1; 1, 1, 0)$.

Proof Consider a type-0 grammar $G = (N, T, P, S)$ in SGNF. The rules of P are assumed to be labelled bijectively with labels from the set $[1 \ldots |P|]$. We construct a graph-controlled ins–del system Π such that $L(\Pi) = L(G)$, with $\Pi = (4, V, T, \{S\}, H, 1, 1, R)$. The alphabet V of Π satisfies $V \subset N \cup T \cup \{p, p', p'', p''' : p \in [1 \ldots |P|]\}$. The set of rules R (of Π) is defined as shown in Table 4. Π has the claimed size. The intended simulation of a context-free rule is as follows.

$$\begin{aligned}(\alpha X\beta)_1 &\Rightarrow_{r1.1} (\alpha Xr\beta)_2 \Rightarrow_{r2.1} (\alpha r\beta)_1 \Rightarrow_{r1.2} (\alpha rr'\beta)_2\\ &\Rightarrow_{r2.2} (\alpha r'\beta)_1 \Rightarrow_{r1.3} (\alpha r'r''\beta)_2 \Rightarrow_{r2.3} (\alpha r'r''Y_2\beta)_3\\ &\Rightarrow_{r3.1} (\alpha r'Y_2\beta)_4 \Rightarrow_{r4.1} (\alpha r'r'''Y_2\beta)_3 \Rightarrow_{r3.2} (\alpha r'r'''Y_1Y_2\beta)_2\\ &\Rightarrow_{r2.4} (\alpha r'Y_1Y_2\beta)_2 \Rightarrow_{r2.5} (\alpha Y_1Y_2\beta)_1.\end{aligned}$$

The intended simulation of a non-context-free rule is as follows.

$$\begin{aligned}(\alpha AB\beta)_1 &\Rightarrow_{f1.1} (\alpha ABf\beta)_2 \Rightarrow_{f2.1} (\alpha Af'Bf\beta)_3 \Rightarrow_{f3.1} (\alpha f'Bf\beta)_4\\ &\Rightarrow_{r4.1} (\alpha f'f\beta)_3 \Rightarrow_{r3.2} (\alpha f\beta)_2 \Rightarrow_{r2.2} (\alpha\beta)_1.\end{aligned}$$

This shows that $L(G) \subseteq L(\Pi)$. The main complication for the correctness proof is the fact that we may return to $C1$ with strings containing rule markers. This brings along a detailed discussion of four different situations for w when considering $(S)_1 \Rightarrow'_* (w)_1 \Rightarrow' (w')_1$ according to Π.

Consider some derivation $(S)_1 \Rightarrow'_* (w)_1$ in Π. Clearly, we can decompose this derivation into $(w_i)_1 \Rightarrow' (w_{i+1})_1$, such that $S = w_0$, $w = w_m$, and the chosen derivation moves exactly the strings $w_0, \ldots, w_m$ into component $C1$. Note that markers are attached when applying rules in $C1$, except the (only) context-free deletion rule. The rule markers introduced in the (non-)context-free rules avoid the interference of the rules among themselves. All rules in each component are guarded by markers and hence a rule can be applied only if the derivations comes with the appropriate marker. Let us make this more precise. Consider a configuration $(w)_1$ such that $(S)_1 \Rightarrow'_* (w)_1$ in Π. Again, we will discuss several cases for a derivation $(w)_1 \Rightarrow' (w')_1$ showing that w' falls under one of these cases whenever the whole discussed derivation may lead to a final configuration. An easy induction argument then shows that this case distinction is complete.

(i) If w contains no symbols from N' nor unprimed or primed markers of context-free rules, then only $f1.1$ is possibly applicable. As all rules in $C2$ and $C3$ are guarded, a successful complete simulation of $f : AB \to \lambda$ is enforced. Observe that also w' contains no symbols from N' nor unprimed or primed markers of context-free rules, but if w had contained an occurrence of a marker of non-context-free rules, then w' would also contain such an occurrence.
(ii) If w contains one occurrence of a primed marker of a context-free rule r, but no symbols from N', nor unprimed markers of context-free rules, then $r1.3$ or $f1.1$ are possibly applicable. On applying $p1.3$, we arrive at $(w_1)_2$ with $r'r''$ as a substring of w_1, i.e., $w_1 = \alpha r'r''\beta$ for some $\alpha, \beta \in (N'' \cup T)^*$. Observe that $r2.5$ is not applicable to this configuration. By the use of rule markers, the rules $r2.3$, $r3.1$, $r4.1$, $r3.2$, $r2.4$, $r2.5$ must be applied in this order, leading to $(w)_1 \Rightarrow' (w')_1$ such that w' is obtained from w by applying the context-free rule r as intended. Alternatively, we could start $(w)_1 \Rightarrow (w_1)_2$ by applying, say, $f1.1$. If we now apply $f2.1$, there is no alternative but applying $f3.1$ in $C3$, which would mean that we end up (on applying $f2.1$ and $f3.1$) simulating a rule application of $AB \to \lambda$. As the resulting string w_1' enjoys the same properties as w_1 in $C2$, let us explore further alternatives for w_1, containing both an occurrence of r' and an occurrence of, say, f. In fact, now $r2.5$ could be applicable, which would yield $(w)_1 \Rightarrow' (w')_1$, where w' now contains no occurrences of symbols from N' nor unprimed or primed markers of context-free rules, but one occurrence of a marker of non-context-free rules. Inductively, to such configurations, the reasoning of case (i) would apply, which means that such derivations can never lead to terminal strings, as whenever $(w')_1 \Rightarrow'_* (w'')_1$, then w'' also contains an occurrence of a marker of non-context-free rules.
(iii) If w contains one occurrence of an unprimed marker of a context-free rule r, but no symbols from N' nor primed markers of context-free rules, then $r1.2$ or $f1.1$ are possibly applicable. On applying, say, $f1.1$, we are now enforced to (correctly) simulate applications of the rule $r : AB \to \lambda$ as described in item (i), as in particular no rules in $C2$ or $C3$ can work with the marker r without r' or a fitting nonterminal from N'. As such simulations lead to configurations in $C2$ that are similar to the ones under current study, we need to explore only $(w)_1 \Rightarrow (w_1)_2$ due to applying $r1.2$. As w_1 contains no occurrence of a nonterminal from N', $r2.2$ must be applied, leading to $(w')_1$, where w' is obtained from w by replacing the only occurrence of the marker r by the marker r'. Additionally, w' inherits from w the properties of not containing further markers of context-free rules nor symbols from N'. Hence, a possible further discussion would continue as in Case (ii).
(iv) If w contains no occurrences of primed or unprimed rule markers but one symbol $X \in N'$, then $r1.1$ or $f1.1$ are possibly applicable, for some context-free rule r including the context-free deletion rule. Hence, $w = \alpha X \beta$ with $\alpha, \beta \in (N'' \cup T)^*$. If $(w)_1 \Rightarrow (w')_1$ due to applying $h1.1$, then $w' = \alpha\beta$, and we have to continue the derivation as discussed in Case (i). Otherwise, we might apply $f1.1$. In that case, we have to follow a (correct) simulation of some non-context-free rule f (possibly a number of times) in the arising derivation $(w)_1 \Rightarrow (w')_1$, as there are no rules in $C2$ or $C3$ that can deal with nonterminals from N' without referring to a fitting context-free rule marker (or double- or triple-primed variants thereof). Therefore, we are left with discussing the application of a rule $r1.1$, which starts the simulation of a context-free rule r with left-hand side X. In that case, $(w)_1 \Rightarrow (w_1)_2$ with $w_1 = \alpha X r \beta$. Now, due to the use of rule markers, the application of $r2.1$ is enforced, leading to the configuration $(w')_1$ with $w' = \alpha r \beta$, i.e., w' was obtained from w by replacing X by r. Now, the discussion continues as in Case (iii).

The observations above on the non-interference of context-free and non-context-free rule simulations shows that no malicious derivations are possible. Hence, $L(\Pi) \subseteq L(G)$.

Table 5 GCID rules of size (4; 2, 0, 0; 1, 1, 0) simulating a type-0 grammar in SGNF

Component C1	Component C2	Component C3	Component C4
$r1.1 : (1, (\lambda, rr', \lambda)_I, 2)$	$r2.1 : (2, (\lambda, Y_1 r'', \lambda)_I, 3)$	$r3.1 : (3, (r'', X, \lambda)_D, 4)$	$r4.1 : (4, (r'', r, \lambda)_D, 3)$
	$r2.2 : (2, (\lambda, Y_2 r''', \lambda)_I, 3)$	$r3.2 : (3, (r'', r', \lambda)_D, 2)$	
	$r2.3 : (2, (\lambda, r''', \lambda)_D, 1)$	$r3.3 : (3, (r''', r'', \lambda)_D, 2)$	
$f1.1 : (1, (\lambda, f, \lambda)_I, 2)$	$f2.1 : (2, (\lambda, f', \lambda)_I, 3)$	$f3.1 : (3, (f', A, \lambda)_D, 4)$	$f4.1 : (4, (f', B, \lambda)_D, 3)$
	$f2.2 : (2, (\lambda, f, \lambda)_D, 1)$	$f3.2 : (3, (\lambda, f', \lambda)_D, 2)$	
$h1.1 : (1, (\lambda, S', \lambda)_D, 1)$			

The second claim follows by Proposition 1. The underlying graph of the simulation is shown in Fig. 1b; obviously, it is a path.

3.2 GCID systems with insertion length two

In [13], it is shown that GCID(4; 2, 0, 0; 1, 1, 0) = RE with the underlying control graph not even being a tree. In this subsection, we show that, even if we restrict the control graph to be a path, $\mathrm{GCID}_P(4; 2, 0, 0; 1, 1, 0)$ systems will still characterize RE. Further, if we allow a context (either left or right) for insertion, then we can still describe RE while decreasing the number of components from 4 to 3, yet obtaining path-structured GCID systems.

Notice that we do need some context somewhere, as it is known that a language as simple as $\{a^n b \mid n \geq 1\}$ cannot be described by any GCID_P system of size $(k; 2, 0, 0; 2, 0, 0)$ for any k; see [22, Thm. 17].

Theorem 4 $\mathrm{RE} = \mathrm{GCID}_P(4; 2, 0, 0; 1, 1, 0) = \mathrm{GCID}_P(4; 2, 0, 0; 1, 0, 1)$.

Proof Consider a type-0 grammar $G = (N, T, P, S)$ in SGNF. The rules of P are assumed to be labelled bijectively with labels from the set $[1 \ldots |P|]$. We construct a graph-controlled insertion–deletion system Π as follows such that $L(\Pi) = L(G)$: $\Pi = (4, V, T, \{S\}, H, 1, 1, R)$. The alphabet of Π is $V \subset N \cup T \cup H$ where $H = \{r, r', r'', r''' : r \in [1 \ldots |P|]\}$. The set of rules R (of Π) is given in Table 5.
The intended derivation of a context-free rule $r : X \to Y_1 Y_2$ is as follows.

$$\begin{aligned}(\alpha X \beta)_1 &\Rightarrow_{r1.1} (\alpha X r r' \beta)_2 \Rightarrow_{r2.1} (\alpha Y_1 r'' X r r' \beta)_3 \Rightarrow_{r3.1} (\alpha Y_1 r'' r r' \beta)_4 \\ &\Rightarrow_{r4.1} (\alpha Y_1 r'' r' \beta)_3 \Rightarrow_{r3.2} (\alpha Y_1 r'' \beta)_2 \Rightarrow_{r2.2} (\alpha Y_1 Y_2 r''' r'' \beta)_3 \\ &\Rightarrow_{r3.3} (\alpha Y_1 Y_2 r''' \beta)_2 \Rightarrow_{r2.3} (\alpha Y_1 Y_2 \beta)_1.\end{aligned}$$

The intended derivation of a non-context-free deletion rule is as follows.

$$\begin{aligned}(\alpha A B \beta)_1 &\Rightarrow_{f1.1} (\alpha f A B \beta)_2 \Rightarrow_{f2.1} (\alpha f f' A B \beta)_3 \Rightarrow_{f3.1} (\alpha f f' B \beta)_4 \\ &\Rightarrow_{f4.1} (\alpha f f' \beta)_3 \Rightarrow_{f3.2} (\alpha f \beta)_2 \Rightarrow_{f2.2} (\alpha \beta)_1.\end{aligned}$$

This show that $L(G) \subseteq L(\Pi)$. Conversely, consider some derivation $(S)_1 \Rightarrow'_* (w)_1 \Rightarrow' (w')_1$ in Π. By induction, assume that w contains at most one symbol from N' and no (unprimed or primed variants of) rule markers. In the following, let $r : X \to Y_1 Y_2$, $s : X' \to Y'_1 Y'_2$ and $t : X'' \to Y''_1 Y''_2$ be three (not necessarily distinct) context-free rules, with the corresponding markers r, s and t, while $f : AB \to \lambda$ is a non-context-free deletion rule. Notice (*) that whenever we discuss a configuration $(u)_2$, we could (in principle) always paste in a (correct) simulation of some non-context-free deletion rule f, leading to $(u')_2$ with $u \Rightarrow_f u'$.

First scenario: $(w)_1 \Rightarrow_{r1.1} (w_1)_2 \Rightarrow_{s2.1} (w_2)_3$. Hence, first rr' and then $Y_1's''$ have been introduced into w to obtain w_2. The only rule in $C3$ that is potentially applicable is $s3.1$, which deletes the only occurrence from N' from w_2 (which hence also checks that there is exactly one such occurrence in w, as otherwise no rule in $C3$ is applicable, because w contains no primed rule markers), so that the configuration $(w_3)_4$ can be described as being obtained from $w = \alpha X'\beta$ by inserting rr' into $\alpha Y_1's''\beta$. Now in $C4$, it is checked if $r = s$, as only in that case and if the previous insertions were carried out in the right places, be can arrive at $(w_4)_3$ with $w_4 = \alpha Y_1 r'' r' \beta$. As $r'' r'$ is a substring of w_4, the only applicable rule in $C3$ would be $r3.2$. This enforces the subsequent configuration $(w_5)_2$ with $w_5 = \alpha Y_1 r''\beta$.

According to (*), we could now digress and paste in several (correct) simulations of applications of non-context-free deletion rules, which does not interfere with the flow of the argument. In particular, observe that if $(w_5)_2 \Rightarrow_{f2.1} (x)_3$, then $(x)_3 \Rightarrow_{f3.1} (y)_4$ is enforced, ignoring the back-loop $(x)_3 \Rightarrow_{f3.2} (w_5)_2$.

Notice that we also might try to start some new simulation of a context-free rule t by $(w_5)_2 \Rightarrow_{t2.1} (x)_3$. In order to continue, we must have $Y_1 = X''$, leading to $(y)_4$ with $y = \alpha t'' r'' \beta$, allowing no further continuation.

So, we are left with discussing an application of $r2.2$, leading to $(w_6)_3$ with $w_6 = \alpha Y_1 Y_2 r''' r'' \beta$, where the correct position of the insertion is tested in $C3$, as otherwise rule is applicable there. This also enforces $(w_6)_3 \Rightarrow_{r3.3} (w_7)_2$, with $w_7 = \alpha Y_1 Y_2 r''' \beta$.

Clearly, by applying $r2.3$, we could now arrive at $(w_8)_1$ with $w \Rightarrow_r w_8$ as desired, where w_8 contains no (primed) rule markers and only one occurrence of a symbol from N', concluding the induction step for this case.

Also, according to (*), we could paste in several (correct) simulations of applications of non-context-free deletion rules, which does not interfere with the overall argument, as completed such simulations are enforced, as discussed above.

Finally, restarting some new simulation of a context-free rule would again get stuck in $C3$ or in $C4$, as described above.

Second scenario: $(w)_1 \Rightarrow_{r1.1} (w_1)_2 \Rightarrow_{s2.2} (w_2)_3$. It is easy to check that now the derivation is stuck in $C3$.

Third scenario: $(w)_1 \Rightarrow_{r1.1} (w_1)_2 \Rightarrow_{f2.1} (w_2)_3$. As noted in (*), we can paste in several correct simulations of non-context-free deletion rules, not changing the main argument of this proof. Notice that no other rules are applicable in $C3$ but $f3.1$ (or $f3.2$, but this makes no progress), and similarly in $C4$.

Fourth scenario: $(w)_1 \Rightarrow_{f1.1} (w_1)_2 \Rightarrow_{r2.1} (w_2)_3$. Now, the only possible continuation is via $(w_2)_3 \Rightarrow_{r3.1} (w_3)_4$, but then the derivation is stuck in $C4$, as w_3 neither contains f' nor a double-primed rule marker left to an unprimed one of the same rule.

Fifth scenario: $(w)_1 \Rightarrow_{f1.1} (w_1)_2 \Rightarrow_{r2.2} (w_2)_3$. This attempted derivation is immediately stuck in $C3$.

Sixth scenario: $(w)_1 \Rightarrow_{f1.1} (w_1)_2 \Rightarrow_{f2.1} (w_2)_3$. As described above, this may now start an intended simulation of a non-context-free deletion rule. Any attempts to intercalate other rules will get stuck, apart from those un-doing the previous step, which can be ignored.

Seventh scenario: $(w)_1 \Rightarrow_{f1.1} (w_1)_2 \Rightarrow_{f2.2} (w_2)_1$. As $w_2 = w$, this scenario can be ignored.

As the reader can easily check, these are all possibilities of derivations, starting with $(w)_1$. This proves that whenever $(w)_1 \Rightarrow' (w')_1$, then $w \Rightarrow^+ w'$ in G, which shows $L(\Pi) \subseteq L(G)$ by induction.

The claimed path structure which can be easily seen by inspection is depicted in Fig. 1b without the loop over $C2$. Proposition 1 shows that GCID_P systems of size (4; 2, 0, 0; 1, 0, 1) are computationally complete. □

Table 6 GCID rules of size (3; 2, 1, 0; 1, 0, 1) simulating a type-0 grammar in SGNF

Component C1	Component C2	Component C3
$r1.1 : (1, (X, r, \lambda)_I, 2)$	$r2.1 : (2, (\lambda, X, r)_D, 3)$	$r3.1 : (3, (r, Y_1Y_2, \lambda)_I, 2)$
	$r2.2 : (2, (\lambda, r, \lambda)_D, 1)$	
$f1.1 : (1, (B, f, \lambda)_I, 2)$	$f2.1 : (2, (\lambda, B, f)_D, 3)$	$f3.1 : (3, (\lambda, A, f)_D, 2)$
	$f2.2 : (2, (\lambda, f, \lambda)_D, 1)$	
$h1.1 : (1, (\lambda, S', \lambda)_D, 1)$		

Theorem 5 $\mathrm{RE} = \mathrm{GCID}_P(3; 2, 1, 0; 1, 0, 1) = \mathrm{GCID}_P(3; 2, 0, 1; 1, 1, 0)$.

Proof Consider a type-0 grammar $G = (N, T, P, S)$ in SGNF as in Def. 1. The rules of P are assumed to be labelled bijectively with labels from $[1 \ldots |P|]$. We construct a GCID_P system $\Pi = (3, V, T, \{S\}, H, 1, 1, R)$ of size (3; 2, 1, 0; 1, 0, 1) as follows such that $L(\Pi) = L(G)$. Here, let $V \subset N \cup T \cup [1 \ldots |P|]$ contain in particular those rule labels used in the rules listed in Table 6. The three columns of the table correspond to the three components of Π. The rows correspond to the rules simulating $r : X \to Y_1Y_2$, $f : AB \to \lambda$ and $h : S' \to \lambda$. Π is of size (3; 2, 1, 0; 1, 0, 1). We now prove that $L(G) \subseteq L(\Pi)$. As the claim is evident for $h : S' \to \lambda$, we show that if $w \Rightarrow w'$ in G, then $(w)_1 \Rightarrow' (w')_1$ according to Π in two more cases.

Case CF: Here, $w = \alpha X \beta$ and $w' = \alpha Y_1 Y_2 \beta$ for some $\alpha, \beta \in (N'' \cup T)^*$. The simulation of $r : X \to Y_1Y_2$ performs as follows:

$$(\alpha X \beta)_1 \overset{\Leftarrow r2.2}{\Rightarrow r1.1} (\alpha X r \beta)_2 \Rightarrow_{r2.1} (\alpha r \beta)_3 \Rightarrow_{r3.1} (\alpha r Y_1 Y_2 \beta)_2 \Rightarrow_{r2.2} (\alpha Y_1 Y_2 \beta)_1 .$$

Note the role of the right context r in the deletion rule $r2.1$. If the marker r is not present for the deletion, then after applying $r3.1$, when we come back to $C2$, we can apply $r2.1$ again and could end-up with a malicious derivation.

Case $\overline{\mathrm{CF}}$: Here $w = \alpha AB\beta$ and $w' = \alpha\beta$ for some $\alpha, \beta \in (N \cup T)^*$. The rules $f : AB \to \lambda$ can be simulated as follows.

$$(\alpha AB \beta)_1 \overset{\Leftarrow f2.2}{\Rightarrow f1.1} (\alpha AB f \beta)_2 \Rightarrow_{f2.1} (\alpha A f \beta)_3 \Rightarrow_{f3.1} (\alpha f \beta)_2 \Rightarrow_{f2.2} (\alpha\beta)_1 .$$

We now prove the converse $L(\Pi) \subseteq L(G)$. Consider some derivation $(S)_1 \Rightarrow'_* (w)_1$ in Π. We claim that then $S \Rightarrow^* w$ in G; in particular, $w \in (N'' \cup T)^*(N' \cup \{\lambda\})(N'' \cup T)^*$. This trivially holds for $w = S$. So, suppose we have some $w \in (N'' \cup T)^*(N' \cup \{\lambda\})(N'' \cup T)^*$ for which $(S)_1 \Rightarrow'_* (w)_1$ in Π, as well as $S \Rightarrow^* w$ in G by induction hypothesis. Let $(w)_1 \Rightarrow' (w')_1$ in Π, so that $(S)_1 \Rightarrow'_* (w')_1$ in Π. As $(w)_1 \Rightarrow' (w')_1$, inspecting the rules tells us that there are various ways in which $w = w'$ is possible. Clearly, then w' enjoys the same properties as w. So, assume $w' \neq w$ from now on. The derivation $(w)_1 \Rightarrow' (w')_1$ has to start like $(w)_1 \Rightarrow (w_1)_j$ in Π. If $j = 1$, then rule $h1.1$ was applied, so in fact $w_1 = w'$, and the effect is just the same as applying $S' \to \lambda$ in G, so that w' satisfies the claim. Otherwise, $j = 2$. Here, there are two different ways to continue, which we are going to discuss next.

Applying $r1.1$ to obtain w_1 from w, for some $r : X \to Y_1Y_2$. Hence, w contains an occurrence of X, and this is also the only place where any symbol from N' can appear by assumption. So, $w = \alpha X \beta$ for some $\alpha, \beta \in (N'' \cup T)^*$ and $w_1 = \alpha X r \beta$ is transferred to $C2$. All rules in $C2$ are guarded by a rule marker. Hence, the only rules applicable now are $r2.1$ and $r2.2$, yielding a string w_2. On applying the latter rule, $w_2 = w$ is returned to $C1$, so that $w' = w$, a case ruled out before. Hence, $w_2 = \alpha r \beta$ is moved to component $C3$.

Table 7 GCID rules of size (3; 2, 1, 0; 1, 1, 0) simulating a type-0 grammar in SGNF

Component C1	Component C2	Component C3
$r1.1 : (1, (\lambda, r, \lambda)_I, 2)$	$r2.1 : (2, (r, X, \lambda)_D, 3)$	$r3.1 : (3, (r, Y_1Y_2, \lambda)_I, 2)$
	$r2.2 : (2, (\lambda, r, \lambda)_D, 1)$	
$f1.1 : (1, (\lambda, f, \lambda)_I, 2)$	$f2.1 : (2, (f, A, \lambda)_D, 3)$	$f3.1 : (3, (f, B, \lambda)_D, 2)$
	$f2.2 : (2, (\lambda, f, \lambda)_D, 1)$	
$h1.1 : (1, (\lambda, S', \lambda)_D, 1)$		

Again, all rules in $C3$ are guarded by rule markers, so that the only applicable rule is $r3.1$. The resulting string $w_3 = \alpha r Y_1 Y_2 \beta$ moves to $C2$. The rule $r2.1$ cannot be applied due to an inappropriate position of r and the only nonterminal of w_3 is after r, however the rule $r2.1$ demands the nonterminal to be on the left of r. Hence, $r2.2$ has to be applied to w_3, yielding $w_4 = \alpha Y_1 Y_2 \beta$ which moves to $C1$, i.e., $w' = w_4$. Obviously, $w \Rightarrow w'$ using rule $r : X \to Y_1 Y_2$ of G. Also, $w' \in (N'' \cup T)^*(N' \cup \{\lambda\})(N'' \cup T)^*$.
Applying $f1.1$ to obtain w_1 from w, for $f : AB \to \lambda$. Hence, w contains some occurrence of B, i.e., $w = \alpha B \beta$, which is the occurrence checked when inserting f, i.e., $w_1 = \alpha B f \beta$ is the string moved to $C2$. Applying $f2.2$ now would mean that $w' = w$, contradicting our assumptions. Hence, $f2.1$ is applied, so that the resulting string $w_2 = \alpha f \beta$ moves to $C3$. Due to the use of rule markers, the only potentially applicable rule is $f3.1$. As the derivation is continued (by assumption of the existence of w'), $\alpha = \alpha' A$ holds. Hence, $w_3 = \alpha' f \beta$ moves to $C2$. Now, if $f2.2$ is applied, $w_4 = \alpha' \beta$ moves to $C1$, i.e., $w' = w_4$. As $w = \alpha B \beta = \alpha' A B \beta$, w' can be obtained from w by applying $AB \to \lambda$, which shows the claim in this case. Recall that w' was obtained by first inserting f into w and then applying the rules $f2.1$, $f3.1$, and $f2.2$ in this order. So, if we would choose to apply $f2.1$ to w_4 (instead of $f2.2$), this means that $\alpha' = \alpha'' B$, and $w_4 = \alpha'' f \beta$. But then,

$$(w_3)_2 = (\alpha' f \beta)_2 \Rightarrow_{f2.2} (\alpha' \beta)_1 = (\alpha'' B \beta)_1 \Rightarrow_{f1.1} (\alpha'' B f \beta)_2 \Rightarrow_{f2.1} (w_4)_3$$

is also possible, so we can arrive at the same situation by first finishing the simulation of one application of $AB \to \lambda$ and then starting another such application. Therefore, using the rule sequence $f1.1$, followed by $(f2.1, f3.1)^m$, and finalized by $f2.2$, in order to obtain w' from w in $C1$, simulates an m-fold application of $f : AB \to \lambda$. Hence, $w \Rightarrow^m w'$ in G and w' satisfies the required properties.

This concludes that proof that $L(G) = L(\Pi)$. Proposition 1 shows that also GCID systems of size (3; 2, 0, 1; 1, 1, 0) are computationally complete. Figure 1a depicts the control graph of the simulation. □

Theorem 6 $\mathrm{RE} = \mathrm{GCID}_P(3; 2, 1, 0; 1, 1, 0) = \mathrm{GCID}_P(3; 2, 0, 1; 1, 0, 1)$.

Proof Consider a type-0 grammar $G = (N, T, P, S)$ in SGNF as in Def. 1. The rules of P are assumed to be labelled bijectively with labels from $[1 \ldots |P|]$. We construct a GCID system $\Pi = (3, V, T, \{S\}, H, 1, 1, R)$ of size (3; 2, 1, 0; 1, 1, 0) as follows such that $L(\Pi) = L(G)$. V again contains rule markers, apart from the symbols of G. We refer to Fig. 7 for the rules of Π. The control graph is shown in Fig. 1a. The three columns of the table correspond to the three components of Π. The rows correspond to the rules simulating $r : X \to Y_1 Y_2$, $f : AB \to \lambda$ and $h : S' \to \lambda$.

We now prove that $L(G) \subseteq L(\Pi)$ as follows. We show that if $w \Rightarrow w'$ in G, then $(w)_1 \Rightarrow' (w')_1$ according to Π. From this fact, the claim follows by a simple induction

argument. $w \Rightarrow w'$ could be due to different rule types, as discussed above. As the correctness is evident for $h : S' \to \lambda$, it remains to discuss two more cases. We first discuss the simulation of context-free rules and then of non-context-free rules.
Context-free rules $r : X \to Y_1Y_2$. Here, $w = \alpha X\beta$ and $w' = \alpha bY\beta$ for some $\alpha, \beta \in (N'' \cup T)^*$. The simulation performs as follows:

$$(\alpha X\beta)_1 \overset{\Leftarrow r2.2}{\Rightarrow r1.1} (\alpha rX\beta)_2 \Rightarrow_{r2.1} (\alpha r\beta)_3 \Rightarrow_{r3.1} (\alpha rY_1Y_2\beta)_2 \Rightarrow_{r2.2} (\alpha Y_1Y_2\beta)_1 .$$

Non-context-free rules $f : AB \to \lambda$. This means that $w = \alpha AB\beta$ and $w' = \alpha\beta$ for some $\alpha, \beta \in (N \cup T)^*$. Within Π, this can be simulated as follows.

$$(\alpha AB\beta)_1 \overset{\Leftarrow f2.2}{\Rightarrow f1.1} (\alpha fAB\beta)_2 \Rightarrow_{f2.1} (\alpha fB\beta)_3 \Rightarrow_{f3.1} (\alpha f\beta)_2 \Rightarrow_{f2.2} (\alpha\beta)_1 .$$

Conversely, a derivation $(w)_1 \Rightarrow' (w')_1$ has to start like $(w)_1 \Rightarrow (w_1)_j$ in Π. If $j = 1$, the applied rule is $h1.1$, then S' is deleted from w to obtain w', and this exactly corresponds to an application of the context-free rule $S' \to \lambda$. More precisely, in the assumed derivation $(w)_1 \Rightarrow' (w')_1$, if some rule from $C1$ (other than $h1.1$) is applied to w, the rule will insert a rule marker into the string w and branch to $C2$. The introduction of rule markers in $C1$ will take care of the non-interference among the non-context-free and context-free rules. We now discuss the possibilities in detail. For $(w)_1 \Rightarrow' (w')_1$, inspecting the rules tells us that there are various ways in which $w = w'$ is possible. Clearly, then w' enjoys the same properties as w. So, assume $w' \neq w$ from now on.

Applying $r1.1$ to $w_1 = (\alpha X\beta)_1$ for $\alpha, \beta \in (N'' \cup T)^*$ and $X \in N'$, the only nonterminal from N', will insert a marker r randomly into the string $w = \alpha X\beta$ yielding $(w_1)_2$. In $C2$, all rules are guarded by markers and the only applicable rules are $r2.1$ and $r2.2$ yielding a string w_2. If the rule $r2.2$ is applied, then $w_2 = w$ is returned to $C1$, so that $w' = w$, a ruled out case. Hence on applying $r2.1$, $w_2 = \alpha r\beta$ is moved to $C3$. Again all rules are guarded by a marker and the only applicable rule is $r3.1$ which inserts Y_1Y_2 into w_2 to yield $w_3 = \alpha rY_1Y_2\beta$ and moves back to $C2$. At this point, the rule $r2.1$ is not applicable since $Y_1 \neq X$ by SGNF. The only other applicable rule to w_3 is $r2.2$ which deletes the marker r in w_3 and yields $\alpha Y_1Y_2\beta = w'$ and moves to $C1$. Also, $w \Rightarrow w'$ using the rule $r : X \to Y_1Y_2$ of G. Note that $w' \in (N'' \cup T)^*N'(N'' \cup T)^*$.

Applying $f1.1$ to $w = \alpha AB\beta$, we obtain a string w_1 by inserting f anywhere within w. w_1 is transferred to component $C2$. If now $f2.2$ is applied, a string w_2 would be moved back to $C1$ that can be described as undoing the insertion of f and this leads to our starting point, ruled out before. So, we can assume that $f2.1$ is applied. To make it applicable, w_1 should be of the form $\alpha fA\beta'$ to produce a string $w_2 = \alpha f\beta'$ that is moved to $C3$. Notice that possible strings w_2 can be described as being obtained from w by replacing some occurrence of A in w by f. Now at $C3$, to make the rule $f3.1$ applicable to $w_2 = \alpha f\beta'$, w_2 should be of the form $\alpha fB\beta$ to produce a string $w_3 = \alpha f\beta$ that is moved back to $C2$. Notice that w_3 can be described as being obtained from w by replacing some occurrence of AB in w by f. If $f2.2$ is applied to w_3, then the marker f is deleted and the resultant string $w_4 = \alpha\beta$ is moved back to $C1$. This means that $(w)_1 \Rightarrow' (w')_1$ implies $w \Rightarrow w'$ in G as required. Recall that w' was obtained by first inserting f into w and then applying the rules $f2.1$, $f3.1$, and $f2.2$ in this order. So, if we would choose to apply $f2.1$ to w_3 (instead of $f2.2$), this means that $\beta = AB\beta''$, and $w_4 = \alpha f\beta''$. But then,

$$(w_3)_2 = (\alpha f\beta)_2 = (\alpha fAB\beta'')_2 \Rightarrow_{f2.1} (\alpha fB\beta'')_3 \Rightarrow_{f3.1} (\alpha f\beta'')_2 = (w_4)_2$$

is also possible, so we can arrive at the same situation by first finishing the simulation of one application of $AB \to \lambda$ and then starting another such application simulation. Therefore,

using the rule sequence $f1.1$, followed by $(f2.1f3.1)^m$, and finalized by $f2.2$, in order to obtain w' from w in $C1$, simulates an m-fold application of $f : AB \to \lambda$. Hence, $w \Rightarrow^m w'$ in G and w' satisfies the required properties.

In particular, once a marker r or f is introduced in $C1$, 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. 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.

The claimed size of Π can be verified by inspecting Fig. 7. Proposition 1 shows that path-structured GCID systems of size (3; 2, 1, 0; 1, 1, 0) are computationally complete. The underlying graph of the simulation can be seen in Fig. 1a and is hence a path.

3.3 Consequences in ins–del P systems

Graph-controlled insertion–deletion systems whose underlying control graph is a path are equivalent to insertion–deletion P systems. The components in the former system correspond to the membranes in the latter and the path structure in the former correspond to the balancedness of the parenthesis (that represent membranes) in the latter. The family of languages generated by ins–del P system with k membranes and size (n, i', i'', m, j', j''), where the size parameters have the same meaning as in GCID system is represented by $\mathrm{ELSP}_k(\mathrm{INS}_n^{i',i''}\mathrm{DEL}_m^{j',j''})$. This notation was used in [16], based on [30]. The results of Theorems 1 to 6 are summarized in the following corollary using the notation of [16,30].

Corollary 1 *For $i', i'', j', j'' \in \{0, 1\}$ with $i' + i'' = j' + j'' = 1$, the following ins–del P systems are computationally complete.*

1. *(Thms 1)* $\mathrm{RE} = \mathrm{ELSP}_3(\mathrm{INS}_1^{i',i''}\mathrm{DEL}_1^{1,1})$.
2. *(Thms 2, 3)* $\mathrm{RE} = \mathrm{ELSP}_4(\mathrm{INS}_1^{i',i''}\mathrm{DEL}_1^{j',j''})$.
3. *(Thms 4)* $\mathrm{RE} = \mathrm{ELSP}_4(\mathrm{INS}_2^{0,0}\mathrm{DEL}_1^{j',j''})$.
4. *(Thms 5, 6)* $\mathrm{RE} = \mathrm{ELSP}_3(\mathrm{INS}_2^{i',i''}\mathrm{DEL}_1^{j',j''})$. □

How the above results improve on or complement the existing results in the domain of ins–del P system or path-structured GCID system is shown in Table 8. It is however open, to discuss one example only, if $\mathrm{ELSP}_3(\mathrm{INS}_2^{0,0}\mathrm{DEL}_1^{1,0})$ equals RE or how $\mathrm{ELSP}_4(\mathrm{INS}_2^{0,0}\mathrm{DEL}_1^{1,0})$ and $\mathrm{ELSP}_3(\mathrm{INS}_1^{1,0}\mathrm{DEL}_2^{0,0})$ relate to each other. Similar questions can be destilled from the subsequent tables, too.

4 Further corollaries and summary

4.1 Consequences in GCID systems

Theorem 7 *(i)* $\mathrm{GCID}(4; 1, 0, 1; 1, 0, 1) = \mathrm{RE}$*; (ii)* $\mathrm{GCID}(4; 1, 0, 1; 1, 1, 0) = \mathrm{RE}$*; (iii);* $\mathrm{GCID}(4; 2, 0, 0; 1, 0, 1) = \mathrm{RE}$*; (iv)* $\mathrm{GCID}_P(4; 1, 0, 1; 2, 0, 0) = \mathrm{RE}$.

Proof In [13], it was proved that GCID systems of sizes (4; 1, 1, 0; 1, 1, 0), (4; 1, 1, 0; 1, 0, 1), (4; 2, 0, 0; 1, 1, 0) and (4; 1, 1, 0; 2, 0, 0) equal RE where the control graph of the system with the last size alone is a path. The theorem follows by Proposition 1. □

Table 8 Results in insertion–deletion P systems

	Results of [22]	Results of this paper	Reference
1.	$\mathrm{ELSP}_5(\mathrm{INS}_1^{1,0}\mathrm{DEL}_1^{1,0})$	$\mathrm{ELSP}_4(\mathrm{INS}_1^{1,0}\mathrm{DEL}_1^{1,0})$	Thm 2
2.	$\mathrm{ELSP}_5(\mathrm{INS}_1^{1,0}\mathrm{DEL}_1^{0,1})$	$\mathrm{ELSP}_4(\mathrm{INS}_1^{1,0}\mathrm{DEL}_1^{0,1})$	Thm 3
3.	$\mathrm{ELSP}_5(\mathrm{INS}_1^{1,0}\mathrm{DEL}_2^{0,0})$	$\mathrm{ELSP}_4(\mathrm{INS}_1^{1,0}\mathrm{DEL}_2^{0,0})$	[13]
4.	$\mathrm{ELSP}_5(\mathrm{INS}_1^{0,1}\mathrm{DEL}_2^{0,0})$	$\mathrm{ELSP}_4(\mathrm{INS}_1^{0,1}\mathrm{DEL}_2^{0,0})$	Thm 7
5.	$\mathrm{ELSP}_5(\mathrm{INS}_2^{0,0}\mathrm{DEL}_1^{1,0})$	$\mathrm{ELSP}_4(\mathrm{INS}_2^{0,0}\mathrm{DEL}_1^{1,0})$	Thm 4
		$\mathrm{ELSP}_3(\mathrm{INS}_2^{0,1}\mathrm{DEL}_1^{1,0})$	Thm 5
		$\mathrm{ELSP}_3(\mathrm{INS}_2^{1,0}\mathrm{DEL}_1^{1,0})$	Thm 6
6.	$\mathrm{ELSP}_5(\mathrm{INS}_2^{0,0}\mathrm{DEL}_1^{0,1})$	$\mathrm{ELSP}_4(\mathrm{INS}_2^{0,0}\mathrm{DEL}_1^{0,1})$	Thm 4
		$\mathrm{ELSP}_3(\mathrm{INS}_2^{1,0}\mathrm{DEL}_1^{0,1})$	Thm 5
		$\mathrm{ELSP}_3(\mathrm{INS}_2^{0,1}\mathrm{DEL}_1^{0,1})$	Thm 6

Corollary 2 *The following consequences are trivial.*

1. [11] $\mathrm{RE} = \mathrm{GCID}(3; 1, 1, 0; 1, 1, 0) = \mathrm{GCID}(3; 1, 0, 1; 1, 0, 1)$ *implies*
   (a) $\mathrm{RE} = \mathrm{GCID}(3; 2, 1, 0; 1, 1, 0) = \mathrm{GCID}(3; 2, 0, 1; 1, 0, 1)$
   (b) $\mathrm{RE} = \mathrm{GCID}(3; 1, 1, 0; 2, 1, 0) = \mathrm{GCID}(3; 1, 0, 1; 2, 0, 1)$
2. [11] $\mathrm{RE} = \mathrm{GCID}(3; 1, 1, 0; 1, 0, 1) = \mathrm{GCID}(3; 1, 0, 1; 1, 1, 0)$ *implies*
   (a) $\mathrm{RE} = \mathrm{GCID}(3; 2, 1, 0; 1, 0, 1) = \mathrm{GCID}(3; 2, 0, 1; 1, 1, 0)$
   (b) $\mathrm{RE} = \mathrm{GCID}(3; 1, 1, 0; 2, 0, 1) = \mathrm{GCID}(3; 1, 0, 1; 2, 1, 0)$
3. [11] $\mathrm{RE} = \mathrm{GCID}_P(5; 1, 1, 1; 1, 0, 0) = \mathrm{GCID}_P(5; 1, 0, 0; 1, 1, 1)$ *implies*
   (a) $\mathrm{RE} = \mathrm{GCID}_P(5; 2, 1, 1; 1, 0, 0) = \mathrm{GCID}_P(5; 1, 0, 0; 2, 1, 1)$
4. [11] $\mathrm{RE} = \mathrm{GCID}_P(3; 1, 1, 1; 1, 1, 0) = \mathrm{GCID}_P(3; 1, 1, 1; 1, 0, 1)$ *implies*
   (a) $\mathrm{RE} = \mathrm{GCID}_P(3; 2, 1, 1; 1, 1, 0) = \mathrm{GCID}_P(3; 2, 1, 1; 1, 0, 1)$
5. *(Thm. 1)*$\mathrm{RE} = \mathrm{GCID}_P(3; 1, 1, 0; 1, 1, 1) = \mathrm{GCID}_P(3; 1, 0, 1; 1, 1, 1)$ *implies*
   (a) $\mathrm{RE} = \mathrm{GCID}_P(3; 1, 1, 0; 2, 1, 1) = \mathrm{GCID}_P(3; 1, 0, 1; 2, 1, 1)$
6. *(Thm. 2)* $\mathrm{RE} = \mathrm{GCID}_P(4; 1, 1, 0; 1, 1, 0) = \mathrm{GCID}_P(4; 1, 0, 1; 1, 0, 1)$ *implies*
   (a) $\mathrm{RE} = \mathrm{GCID}_P(4; 1, 1, 0; 2, 1, 0) = \mathrm{GCID}_P(4; 1, 0, 1; 2, 0, 1)$
7. *(Thm. 3)* $\mathrm{RE} = \mathrm{GCID}_P(4; 1, 1, 0; 1, 0, 1) = \mathrm{GCID}_P(4; 1, 0, 1; 1, 1, 0)$ *implies*
   (a) $\mathrm{RE} = \mathrm{GCID}_P(4; 1, 1, 0; 2, 0, 1) = \mathrm{GCID}_P(4; 1, 0, 1; 2, 1, 0)$

5 Summary and open problems

In this paper, we focused on examining the computational power of graph-controlled ins–del systems with paths as control graphs, which naturally correspond to a variant of P systems.

Table 9 Analysis of the generative power of GCID system of sizes $(k; 1, i', i''; 1, j', j'')$

No.	Size of the system $(k; 1, i', i''; 1, j', j'')$	Value of k	Control graph type	Reference
1.	$(k; 1, 0, 0; 1, 1, 1)$ or $(k; 1, 1, 1; 1, 0, 0)$	5	Path	[11]
2.	$(k; 1, 1, 0; 1, 1, 0)$ or $(k; 1, 0, 1; 1, 0, 1)$	3	Non-tree	[11]
		4	Non-tree	[13], Thm. 7
		4	Path	Thm. 2
3.	$(k; 1, 1, 0; 1, 0, 1)$ or $(k; 1, 0, 1; 1, 1, 0)$	3	Non-tree	[11]
		4	Non-tree	[13], Thm. 7
		4	Path	Thm. 3
4.	$(k; 1, 1, 0; 1, 1, 1)$ or $(k; 1, 0, 1; 1, 1, 1)$	3	Path	Thm. 1
5.	$(k; 1, 1, 1; 1, 1, 0)$ or $(k; 1, 1, 1; 1, 0, 1)$	3	Path	[11]
6.	$(k; 1, 1, 1; 1, 1, 1)$	1	Null	[33]

Table 10 Analysis of the generative power of GCID system for $n = 1$ and $m = 2$

No.	Size of the system $(k; 1, i', i''; 2, j', j'')$	Value of k	Control graph type	Reference
1.	$(k; 1, 0, 0; 2, 1, 1)$	5	Path	Cor. 2, 3a
2.	$(k; 1, 1, 0; 2, 0, 0)$ or $(k; 1, 0, 1; 2, 0, 0)$	3	Non-tree	[12]
		4	Path	[13], Thm. 7
3.	$(k; 1, 1, 0; 2, 1, 0)$ or $(k; 1, 0, 1; 2, 0, 1)$	3	Non-tree	Cor. 2, 1b
		4	Path	Cor. 2, 6a
4.	$(k; 1, 1, 0; 2, 0, 1)$ or $(k; 1, 0, 1; 2, 1, 0)$	3	Non-tree	Cor. 2, 2b
		4	Path	Cor. 2, 7a
5.	$(k; 1, 1, 0; 2, 1, 1)$ or $(k; 1, 0, 1; 2, 1, 1)$	3	Path	Cor. 2, 5a
6.	$(k; 1, 1, 1; 2, 0, 0)$ or $(k; 1, 1, 1; 2, 1, 0)$ or $(k; 1, 1, 1; 2, 0, 1)$	1	Null	[31]
7.	$(k; 1, 1, 1; 2, 1, 1)$	1	Null	[33]

We lowered the resource requirements to describe all recursively enumerable languages. On fixing the parameters n, i', i'', m, j', j'' in a GCID size $(k; n, i', i''; m, j', j'')$, we present in Tables 9, 10 and 11, the number of components k and the type of control graph, with which GCID$(k; n, i', i''; m, j', j'')$ describe RE, as our main focus of study was to reduce the number of components required for computational completeness results while keeping the ID size fixed. However, it is open whether these resource bounds are optimal.

Here we considered the underlying graph of GCID systems to be path-structured only. One may also consider also tree structure, which may give additional power, especially to ins–del P systems. The resources used in the results of ins–del P systems need not be optimal since in ins–del P systems, each membrane can have initial strings and they all evolve in parallel which may reduce the size.

The reader may have noticed that we discussed in detail the case of insertion strings of length two, but a similar discussion for the case of deletion strings of length two is missing. We can trivially inherit some computational completeness results from the case when both insertion and deletion strings have length one only. However, it is open whether

Table 11 Analysis of the generative power of GCID system for $n = 2$ and $m = 1$

No.	Size of the system $(k; 2, i', i''; 1, j', j'')$	Value of k	Control graph type	Reference
1.	$(k; 2, 1, 1; 1, 0, 0)$	5	Path	Cor. 2, 3a
2.	$(k; 2, 0, 0; 1, 1, 0)$ or $(k; 2, 0, 0; 1, 0, 1)$	3	Non-tree	[12]
		4	Non-tree	[13], Thm. 7
		4	Path	Thm. 4
3.	$(k; 2, 1, 0; 1, 1, 0)$ or $(k; 2, 0, 1; 1, 0, 1)$	3	Non-tree	Cor. 2, 1a
		3	Path	Thm. 6
4.	$(k; 2, 1, 0; 1, 0, 1)$ or $(k; 2, 0, 1; 1, 1, 0)$	3	Non-tree	Cor. 2, 2a
		3	Path	Thm. 5
5.	$(k; 2, 1, 1; 1, 1, 0)$ or $(k; 2, 1, 1; 1, 0, 1)$	3	Path	Cor. 2, 4a
6.	$(k; 2, 0, 0; 1, 1, 1)$ or $(k; 2, 1, 0; 1, 1, 1)$ or $(k; 2, 0, 1; 1, 1, 1)$	1	Null	[21]
7.	$(k; 2, 1, 1; 1, 1, 1)$	1	Null	[33]

$$\mathrm{RE} = \mathrm{GCID}_P(3; 1, 1, 0; 2, 1, 0) = \mathrm{GCID}_P(3; 1, 1, 0; 2, 0, 1),$$

to state one concrete question in that direction. It is also interesting to observe that we never used the possibility to introduce finite sets of axioms in our simulations ; we only used singleton sets as sets of axioms. Also, only in two simulations (in this paper) we made use of the possibility to have strings of length greater than one as axioms. Is it always the case that these seemingly small modifications in the definition of GCID systems make no difference for the described language class?

In view of the connections with P systems, it would be also interesting to study Parikh images of (restricted) graph-controlled ins–del systems, as started out for matrix-controlled ins–del systems in [7]. This also relates to the macroset GCID systems considered in [6].

Furthermore, as it seems to be quite difficult to achieve computational completeness results for the remaining open cases, attention has now turned to the question of what kinds of known grammatical mechanism could be simulated with such weaker forms of (regulated) ins–del systems. We refer the readers to [9,12] for first results in this direction in connection with graph control.

Acknowledgements Some part of the work done by the second author was during his visit to University of Trier, Germany, in December 2016. The possibility to use some overhead money from the DFG Grant FE 560/6-1 to finance this visit is gratefully acknowledged.

References

1. Alhazov, A., Freund, R., Ivanov, S.: Length P systems. Fundam. Inform. **134**(1–2), 17–38 (2014)
2. Alhazov, A., Krassovitskiy, A., Rogozhin, Y., Verlan, S.: P systems with minimal insertion and deletion. Theor. Comput. Sci. **412**(1–2), 136–144 (2011)
3. Benne, R. (ed.): RNA Editing: The Alteration of Protein Coding Sequences of RNA. Series in Molecular Biology. Ellis Horwood, Chichester (1993)
4. Biegler, F., Burrell, M.J., Daley, M.: Regulated RNA rewriting: modelling RNA editing with guided insertion. Theor. Comput. Sci. **387**(2), 103–112 (2007)
5. Dassow, J., Păun, G.: Regulated Rewriting in Formal Language Theory, EATCS Monographs in Theoretical Computer Science, vol. 18. Springer, Berlin (1989)

6. Fernau, H.: An essay on general grammars. J. Autom. Lang. Comb. **21**, 69–92 (2016)
7. Fernau, H., Kuppusamy, L.: Parikh images of matrix ins-del systems. In: Gopal, T.V., Jäger, G., Steila, S. (eds.) Theory and Applications of Models of Computation, TAMC, LNCS, vol. 10185, pp. 201–215. Springer, Berlin (2017)
8. Fernau, H., Kuppusamy, L., Raman, I.: Computational completeness of path-structured graph-controlled insertion-deletion systems. In: Carayol, A., Nicaud, C. (eds.) Implementation and Application of Automata–22nd International Conference, CIAA, LNCS, vol. 10329, pp. 89–100. Springer, Berlin (2017)
9. Fernau, H., Kuppusamy, L., Raman, I.: Graph-controlled insertion-deletion systems generating language classes beyond linearity. In: Pighizzini, G., Câmpeanu, C. (eds.) Descriptional Complexity of Formal Systems–19th IFIP WG 102 International Conference, DCFS, LNCS, vol. 10316, pp. 128–139. Springer, Berlin (2017)
10. Fernau, H., Kuppusamy, L., Raman, I.: Investigations on the power of matrix insertion-deletion systems with small sizes. Natl. Comput. (accepted) (2017)
11. Fernau, H., Kuppusamy, L., Raman, I.: On the computational completeness of graph-controlled insertion-deletion systems with binary sizes. Theor. Comput. Sci. **682**, 100–121 (2017). (Special Issue on Languages and Combinatorics in Theory and Nature)
12. Fernau, H., Kuppusamy, L., Raman, I.: On the generative power of graph-controlled insertion-deletion systems with small sizes. J. Autom. Lang. Comb. **22**, 61–92 (2017)
13. Freund, R., Kogler, M., Rogozhin, Y., Verlan, S.: Graph-controlled insertion-deletion systems. In: McQuillan, I., Pighizzini, G. (eds.) Proceedings Twelfth Annual Workshop on Descriptional Complexity of Formal Systems, DCFS, EPTCS, vol. 31, pp. 88–98 (2010)
14. Geffert, V.: Normal forms for phrase-structure grammars. RAIRO Theor. Inform. Appl. **25**, 473–498 (1991)
15. Haussler, D.: Insertion languages. Inf. Sci. **31**(1), 77–89 (1983)
16. Ivanov, S., Verlan, S.: About one-sided one-symbol insertion-deletion P systems. In: Alhazov, A., Cojocaru, S., Gheorghe, M., Rogozhin, Y., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing—14th International Conference, CMC 2013, LNCS, vol. 8340, pp. 225–237. Springer, Berlin (2014)
17. Ivanov, S., Verlan, S.: Random context and semi-conditional insertion-deletion systems. Fundam. Inform. **138**, 127–144 (2015)
18. Kari, L.: On insertion and deletion in formal languages. Ph.D. thesis, University of Turku, Finland (1991)
19. Kari, L., Păun, Gh., Thierrin, G., Yu, S.: At the crossroads of DNA computing and formal languages: characterizing recursively enumerable languages using insertion-deletion systems. In: Rubin, H., Wood, D.H. (eds.) DNA Based Computers III, DIMACS Series in Discrete Mathematics and Theretical Computer Science, vol. 48, pp. 329–338 (1999)
20. Kari, L., Thierrin, G.: Contextual insertions/deletions and computability. Inf. Comput. **131**(1), 47–61 (1996)
21. Krassovitskiy, A., Rogozhin, Y., Verlan, S.: Further results on insertion-deletion systems with one-sided contexts. In: Martín-Vide, C., Otto, F., Fernau, H. (eds.) Language and Automata Theory and Applications, Second International Conference, LATA, LNCS, vol. 5196, pp. 333–344. Springer, Berlin (2008)
22. Krassovitskiy, A., Rogozhin, Y., Verlan, S.: Computational power of insertion-deletion (P) systems with rules of size two. Natl. Comput. **10**, 835–852 (2011)
23. Krishna, S.N., Rama, R.: Insertion-deletion P systems. In: Jonoska, N., Seeman, N.C. (eds.) DNA Computing, 7th International Workshop on DNA-Based Computers, 2001, Revised Papers, LNCS, vol. 2340, pp. 360–370. Springer, Berlin (2002)
24. Kuppusamy, L., Mahendran, A., Krishna, S.N.: Matrix insertion-deletion systems for bio-molecular structures. In: Natarajan, R., Ojo, A.K. (eds.) Distributed Computing and Internet Technology–7th International Conference, ICDCIT, LNCS, vol. 6536, pp. 301–312. Springer, Berlin (2011)
25. Kuppusamy, L., Rama, R.: On the power of tissue P systems with insertion and deletion rules. In: Pre-Proceedings of Workshop on Membrane Computing, Report RGML, vol. 28, pp. 304–318. Univ. Tarragona, Spain (2003)
26. Marcus, S.: Contextual grammars. Revue Roumaine de Mathématiques Pures et Appliquées **14**, 1525–1534 (1969)
27. Margenstern, M., Păun, Gh, Rogozhin, Y., Verlan, S.: Context-free insertion-deletion systems. Theor. Comput. Sci. **330**(2), 339–348 (2005)
28. Matveevici, A., Rogozhin, Y., Verlan, S.: Insertion-deletion systems with one-sided contexts. In: Durand-Lose, J.O., Margenstern, M. (eds.) Machines, Computations, and Universality, 5th International Conference, MCU, LNCS, vol. 4664, pp. 205–217. Springer, Berlin (2007)
29. Păun, Gh: Marcus Contextual Grammars, Studies in Linguistics and Philosophy, vol. 67. Kluwer Academic Publishers, Dordrecht (1997)
30. Păun, G.: Membrane Computing: An Introduction. Springer, Berlin (2002)

31. Păun, G., Rozenberg, G., Salomaa, A.: DNA Computing: New Computing Paradigms. Springer, Berlin (1998)
32. Petre, I., Verlan, S.: Matrix insertion-deletion systems. Theor. Comput. Sci. **456**, 80–88 (2012)
33. Takahara, A., Yokomori, T.: On the computational power of insertion-deletion systems. Natl. Comput. **2**(4), 321–336 (2003)
34. Verlan, S.: Recent developments on insertion-deletion systems. Comput. Sci. J. Moldova **18**(2), 210–245 (2010)