Natural Computing (2019) 18:563–577
https://doi.org/10.1007/s11047-019-09742-w

Computational completeness of simple semi-conditional insertion–deletion systems of degree (2,1)

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

Published online: 30 May 2019

Abstract
Insertion–deletion (or ins–del for short) systems are simple models of bio-inspired computing. They are well studied in formal language theory, especially regarding their computational completeness. This concerns the question if all recursively enumerable languages can be generated. This ultimately addresses the question if one can build general-purpose computers rooted in this formalism. The descriptional complexity of an ins–del system is typically measured by its *size*, a 6-dimensional tuple of non-negative integers $(e, e', e''; d, d', d'')$ where e is the maximum length of the insertion string, e' (and e'') is the maximum length of the left (and right) context used for insertion; the last three parameters d, d', d'' are similarly understood for deletion rules. Computational completeness for ins–del systems can even be achieved with rule size (1, 1, 1; 1, 1, 1) but with no rule size strictly smaller than this. This fact has motivated to study ins–del systems in combination with regulation mechanisms. In this context, the six-tuple explained above is called the *ID size* of a system. Several regulations like graph-control, matrix and semi-conditional have been imposed on ins–del systems. Typically, the computational completeness results are obtained as trade-offs, reducing the ID size, say, to (1, 1, 0; 1, 1, 0) at the expense of increasing other measures of descriptional complexity. In this paper, we study *simple semi-conditional ins–del systems*, where an ins–del rule can be applied only in the presence or absence of substrings of the derivation string. This brings along two further natural parameters to measure descriptional complexity, namely, the maximum permitting string length p and the maximum forbidden string length f, usually summarized as the degree $d = (p,f)$. We show that simple semi-conditional ins–del systems of degree (2, 1) and with ID sizes $(1 + e, e', e''; 1 + d, d', d'')$ are computationally complete for any $e, e', e'', d, d', d'' \in \{0, 1\}$, with $e + e' + e'' = 1$ and $d + d' + d'' = 1$. The obtained results complement and improve on the existing results known from the literature. To prove our results, we also show a new normal form for type-0 grammars that appears to be interesting in its own right.

Keywords Insertion–deletion · Simple semi-conditional · Descriptional complexity · Computational completeness · Special Geffert normal form

A preliminary version of this paper appeared in Proceedings of UCNC 2018, LNCS 10867, pp. 86–100, 2018.

✉ Lakshmanan Kuppusamy
klakshma@vit.ac.in

Henning Fernau
fernau@uni-trier.de

Indhumathi Raman
ind.amcs@psgtech.ac.in

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

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

[3] Department of Applied Mathematics and Computational Sciences, PSG College of Technology, Coimbatore 641 004, India

1 Introduction

Insertion–deletion systems are a computational model based on the operations of insertion and deletion of substrings in a string. Initially motivated on linguistic grounds, they more recently became quite popular as a theoretical model for DNA-based computations, as the basic operations fit well into this area. For further discussions on the history of this model, as well as giving insights into the rich literature of this area, we refer to Kari and Thierrin (1996), Verlan (2007, 2010).

In a nutshell, the rules of an insertion–deletion system (or ins–del system) can be of two types: insertion or deletion, i.e., either, a string is specified that may be inserted in a prescribed context within the current string, or it may be deleted relative to the context conditions. The potential biological or bio-chemical meaning of such a rule should be clear: parts of a strand-like molecule are inserted or deleted in a certain context. The main research question is under which restrictions can computational completeness results still be obtained. For instance, it is known (Takahara and Yokomori 2003) that for each recursively enumerable language (or RE language for short), there exists an ins–del system where only single symbols are inserted or deleted, and the allowed context conditions (to the left or to the right) are again (at most) single symbols. However, if we disallow checking contexts both to the left and to the right, then not all RE languages can be described; cf. Verlan (2007). In such situations, several regulation mechanisms have been studied and shown to achieve computational completeness results. From the viewpoint of biocomputing, let us only mention ins–del P systems (Krishna and Rama 2002; Krassovitskiy et al. 2011), sometimes in disguise (Fernau et al. 2019), tissue P systems with ins–del rules (Kuppusamy and Rama 2003), random context and semi-conditional ins–del systems (Ivanov and Verlan 2015).

Meduna and Svec have reported on the use of several variants of context conditions in regulated rewriting in the textbook (Meduna and Svec 2005). In this paper, (simple) semi-conditional rules are of particular importance. In the semi-conditional case, the conditions are sets of words and are associated with each rule. A rule can be applied if all words from its *permitting* condition are present and no word from the *forbidden* condition is present in the string. A semi-conditional grammar is said to be *simple* if each rule has only either a permitting condition or a forbidden condition. Notice that this type of simplicity can be again interesting from a bio-chemical perspective: insertion or deletion operations are usually triggered by some sort of enzymes or other chemical regulation mechanisms; such catalysts can be enabled or disabled by conditions as formalized in conditional grammars, but it should be easier to have only single conditions here when it comes to biological implementations. Let the maximum length of a string in the permitting and forbidden set be denoted by i and j, respectively; then the ordered pair (i, j) is called the *degree* of the semi-conditional grammar. From a biological point of view, these conditions can be interpreted as *global* context conditions, as opposed to the *local* context conditions traditionally represented within the ins–del rules themselves.

The study of semi-conditional ins–del systems was initiated in Ivanov and Verlan (2015). They proved that with degree (2, 2), inserting and deleting single symbols without any local context is sufficient to describe any RE language. Conversely, extending previous computational incompleteness results on non-regulated ins–del systems, it was shown in the same paper that ins–del systems that may insert or delete single symbols in one-sided single-symbol context are not able to describe the regular language $\{ab\}^+$, assuming that these systems can also globally check for single symbols only, i.e., if they are of degree (1, 1).

No previous computational completeness results have been known for other degrees. This motivates the present study. More precisely, the degree (2, 1) that is in the focus of our study somehow interpolates between the degrees (2, 2) and (1, 1) that were previously studied in the literature (Ivanov and Verlan 2015). We also think that it might be possible to also globally check for the presence or absence of short molecular parts (strings) within biocomputational devices. Furthermore, we managed to cope with the already mentioned *simple* restriction on semi-conditional rules. Clearly, this additional restriction is a technical challenge. More specifically, we prove that simple semi-conditional ins–del systems of degree (2, 1) are computationally complete if (i) strings of length two may either be inserted or deleted without any local conditions, or (ii) only single symbols (with one-sided single-symbol local context) may be inserted, but strings of length two may be deleted without any local conditions, or (iii) only single symbols (with one-sided single-symbol local context) may be inserted and single symbols (with one-sided single-symbol local context) may be deleted.

The results of this paper and a sketch on how they complement the existing results of Ivanov and Verlan (2015) are given in Table 1. The notation of the language families (explained in details below) basically distinguishes between semi-conditional (SC) and simple semi-conditional (SSC). In parenthesis, the 6-dimensional integer tuple upper-bounding the admitted ID size is given. Finally, the index refers to the admitted degree. The table also shows how the results of this paper improve on results obtained in the conference version of this paper

Table 1 Comparing the results of Ivanov and Verlan (2015), of Fernau et al. (2018a) and of this paper

	Result of Ivanov and Verlan (2015)	Our complementing result(s)	References
1.	$SC_{2,2}ID(1,0,0;1,0,0) = RE$	$SSC_{2,1}ID(2,0,0;2,0,0) = RE$	Theorem 3
2.	$SC_{1,1}ID(1,1,0;2,0,0) \subsetneq RE$	$SSC_{2,1}ID(1,1,0;2,0,0) = RE$	Theorem 5
3.	$SC_{1,1}ID(1,1,0;1,1,1) \subsetneq RE$	$SSC_{2,1}ID(1,1,0;1,1,1) = RE$	Thm. 4 in Fernau et al. (2018a)
4.	$SC_{1,1}ID(2,0,0;1,1,0) = RE$	$SSC_{2,1}ID(2,0,0;2,0,0) = RE$	Theorem 3
5.		$SSC_{3,1}ID(1,1,0;1,1,0) = RE$	Thm. 5 in Fernau et al. (2018a)
6.		$SSC_{3,1}ID(1,0,1;1,1,0) = RE$	Thm. 6 in Fernau et al. (2018a)
4.	$SC_{1,1}ID(2,0,0;1,1,0) = RE$	$SSC_{2,1}ID(2,0,0;1,1,0) = RE$	Theorem 4
5.	$SC_{1,1}ID(1,1,0;1,1,1) \subsetneq RE$	$SSC_{2,1}ID(1,1,0;1,1,0) = RE$	Theorem 6
6.	$SC_{1,1}ID(1,1,0;1,1,1) \subsetneq RE$	$SSC_{2,1}ID(1,0,1;1,1,0) = RE$	Theorem 7

(Fernau et al. 2018a): In the case of two ID sizes, we are now able to admit degree (2, 1) where previously, we could only prove a computational completeness result for degree (3, 1). Yet, the reader is encouraged to also look into the previous results, as the constructions there are clearly conceptually simpler than the ones contained in this paper, but this simplicity comes at a price: the results themselves are weaker, as can be also seen in Table 1. The comparison of the results is also supported by using the same numbers for results related to the same ID sizes for SSC ins–del systems. Finally from the table, some optimality results can be read off. For instance, the last two rows indicate computational completeness results that cannot be improved when moving from degree (2, 1) to degree (1, 1), even in the case when permitting non-simple rules or when allowing both-sided contexts for deletion rules with deletion strings of any length.

Another contribution of this paper (not contained in the conference version) is the presentation of a new normal form for type-0 grammars. We call this the *space separating special Geffert normal form*, because it is obtained in a way from a variant of the well-known Geffert normal form that is similar to the way how the so-called *special Geffert normal form* was produced. Also, it is called *space separating*, because the rules are designed in a fashion that guarantees that for a certain variety of nonterminal symbols, from $N'' = \{A, B, C, D, E, F\}$, no sentential form is ever generated that contains the substring ZZ for any $Z \in N''$. In other words, the occurrences of two identical symbols from N'' are always separated from each other.

2 Preliminaries

Let $\mathbb{N}$ denote the set of non-negative integers, and $[1..k] = \{i \in \mathbb{N} : 1 \leq i \leq k\}$. If Σ is an *alphabet* (finite set), then Σ^* denotes the free monoid generated by Σ. The elements of Σ^* are called *strings* or *words*; λ denotes the empty string. The morphism from the monoid Σ^* to $\mathbb{N}$ (with addition), defined by $a \mapsto 1$ for $a \in \Sigma$, is called the *length* of a word; usually, we write $|w|$. The set of all words over Σ of length at most i is denoted by $\Sigma^{\leq i}$. A word v is a subword of $x \in \Sigma^*$ if there are words u, w such that $x = uvw$. Let $sub(x) \subseteq \Sigma^*$ denote the set of all subwords of $x \in \Sigma^*$. We also make use of the *insertion operation* $\triangleright$ to describe the effect of insertions at a random position in the string. Hence, $u \triangleright v = \{v_1 u v_2 \mid v = v_1 v_2\}$. This operation is readily lifted to languages: $U \triangleright V = \bigcup_{u \in U, v \in V} u \triangleright v$. We assume that $\triangleright$ is right-associative, i.e, $U \triangleright V \triangleright W := U \triangleright (V \triangleright W)$. We write w^R to denote the reversal of $w \in \Sigma^*$. We can easily lift this notation to sets of words (i.e., languages) and also to families of languages, as well.

2.1 Normal forms for type-0 grammars

For the computational completeness results, we are using the fact that type-0 grammars in certain normal forms characterize the class RE of recursively enumerable languages.

Definition 1 (see Freund et al. 2010; Ivanov and Verlan 2015) A type-0 grammar[1] $G = (N, T, P, S)$ is said to be in *Special Geffert Normal Form*, or SGNF for short, if

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

[1] As usual, N is the nonterminal alphabet, T is the terminal alphabet, P is the set of rules and S is the start symbol of grammar G. Also, we use $\Rightarrow$ to denote a single derivation step of G and $\Rightarrow^*$ to denote an arbitrary number of derivation steps.

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

The way the normal form is constructed in Freund et al. (2010) and is based on Geffert (1991).[2] So, the construction starts out with the classical *Geffert normal form* that has context-free rules of the form $S \to uSa$ and $S \to uSv$, as well as $S \to uv$, for $u \in \{A, C\}^*$, $v \in \{B, D\}^*$, $a \in T$, plus two non-context-free rules $AB \to \lambda$ and $CD \to \lambda$. We can also replace $S \to uv$ by $S \to uS'v$ and $S' \to \lambda$. The SGNF is obtained by first breaking the rules of the type $S \to uSx$ into $S \to uS_x$ and $S_x \to Sx$ and then "spelling out" words u or x of length more than one. For instance, $S \to abSx$, would become $S \to aS_x^b$ and $S_x^b \to bS_x$ and $S_x \to Sx$. Similarly, $S \to uS'v$ can be spelled out.

Also, the derivation of a string is done in two phases. In phase I, the context-free rules are applied repeatedly; this phase is completed by applying the rule $S' \to \lambda$ in the derivation. In phase II, only the non-context-free erasing rules are applied repeatedly until a terminal string is reached. From its invention, this normal form turned out to be a very useful tool for proving computational completeness results for (regulated) ins–del systems. However, notice that Geffert designed his normal form(s) in order to produce grammars with few nonterminals and few non-context-free rules. This is not so much of interest in our present study. Therefore, we will modify the sketched constructions in a way that allows us to have more structure in the sentential forms. This will help us in our arguments, leading to fewer cases to examine.

The first point is that we can modify Geffert's normal form itself by introducing the nonterminal S' already into this normal form and having rules of the forms $S \to uSa$ and $S \to uS'a$ or $S \to \lambda$ if λ is contained in the language[3]. Moreover, we have context-free rules of the form $S' \to uS'v$ and $S' \to \lambda$ in our grammar. The advantage of our construction over the classical Geffert normal form is that all sentential forms that can be derived by using context-free rules belong to

$$\{A, C\}^*\{S\}T^* \cup \{A, C\}^*\{S'\}\{B, D\}^*T^* \cup \{A, C\}^*\{B, D\}^*T^*,$$

where it is clear that sentential forms from $\{A, C\}^*\{S\}T^*$ are produced in the first stage of the first phase, using rules $S \to uSa$ only, then a transitional rule like $S \to uS'a$ has to be executed, and from that time on, it is possible to produce strings from $\{A, C\}^*\{S'\}\{B, D\}^*T^*$ in the second stage of the first phase, using rules like $S' \to uS'v$. Finally, the rule $S' \to \lambda$ is used to produce a string from $\{A, C\}^*\{B, D\}^*T^*$, hence exiting the first phase. In the second phase, as usual, only erasing non-context-free rules ($AB \to \lambda$ and $CD \to \lambda$) are applied. Upon doing so, only sentential forms from $\{A, C\}^*\{B, D\}^*T^*$ can be produced, hence keeping this property as an invariant. When using $S \to \lambda$ to exit the first stage of the first phase, then we also arrive at a string from $\{A, C\}^*\{B, D\}^*T^*$, more precisely, from $\{A, C\}^*T^*$, but here further derivation steps are not possible, as can be easily verified.

Our further discussions will show that we can easily modify the construction leading to SGNF in order to avoid substrings like AA or BB ever occurring in sentential forms derivable in normal form grammars. Namely, we first modify the Geffert normal form construction itself by replacing the context-free rules $S \to uSa$ and $S \to uS'a$, as well as $S' \to uS'v$, for $u \in \{A, C\}^*$, $v \in \{B, D\}^*$, $a \in T$, by $S \to h(u)Sa$, $S \to h(u)S'a$ or $S' \to h(u)S'h(v)$, respectively, where the morphism h is defined by $A \mapsto EA$, $B \mapsto BF$, $C \mapsto EC$ and $D \mapsto DF$. Now, we have three non-context-free rules, namely $AB \to \lambda$, $CD \to \lambda$ and $EF \to \lambda$. It should be clear that the symbol E prevents AA and CC from ever being produced as a substring of any sentential form. Likewise, the symbol F prevents BB and DD from ever being produced as a substring of any sentential form. Clearly, this property is preserved when deriving the "special" form from this variant of Geffert normal form, which leads us to the following definition and theorem.

Definition 2 A type-0 grammar $G = (N, T, P, S)$ is said to be in *Space Separating Special Geffert Normal Form*, or ssSGNF for short, if

- N decomposes as $N = N' \cup N''$, where $N'' = \{A, B, C, D, E, F\}$ and N' further decomposes as $N' = N_S \cup N_{S'}$ such that N_S contains at least the nonterminal S and $N_{S'}$ contains at least the nonterminal S',
- the only non-context-free rules in P are the erasing rules $AB \to \lambda$, $CD \to \lambda$ and $EF \to \lambda$,
- the non-erasing context-free rules are of the following forms:
  - $X \to Yb$ or $X \to b'Y$, where $X \in N_S$, $Y \in N'$, $X \neq Y$, $b \in T$, $b' \in \{A, C, E\}$, or
  - $X \to Yb$ or $X \to b'Y$, where $X, Y \in N_{S'}$, $X \neq Y$, $b \in \{B, D, F\}$, $b' \in \{A, C, E\}$;
- G contains the erasing context-free rule $S' \to \lambda$ and also possibly $S \to \lambda$.

It is clear from the h morphism and the discussion above, that without loss of generality, we can restrict our attention to ssSGNF's whose derivations split into two

[2] The construction sketched in Freund et al. (2010) is a bit different where $b, b' \in \{N'' \cup T\}$.

[3] With this requirement, we introduce a step that is non-constructive, mostly for reasons of laziness, but as we do not bother about decidability issues, this does not matter here.

phases; where in the first phase, only context-free rules are applied and only strings from

$$\{EA, EC\}^* N_S T^* \cup \{EA, EC\}^* N_{S'} \{BF, DF\}^* T^* \cup \{EA, EC\}^* \{BF, DF\}^* T^*$$

are produced; in the second phase, only non-context-free erasing rules are applied and only strings from $\{EA, EC\}^* \{\lambda, EF\} \{BF, DF\}^* T^*$ can be derived.

Furthermore, this ssSGNF can be obtained from SGNF by simply replacing a previously existing rule $X \to b'Y$ (where $b' \in \{A, C\}$) by the rules $X \to EZ$ and $Z \to b'Y$ (and similarly for rules $X \to b'Y$ where $b' \in \{B, D\}$).

Our considerations allow us to state the following result.

Theorem 1 *For every recursively enumerable language, there exists a type-0 grammar in ssSGNF that describes it. Moreover, we know for ssSGNF grammars: for any* $w \in (N \cup T)^*$ *such that* $S \Rightarrow^* w$, *then* $\{AA, BB, CC, DD, EE, FF\} \cap sub(w) = \emptyset$. □

2.2 Insertion–deletion systems

We now give the basic definition of insertion–deletion systems, following (Kari and Thierrin 1996; Păun et al. 1998).

Definition 3 Kari and Thierrin (1996); Păun et al. (1998) An *insertion–deletion system*, or *ins–del system* for short, is a construct $\gamma = (V, T, A, R)$, where V is an alphabet, $T \subseteq V$ is the terminal alphabet, A is a finite language over V, R is a finite set of triplets of the form $(u, \eta, v)_{ins}$ or $(u, \delta, v)_{del}$, where $(u, v) \in V^* \times V^*$, $\eta, \delta \in V^+$.

The pair (u, v) is called the *context*, where we differentiate between the *left context* u and the *right context* v, η is called the *insertion string*, δ is called the *deletion string* and $x \in A$ is called an *axiom*. If either the left or the right context is empty for all the insertion (deletion) contexts, then we call the insertion (deletion) *one-sided*. If both contexts are empty for every insertion (deletion) rule, then the insertion (deletion) is called *context-free*. The *descriptional complexity* of an ins–del system is measured by its *size* $s = (n, i', i''; m, j', j'')$, where the parameters represent resource bounds as given in Table 2.

2.3 Semi-conditional insertion–deletion systems

Definition 4 (Ivanov and Verlan 2015) A semi-conditional insertion–deletion system of *degree* (i, j), $i, j \geq 0$ is a construct $\Pi = (V, T, A, R)$, where V is a finite alphabet, $T \subseteq V$ is the terminal alphabet, $A \subseteq V^*$ is a finite set of axioms, R is a finite set of rules of the form $[(u, s, v)_t, \mathcal{P}, \mathcal{F}]$ where $u, s, v \in V^*$, $t \in \{ins, del\}$, $\mathcal{P}, \mathcal{F}$ are finite subsets of V^*. The set $\mathcal{P}$ is called the *permitting* set and $\mathcal{F}$ is called the *forbidden* set. For clarity, we often use unique labels for rules, even identifying a rule with its label, i.e., if $r \in R$ is a rule (label), then $r : [(u_r, s_r, v_r)_{t_r}, \mathcal{P}_r, \mathcal{F}_r]$. The ordered pair (i, j) is called the *degree* of the semi-conditional ins–del system Π where i is the smallest integer such that $\bigcup_{r \in R} \mathcal{P}_r \subseteq V^{\leq i}$ and j is the smallest integer such that $\bigcup_{r \in R} \mathcal{F}_r \subseteq V^{\leq j}$.

Table 2 Parameters in the size of ins–del system

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

We write $x \Rightarrow_r y$ if $\mathcal{P}_r \subseteq sub(x)$ and $\mathcal{F}_r \cap sub(x) = \emptyset$ and either

1. $t_r = ins$ and $x = x_1 u_r v_r x_2$, $y = x_1 u_r s_r v_r x_2$, for some $x_1, x_2 \in V^*$; or
2. $t_r = del$ and $x = x_1 u_r s_r v_r x_2$, $y = x_1 u_r v_r x_2$, for some $x_1, x_2 \in V^*$.

The language generated by a semi-conditional insertion–deletion system Π is

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

where $\Rightarrow^*$ is the reflexive and transitive closure of $\Rightarrow$, which is $\bigcup_{r \in R} \Rightarrow_r$.

Given $\Pi = (V, T, A, R)$, the *underlying ins–del system* $\gamma_\Pi = (V, T, A, R_\gamma)$ is given by $R_\gamma = \{(u, s, v)_{t_r} \mid r \in R\}$. The size (six-tuple) associated to $(u, s, v)_{t_r}$ is also called the *ID size* of r, and the ID size of Π is hence the size of γ_Π.

The families of languages generated by semi-conditional insertion–deletion systems of degree at most (i, j) having ID size at most $s = (n, i', i''; m, j', j'')$ is denoted as $\mathrm{SC}_{i,j}\mathrm{ID}(s)$. If, for each $r \in R$, either $\mathcal{P}_r = \emptyset$ or $\mathcal{F}_r = \emptyset$, then the semi-conditional ins–del system is said to be *simple*. The families of languages generated by such simple semi-conditional insertion–deletion (denoted in short as SSCID) systems of degree at most (i, j) and ID size at most s is denoted as $\mathrm{SSC}_{i,j}\mathrm{ID}(s)$. If both $\mathcal{P} = \mathcal{F} = \emptyset$ for all rules of R then the semi-conditional ins–del system is same as the ordinary ins–del system. Hence, $\mathrm{SC}_{0,0}\mathrm{ID}(s)$ refers to the family of languages that can be described by ins–del systems of degree at most (i, j).

Example 1 Consider the non context-free language $L_1 = \{a^n b^n c^n \mid n \geq 1\}$. We construct a simple semi-conditional ins–del system Π of degree (1, 1) and ID size (3, 1, 1; 1, 0, 0) describing L_1 as follows: $\Pi = (\{A, B, a, b, c\}, \{a, b, c\}, \{abc\}, R)$ where the set of rules of

Table 3 SSCID rules describing $\{a^n b^n c^n \mid n \geq 1\}$

$r1 : [(a, aAb, b)_{ins}, \emptyset, B]$	$r2 : [(b, Bc, c)_{ins}, A, \emptyset]$
$r3 : [(\lambda, A, \lambda)_{del}, B, \emptyset]$	$r4 : [(\lambda, B, \lambda)_{del}, \emptyset, A]$

R is given in Table 3. We will now explain the working of the rules in Table 3. From the rules, we can see that $r1$ can be applied in the absence of B and $r2$ can be applied in the presence of A, thus, $r1$ has to be applied before $r2$ is applied. Note that in $r1$, as the contexts are a and b, once aAb is introduced between a and b, the rule $r1$ cannot be (immediately) applied again until A is deleted. Similarly, rule $r2$ cannot be applied for a second time unless B is deleted. Starting from the axiom abc, the only applicable rule is $r1$ which will results in $aaAbbc$. Now, $r3$ cannot be applied, as deleting A requires the presence of B and this symbol is not introduced yet. So, the only applicable rule is $r2$ which results in $aaAbbBcc$. Now, $r4$ cannot be applied as it requires the absence of A and A is still present in the derived string. The only applicable rule is hence $r3$ which deletes the A and then the only applicable rule is $r4$ which deletes the B and results to $aabbcc$. A sample derivation is given below for better understanding how the system works.

$$abc \Rightarrow_{r1} aaAbbc \Rightarrow_{r2} aaAbbBcc \Rightarrow_{r3} aabbBcc \Rightarrow_{r4} aabbcc.$$

The process above is repeated and as the rules are applied in a deterministic manner, it is easy to see that $L(\Pi) = L_1$. □

Remark 1 The purpose of Example 1 is to explain how the system works and the size used in this example does not necessarily correspond to computational completeness results obtained in this paper. On the other hand, if a type-0 grammar (in SGNF) is given for L_1, then L_1 can be simulated by a simple semi-conditional ins–del system with the sizes that are shown in the computational completeness result. □

3 Main results

In order to make some of our results simple, we claim the following, similar to other regulation mechanisms, as for example in Fernau et al. (2019).

Theorem 2 *If $s = (n, i', i''; m, j', j'')$ is some ID size and (i, j) is some degree, then* $\mathrm{SSC}_{i,j}\mathrm{ID}(s) = [\mathrm{SSC}_{i,j}\mathrm{ID}(s')]^R$, *with $s' = (n, i'', i'; m, j'', j')$.*

Namely, for every conditional rule $p : ((u, s, v)_t, \mathcal{P}, \mathcal{F})$ of Π we take the rule $p^R : ((v^R, s^R, u^R)_t, \mathcal{P}^R, \mathcal{F}^R)$ into Π^R, and also if A is the set of axioms of Π, then A^R is the set of axioms of Π^R, so that $L(\Pi^R) = L(\Pi)^R$. Clearly, the upper-bounds on the lengths of the left and right contexts are interchanged when moving from Π to Π^R. This allows us to go from ID size s to ID size s' as stated in the theorem. As RE is closed under reversal, whenever $\mathrm{SSC}_{k,l}\mathrm{ID}(s) = \mathrm{RE}$, then $\mathrm{SSC}_{k,l}\mathrm{ID}(s') = \mathrm{RE}^R = \mathrm{RE}$ and vice versa.

With s, s' as in the previous theorem, we know that $\mathrm{SSC}_{k,l}\mathrm{ID}(s) = \mathrm{RE}$ if and only if $\mathrm{SSC}_{k,l}\mathrm{ID}(s') = \mathrm{RE}$.

In order to show that simple semi-conditional ins–del systems of certain sizes describe RE, we make use of the fact that RE languages can be generated by grammars in Special Geffert Normal Form (SGNF), where the rules are of the type $p : X \to bY$, $q : X \to Yb$, $f : AB \to \lambda$, $g : CD \to \lambda$, and $e' : S' \to \lambda$, where $p, q, f, g, e' \in [1 \ldots |P|]$ are labels associated with each type of rule of SGNF. We provide a simulation of these rules by rules of simple semi-conditional ins–del system. As we are sometimes also starting with ssSGNF, rules can also look like $h : EF \to \lambda$ and $e : S \to \lambda$.

The simulation of rules g, h is similar to the simulation of f-rules. Therefore, we will only present f-rule simulations in the following, assuming a similar setting for g- and h-rules. Also, we always simulate rule e' by $[(\lambda, S', \lambda)_{del}, \emptyset, \mathcal{M}]$, and similarly we can simulate rule e, with $\mathcal{M} \in \{\mathcal{M}'', \mathcal{M}'''\}$ as defined below. Therefore, in the following proofs we mostly discuss the simulations of rules of type p, q, f and we let

$$\begin{array}{rclcrcl} M & = & \{m \mid m \in [1 \ldots |P|]\}, & & M' & = & \{m' \mid m \in [1 \ldots |P|]\}, \\ M'' & = & \{m'' \mid m \in [1 \ldots |P|]\}, & & M''' & = & \{m''' \mid m \in [1 \ldots |P|]\}, \\ \mathcal{M}'' & = & M \cup M' \cup M'', & & \mathcal{M}''' & = & M \cup M' \cup M'' \cup M''', \end{array}$$

and similar notations for fourfold and fivefold primed symbols. If no confusion arises, we will use, say, M''' even in the case when, for instance for the simulation of p-rules, no triple-primed markers are necessary. Also, we will simply write $\mathcal{M}$ to refer to all marker symbols altogether.

After this general description, we move on to our more specific results.

We first recall from Ivanov and Verlan (2015) that $\mathrm{SC}_{2,2}\mathrm{ID}(1, 0, 0; 1, 0, 0) = \mathrm{RE}$. In the following we decrease the degree to (2, 1) and further make the system simple but at the cost of increasing the insertion and deletion lengths from one to two.

Theorem 3 $SSC_{2,1}ID(2, 0, 0; 2, 0, 0) = RE$.

Proof Consider a type-0 grammar $G = (N, T, P, S)$ in SGNF in which the rules of P are labelled uniquely by numbers $[1 \ldots |P|]$. We construct an SSCID system $\Pi = (V, T, \{S\}, R)$ of degree (2, 1) and ID size (2, 0, 0; 2, 0, 0) as follows such that $L(\Pi) = L(G)$. The alphabet of Π is $V \subset N \cup T \cup \mathcal{M}$, where $\mathcal{M}$ is the set of all markers; we

$p1 = [(\lambda, pp', \lambda)_{ins}, \emptyset, \mathcal{M}]$
$p2 = [(\lambda, p'X, \lambda)_{del}, \{pp'\}, \emptyset]$
$p3 = [(\lambda, bp'', \lambda)_{ins}, \emptyset, N' \cup (\mathcal{M} \setminus \{p\})]$
$p4 = [(\lambda, Yp''', \lambda)_{ins}, \emptyset, N' \cup (\mathcal{M} \setminus \{p, p''\})]$
$p5 = [(\lambda, p''', \lambda)_{del}, \{p'''p''\}, \emptyset]$
$p6 = [(\lambda, p''p, \lambda)_{del}, \{bY\}, \emptyset]$

(a) Simulating $p : X \to bY$

$q1 = [(\lambda, qq', \lambda)_{ins}, \emptyset, \mathcal{M}]$
$q2 = [(\lambda, q'X, \lambda)_{del}, \{qq'\}, \emptyset]$
$q3 = [(\lambda, q''b, \lambda)_{ins}, \emptyset, N' \cup (\mathcal{M} \setminus \{q\})]$
$q4 = [(\lambda, q'''Y, \lambda)_{ins}, \emptyset, N' \cup (\mathcal{M} \setminus \{q, q''\})]$
$q5 = [(\lambda, q''', \lambda)_{del}, \{q''q'''\}, \emptyset]$
$q6 = [(\lambda, qq'', \lambda)_{del}, \{Yb\}, \emptyset]$

(b) Simulating $q : X \to Yb$

Fig. 1 Simulating context-free rules of SGNF by $\mathrm{SSC}_{2,1}\mathrm{ID}(2,0,0;2,0,0)$

need in particular up to triple-primed versions of markers for context-free rules. The set of rules of R in Π is given as follows. (i) For every rule of type $p : X \to bY$ in G, the simulating rules are stated in Fig. 1a. (ii) For every rule of type $q : X \to Yb$ in G, the simulating rules are stated in Fig. 1(b). (iii) Rules of type $f : AB \to \lambda$ are simulated by the (SSC)ID rule $f1 = [(\lambda, AB, \lambda)_{del}, \emptyset, \emptyset]$.

We now proceed to prove that $L(\Pi) = L(G)$. We initially prove that $L(G) \subseteq L(\Pi)$ by showing that Π correctly simulates the application of the rules of the types p, q, f. We focus on the p rule simulation, as this is the most complicated one.

Simulation of $p : X \to bY$: Consider a sentential form $\alpha X\beta$ derivable in G, where $X \in N'$ and $\alpha, \beta \in (N'' \cup T)^*$. By induction, $\alpha X\beta$ can be also derived in Π. The application of $p : X \to bY$ to $\alpha X\beta$ derives $\alpha bY\beta = w$, which is correctly simulated by Π as follows:

$$\alpha X\beta \Rightarrow_{p1} \alpha pp'X\beta \Rightarrow_{p2} \alpha p\beta \Rightarrow_{p3} \alpha bp''p\beta \Rightarrow_{p4} \alpha bYp'''p''p\beta \Rightarrow_{p5} \alpha bYp''p\beta \Rightarrow_{p6} w.$$

Being slightly different, we also show the simulation of $q : X \to Yb$; here, $\alpha Yb\beta = w'$ is the result of applying q to $\alpha X\beta$.

$$\alpha X\beta \Rightarrow_{q1} \alpha qq'X\beta \Rightarrow_{q2} \alpha q\beta \Rightarrow_{q3} \alpha qq''b\beta \Rightarrow_{q4} \alpha qq''q'''Yb\beta \Rightarrow_{q5} \alpha qq''Yb\beta \Rightarrow_{q6} w'.$$

Observe how the simulation of q *mirrors* that of p, while the logic behind is clearly identical. This proves that $L(G) \subseteq L(\Pi)$.

Simulation idea: We insert strings of length two in a random manner, such that one symbol of it acts as a marker to stitch to the correct position in the string. The correct position is verified with permitting strings or deletion strings of length two, which verifies that the previously introduced string has been inserted only at a particular correct position. For example, pp' is randomly inserted by rule $p1$ and the rule $p2$ demands that this insertion happens to the left of the only non-terminal X present in the string. In particular, this technique prevents us from re-starting the rules on a string consisting of terminal symbols only. Similarly, the permitting string in $p5$ demands to have the substring $p'''p''$ present in the string, thus Yp''' (see rule $p4$) is inserted between b and p'' and bp'' itself is inserted by rule $p3$. The forbidden strings in insertion rules prevent us from using of the same rule again and also indirectly bring the order among the applications of the rules. We now proceed to the formal simulation.

We now prove the converse inclusion $L(\Pi) \subseteq L(G)$ by showing that the rules stated in Fig. 1a can only be used in the intended way, simulating the context-free rule $p : X \to bY$ of G.

Consider a sentential form $w_0 = \alpha X\beta$ derivable in Π and G, where $X \in N'$ and $\alpha, \beta \in (N'' \cup T)^*$. Notice that, from the perspective of G, we are (still) in phase I. The only applicable rule is $p1$ (or any other insertion rule $r1$ where the left-hand side of rule r is X) since other insertion rules like $p3$ or $p4$ forbid the presence of any non-terminal of N'. All deletion rules of Fig. 1 require the presence of rule markers (i.e., elements of $\mathcal{M}$), but $sub(w_0) \cap \mathcal{M} = \emptyset$. On applying the rule $p1$, pp' is inserted anywhere in the string thus yielding $w_1 \in pp' \triangleright \alpha X\beta$, with $pp' \in sub(w_1)$. We cannot apply any insertion rule $r1$, $r3$ or $r4$, as $p' \in \mathcal{F}_{r1} \cap \mathcal{F}_{r3} \cap \mathcal{F}_{r4}$. In particular, this rules out repeated applications of $p1$. Also, we cannot apply rule $h1 = [(\lambda, EF, \lambda)_{del}, \emptyset, \emptyset]$ now, as here (and also in any of the further steps discussed below) some rule marker is present in the string. Hence, we must apply a deletion rule of Fig. 1 to w_1. The application of any $r5$ or $r6$ requires r'' to appear, which is not the case for w_1. By the uniqueness of rule labels, the only applicable rule is $p2$ which actually fixes the position of pp' on the left of X, thereby deleting $p'X$. Hence, we obtain a unique string w_2 satisfying $w_1 \Rightarrow_{p2} w_2 = \alpha p\beta$.

Alternatively, we can think of obtaining w_2 by applying the rewriting rule $X \to p$ to w_0. Now, there is a choice in applying $r3$ or $r4$ for some rule r. We focus on $r = p$ in the following, as this is the only possible fruitful continuation, as we will soon see. If $p4$ is applied to w_2, we get $w_2' \in Yp''' \triangleright \alpha p\beta$ and now $p3$ cannot be applied, as $p''' \in sub(w_2') \cap \mathcal{F}_{p3}$. The derivation is stuck, as no other rule can be applied. In particular, $p5$ is not applicable, since $p'' \notin sub(w_2')$. Thus, the only applicable rule on w_2 is $p3$ which inserts bp'' randomly into w_2 yielding $w_3 \in bp'' \triangleright \alpha p\beta$, with $bp'' \in sub(w_3)$. The re-application of $p3$ on w_3 is stopped since p'' is a member of its forbidden set. On applying the only possible rule $p4$ on w_3,[4] Yp''' is

[4] Again, any $r4$ could be applied, but we will soon see that $r = p$ is enforced.

randomly inserted, resulting in $w_4 \in Yp''' \triangleright bp'' \triangleright \alpha p \beta$, with $Yp''', bp'' \in sub(w_4)$. At this point we note that both bp'' and Yp''' are floating across w_4 and their position is not yet fixed. This is done in the step after the next step by applying the rule $p6$. A careful case analysis reveals that now $p5$ is the only applicable rule.[5] Since $p5$ demands that $p'''p'' \in sub(w_4)$, this crucial rule application fixes several of our previous choices:

(a) Recall that we could have applied any rule $r3$ (instead of $p3$) and any rule $\bar{r}4$ (instead of $p4$). But if we would have chosen $\bar{r} \neq r$, then the substring $r'''r''$ would not be present in w_4. As we promised above, we will see in the next step that only $r = p$ is possible, which we will therefore use already in the following to avoid clumsy formulations.

(b) Previously, we had the choice of inserting Yp''', bp'' anywhere into w_2. However, $p'''p'' \in sub(w_4)$ ensures that Yp''' must have been inserted between b and p''. Hence, we know that $bYp'''p'' \in sub(w_4)$. Now, $w_4 \Rightarrow_{p5} w_5$ yields $bYp'' \in sub(w_5)$. With symbols from $M \cup M''$ being present in w_5, we understand that only rule $p6$ is applicable. Also, the deletion operation fixes that the right-hand side bY introduced with rules $r3$ and $r4$ corresponds to that of p, as this deletion is only possible if $r = p$. Similarly, bp'' must have been inserted to the left of p due to $p''p \in sub(w_5)$. Applying p_6 on w_5 deletes the markers $p''p$, thus yielding $w_6 = \alpha bY\beta$. This series of rule applications that yields $w_6 = \alpha bY\beta$ from $w_0 = \alpha X\beta$ corresponds to the rewriting rule $X \to bY$ of G.

Recall that during the whole sequence of derivation steps that we studied, always a marker from $\mathcal{M}'''$ was present in the string, in particular preventing premature starts of new simulations of some rule $r : Y \to Y_1Y_2$ from G after applying $p4$ or $p5$.

Consider now a sentential form w_0 derivable both in Π and in G, with $N' \cap sub(w_0) = \emptyset$. This means that the derivation of grammar G is in phase II. Hence, $w_0 = xyt$, where $x \in \{A, C\}^*$, $y \in \{B, D\}^*$, $t \in T^*$. Clearly, if $w_0 \in T^*$, no further derivation is possible. If AB or CD are substrings of w_0, we can (directly) apply $f1$ or $g1$, this way removing this substring as intended. Alternatively, we can apply $r1$ for some context-free rule r of G. As we have considered above, we would have to apply $r2$ next, but this is not possible due to the absence of symbols from N'. Hence, any such attempt will get stuck.

By induction, the previous arguments (that basically present the induction steps) show that $L(\Pi) \subseteq L(G)$, thus proving the theorem. □

[5] Again, any $r5$ could be applied, but we will soon see that $r = p$ is enforced.

Remark 2 We like to mention that ins–del systems that work completely without local context are always challenging to handle. There is always the danger of re-starting a derivation, even after having derived a terminal string, because in particular insertion may happen without checking context conditions. Even in our case, we can always insert pp' in such a situation, but afterwards (as the given analysis proves) there is no further continuation. Likewise, we could insert bp'' or Yp''', even both, one after the other, but after possibly deleting p''' with rule $p5$, the derivation is stuck, as no p marker was introduced. But these possibilities (which are easily blocked if local context conditions are present) already indicate that rules have to be designed carefully.

The following result is not just a simple modification of the previous one, as it is sometimes the case when trading the possibility of deleting substrings of length two for deleting single symbols in one-sided contexts. This can be seen not only by looking at the simulation rules, which are much more elaborate, using more markers, etc., but this can be also seen by the fact that now we need the new normal form called ssSGNF above. In fact, we could have shortened the whole presentation of this paper a bit by replacing the simulation of the context-free rules shown in Fig. 1 by that of Fig. 2 in the previous theorem; yet, we found it instructive to see that the simulation in Fig. 1 is much simpler than that in Fig. 2, hence proving a kind of trade-off between the resources that we are actually measuring in this paper and the number of rules, or also the number of markers, that are required by the said simulations.

Theorem 4 $SSC_{2,1}ID(2, 0, 0; 1, 0, 1) = SSC_{2,1}ID(2, 0, 0; 1, 1, 0) = RE$.

Proof Consider a type-0 grammar $G = (N, T, P, S)$ in ssSGNF. The rules of P are labelled uniquely by numbers $[1 \ldots |P|]$. We construct an SSCID system $\Pi = (V, T, \{S\}, R)$ of degree (2, 1) and ID size (2, 0, 0; 1, 0, 1) as follows such that $L(\Pi) = L(G)$. The alphabet of Π is $V \subset N \cup T \cup \mathcal{M}$, with $\mathcal{M} := (M \cup M' \cup M'' \cup M''' \cup M^{iv} \cup M^v) \setminus \{f^{iv}, g^{iv}, h^{iv}, f^v, g^v, h^v\}$. The set of rules R of Π is given as follows:

1. For every rule of type $p : X \to bY$ in G, the simulating rules are described in Fig. 2a.
2. For every rule of type $q : X \to Yb$ in G, the simulating rules are listed in Fig. 2b.
3. Rules like $f : AB \to \lambda$ in G are simulated by rules as given in Fig. 3.

Let us first see how the intended derivations are.

Simulating $p : X \to bY$: See Fig. 2a. Consider a sentential form $\alpha X \beta t$ of G, with $\alpha, \beta \in (N'')^*$, $t \in T^*$. The

$p1 = [(\lambda, pp', \lambda)_{ins}, \emptyset, (N' \setminus \{X\}) \cup \mathcal{M}]$
$p2 = [(\lambda, bp'', \lambda)_{ins}, \emptyset, (N' \setminus \{X\}) \cup \mathcal{M} \setminus \{p, p'\}]$
$p3 = [(\lambda, p'''p^{iv}, \lambda)_{ins}, \emptyset, (N' \setminus \{X\}) \cup \mathcal{M} \setminus \{p, p', p''\}]$
$p4 = [(\lambda, p'', \lambda)_{del}, \{Xp, pb, bp''', p'''p^{iv}, p^{iv}p'', p''p'\}, \emptyset]$
$p5 = [(\lambda, X, \lambda)_{del}, \{Xp, pb, bp''', p'''p^{iv}, p^{iv}p'\}, \emptyset]$
$p6 = [(\lambda, p^{iv}, \lambda)_{del}, \emptyset, N' \cup \mathcal{M}^v \setminus \{p, p', p''', p^{iv}\}]$
$p7 = [(\lambda, Yp^v, \lambda)_{ins}, \emptyset, N' \cup \mathcal{M}^v \setminus \{p, p', p'''\}]$
$p8 = [(\lambda, p^v, \lambda)_{del}, \{pb, bp''', p'''Y, Yp^v, p^vp'\}, \emptyset]$
$p9 = [(\lambda, p''', \lambda)_{del}, \{pb, bp''', p'''Y, Yp'\}, \emptyset]$
$p10 = [(\lambda, p', \lambda)_{del}, \{pb, bY, Yp'\}, \emptyset]$
$p11 = [(\lambda, p, \lambda)_{del}, \emptyset, (N' \setminus \{Y\}) \cup \mathcal{M} \setminus \{p\}]$

(a) Simulating $p : X \to bY$

$q1 = [(\lambda, qq', \lambda)_{ins}, \emptyset, (N' \setminus \{X\}) \cup \mathcal{M}]$
$q2 = [(\lambda, q''b, \lambda)_{ins}, \emptyset, (N' \setminus \{X\}) \cup \mathcal{M} \setminus \{q, q'\}]$
$q3 = [(\lambda, q'''q^{iv}, \lambda)_{ins}, \emptyset, (N' \setminus \{X\}) \cup \mathcal{M} \setminus \{q, q', q''\}]$
$q4 = [(\lambda, q'', \lambda)_{del}, \{Xq, qq'', q''q''', q'''q^{iv}, q^{iv}b, bq'\}, \emptyset]$
$q5 = [(\lambda, X, \lambda)_{del}, \{Xq, qq''', q'''q^{iv}, q^{iv}b, bq'\}, \emptyset]$
$q6 = [(\lambda, q^{iv}, \lambda)_{del}, \emptyset, N' \cup \mathcal{M}^v \setminus \{q, q', q''', q^{iv}\}]$
$q7 = [(\lambda, Yq^v, \lambda)_{ins}, \emptyset, N' \cup \mathcal{M}^v \setminus \{q, q', q'''\}]$
$q8 = [(\lambda, q^v, \lambda)_{del}, \{qY, Yq^v, q^vq''', q'''b, bq'\}, \emptyset]$
$q9 = [(\lambda, q''', \lambda)_{del}, \{qY, Yq''', q'''b, bq'\}, \emptyset]$
$q10 = [(\lambda, q', \lambda)_{del}, \{qY, Yb, bq'\}, \emptyset]$
$q11 = [(\lambda, q, \lambda)_{del}, \emptyset, (N' \setminus \{Y\}) \cup \mathcal{M} \setminus \{q\}]$

(b) Simulating $q : X \to Yb$

Fig. 2 Simulating context-free rules of SGNF by $\mathrm{SSC}_{2,1}\mathrm{ID}(2,0,0;1,0,0)$

following sequence of rule applications simulates the application of the rule $p : X \to bY$.

$$\alpha X\beta t \Rightarrow_1 \alpha Xpp'\beta t \Rightarrow_2 \alpha Xpbp''p'\beta t \Rightarrow_3 \alpha Xpbp'''p^{iv}p''p'\beta t$$
$$\Rightarrow_4 \alpha Xpbp'''p^{iv}p'\beta t \Rightarrow_5 \alpha pbp'''p^{iv}p'\beta t \Rightarrow_6$$

$$\alpha pbp'''p'\beta t \Rightarrow_7 \alpha pbp'''Yp^vp'\beta t \Rightarrow_8 \alpha pbp'''Yp'\beta t$$
$$\Rightarrow_9 \alpha pbYp'\beta t \Rightarrow_{10} \alpha pbY\beta t \Rightarrow_{11} \alpha bY\beta t$$

Simulating $q : X \to Yb$: See Fig. 2b. Consider a sentential form $\alpha X\beta t$ of G, with $\alpha, \beta \in (N'')^*$, $t \in T^*$. The following sequence of rule applications simulates the application of the rule $q : X \to Yb$.

$$\alpha X\beta t \Rightarrow_1 \alpha Xqq'\beta t \Rightarrow_2 \alpha Xqq''bq'\beta t \Rightarrow_3 \alpha Xqq''q'''q^{iv}bq'\beta t$$
$$\Rightarrow_4 \alpha Xqq'''q^{iv}bq'\beta t \Rightarrow_5 \alpha qq'''q^{iv}bq'\beta t \Rightarrow_6 \alpha qq'''bq'\beta t$$
$$\Rightarrow_7 \alpha qYq^vq'''bq'\beta t \Rightarrow_8 \alpha qYq'''bq'\beta t \Rightarrow_9 \alpha qYbq'\beta t$$
$$\Rightarrow_{10} \alpha qYb\beta t \Rightarrow_{11} \alpha Yb\beta t$$

Simulating $f : AB \to \lambda$: See Fig. 3. Consider a sentential form $\alpha AB\beta t$ of G, with $\alpha, \beta \in (N'')^*$, $t \in T^*$, where α does not end with A and β does not start with B. The following sequence of rule applications simulates the application of this rule.

$$\alpha AB\beta t \Rightarrow_1 \alpha fAB\beta t \Rightarrow_2 \alpha fAf'B\beta t \Rightarrow_3 \alpha fAf'Bf''f'''\beta t \Rightarrow_4$$
$$\alpha fAf'Bf'''\beta t \Rightarrow_5 \alpha fAf'f'''\beta t \Rightarrow_6 \alpha ff'f'''\beta t \Rightarrow_7 \alpha ff'''\beta t$$
$$\Rightarrow_8 \alpha f\beta t \Rightarrow_9 \alpha\beta t$$

$f1 = [(\lambda, f, \lambda)_{ins}, \emptyset, N' \cup \mathcal{M}]$
$f2 = [(\lambda, f', \lambda)_{ins}, \emptyset, N' \cup \mathcal{M} \setminus \{f\}]$
$f3 = [(\lambda, f''f''', \lambda)_{ins}, \emptyset, N' \cup \mathcal{M} \setminus \{f, f'\}]$
$f4 = [(\lambda, f'', \lambda)_{del}, \{fA, Af', f'B, Bf''\}, \emptyset]$
$f5 = [(\lambda, B, f''')_{del}, \{fA, Af', f'B, Bf'''\}, \emptyset]$
$f6 = [(\lambda, A, f')_{del}, \{fA, Af', f'f'''\}, \emptyset]$
$f7 = [(\lambda, f', \lambda)_{del}, \{ff', f'f'''\}, \emptyset]$
$f8 = [(\lambda, f''', \lambda)_{del}, \{ff'''\}, \emptyset]$
$f9 = [(\lambda, f, \lambda)_{del}, \emptyset, N' \cup (\mathcal{M} \setminus \{f\})]$

Fig. 3 Simulating $f : AB \to \lambda$ by $\mathrm{SSC}_{2,1}\mathrm{ID}(2,0,0;1,0,1)$

The context-free deletion rules $S' \to \lambda$ (and possibly $S \to \lambda$) can be trivially simulated.

Let us now prove the converse inclusion. Recall that due to the properties of ssSGNF, only very special strings can be derived by a ssSGNF grammar. In our inductive argument, we assume that we start with such a string w and we assume that w can be also derived by the constructed SSC ins–del system. We then prove that the derivation of the SSC ins–del system either gets stuck or produces a string that could have been also derived in the ssSGNF grammar. By induction, this proves that the constructed SSC ins–del system cannot derive any terminal strings that do not belong to the originally given RE language.

General Observation: There is a certain danger when applying deletion rules in a context-free fashion, even if there are global permitting context conditions. Namely, if the symbol to be deleted occurs arbitrarily often in the current string, then the deletion can occur multiple times in general, because we cannot shield all occurrences with a finite number of permitting contexts only. Therefore, only the simulations for p-rules or q-rules could delete the symbols in a context-free way, because all the deleted symbols that are deleted in these simulations occur at most once in a sentential form. However, when deleting, say, A's or B's in the simulation of the f-rule, we need to do this at least with one-sided context to make sure that the correct occurrence of A or B is deleted. We will employ this observation in the following quite frequently, mostly without saying.

Case 1: Assume that $w \in \{EA, EC\}^* N_S T^*$. Hence, $w = \alpha Xt$ for some $\alpha \in \{EA, EC\}^*$, $X \in N_S$, $t \in T^*$. As all deletion rules require the presence of markers,[6] only insertion rules might apply. The insertion rules introduced for simulating rules f, g, h of the ssSGNF grammar do not apply, because they forbid the presence of symbols from $N' \supset N_S$. Only if the correct symbol $X \in N_S$ is present, insertion rules labeled $r1$, $r2$, $r3$ are applicable, where r is a context-free rule with left-hand side X. Observe that if we apply $r2$, this blocks applying $r1$ (before erasing the marker r''). However, we can delete the marker r'' only if marker r'

[6] with the trivial exception of the simulation of $S' \to \lambda$ (and possibly $S \to \lambda$)

is present, which requires $r1$ to be applied before applying $r2$. Similarly, we can rule out applying $r3$ first. The presence of X also blocks the insertion rule $p7$. Hence, we have to apply Rule $r1$, and this blockage remains true until we delete X. As now details become different, we will differentiate between rules of type p and q from now on. However, notice that once some marker like p is introduced, this rules out any simulations different from p because of the respective permitting or forbidden contexts.

Simulating $p : X \to bY$: As we have to apply $p1$, some string w_1 from $pp' \triangleright \alpha Xt$ results. Let us first check out possibly applicable deletion rules. The deletion rule $p5$ that deals with X requires all markers but p'' to be present and is hence not applicable. The deletion rule $p10$ dealing with p' is not applicable either, as it requires some non-marker b being to the right of p. Finally, the deletion rule $p11$ is not applicable, because all markers but p are forbidden contexts, while p' is contained in w_1. Our previous discussion shows that $w_1 \Rightarrow_{p2} w_2$ is enforced. Hence, $w_2 \in bp'' \triangleright pp' \triangleright \alpha Xt$. The previous arguments carry over and show that neither $p5$ nor $p10$ nor $p11$ are applicable. Also $p4$ (for deleting p'') is inapplicable, as the marker p''' is missing. Hence, necessarily $w_2 \Rightarrow_{p3} w_3$, i.e., $w_3 \in p'''p^{iv} \triangleright bp'' \triangleright pp' \triangleright \alpha Xt$. Clearly, no more insertion rules are applicable. Because of the presence of X, we cannot apply Rules $p6$ and $p7$. As $X \neq Y$, if X is present, Y cannot be present, so that $p8$, $p9$, $p10$ are not applicable either. As markers different from p occur, $p11$ is inapplicable. If we try to apply $p4$ or $p5$, then pp' must have been inserted to the right of X due to the permitting context Xp. Hence, $w_3 \in p'''p^{iv} \triangleright bp'' \triangleright \alpha Xpp't$. The permitting context pb (both for $p4$ and for $p5$) is only possible if bp'' was inserted inbetween p and p', i.e., $w_3 \in p'''p^{iv} \triangleright \alpha Xpbp''p't$. The permitting context bp''' enforces that $w_3 = \alpha Xpbp'''p^{iv}p''p't$. Now, Rule $p4$ deviates from $p5$. If we want to apply $p5$, then the permitting context $p^{iv}p'$ is not present in w_3. In conclusion, we cannot apply $p5$ to w_3. If we follow the only remaining alternative, we have to apply Rule $p4$. As it is easy to check, the permitting context are all satisfied. Hence, $w_3 \Rightarrow_{p4} w_4 = \alpha Xpbp'''p^{iv}p't$. A reasoning very similar to the one given just before shows that now $p5$ must be applied, leading to $w_4 \Rightarrow_{p5} w_5 = \alpha pbp'''p^{iv}p't$. The absence of p'' and X and the presence of p^{iv} block all rules except for Rule $p6$. Hence, $w_5 \Rightarrow_{p6} w_6 = \alpha pbp'''p't$. The absence of p'', p^{iv} and X and the presence of p''' to the left of p' block all rules except for Rule $p7$. This leads to $w_6 \Rightarrow_{p7} w_7 \in Yp^v \triangleright \alpha pbp'''p't$. The presence of p^v in w_7 rules out all rules with forbidden contexts. The absence of p^{iv} leaves us with $p8$, $p9$ or $p10$. However, wherever we insert Yp^v, to the right of Y, we won't find p' as required by $p9$ and $p10$. Hence, we have to apply Rule $p8$, which also fixes the position where Yp^v is inserted. Therefore, $w_7 = \alpha pbp'''Yp^vp't \Rightarrow_{p8} w_8 = \alpha pbp'''Yp't$. The presence of Y (ruling out X) to the right of p''' forces us to apply Rule $p9$, followed by $p10$ for similar reasons: $w_8 \Rightarrow_{p9} w_9 = \alpha pbYp't \Rightarrow_{p10} w_{10} = \alpha pbYt$. The presence of p, Y together rule out all insertion rules. Rules that might delete Y would require a rule marker to the right of Y. Hence, the only possible continuation has to delete p, using Rule $p11$. This results in $w_{11} = \alpha bYt$. Clearly, the transformation of w into w_{11} truthfully models the application of Rule p of G. Notice that $Y \in N'$ but not necessarily $Y \in N_S$.

Simulating $q : X \to Yb$: Consider $w \Rightarrow_{q1} w_1 \Rightarrow_{q2} w_2 \Rightarrow_{q3} w_3$. As before, we can rule out using Rule $q11$ on w_1, w_2, w_3. Hence, $w_3 \in q'''q^{iv} \triangleright q''b \triangleright qq' \triangleright \alpha Xt$. With similar arguments as in the p-case, Rules $q6$ through $q10$ are disabled. Trying to apply $q4$ or $q5$ on w_3 fixes the structure of w_3 due to the permitting strings. $Xq \in sub(w_3)$ means that qq' was inserted immediately to the right of X. $qq'' \in sub(w_3)$ tells us that $q''b$ was inserted inbetween q and q' (assuming we try to apply Rule $q4$), while $qq''' \in sub(w_3)$ tells us that $q'''q^{iv}$ was inserted after q (assuming we try to apply Rule $q5$). In the latter case, however, there is no way to produce the substring $q^{iv}b$, as $q'''q^{iv}$ has been inserted to the left of $q''b$. Therefore, we must use Rule $q4$, leading us to conclude that $w_3 = \alpha Xqq''q'''\ q^{iv}\ bq't \Rightarrow_{q4} w_4 = \alpha Xqq'''q^{iv}bq't$. Now, $w_4 \Rightarrow_{q5} w_5 = \alpha qq'''q^{iv}bq't$ is enforced. As in the p-case, we can argue that Rules $q6$ and $q7$ must be applied now, leading us to $w_7 \in Yq^v \triangleright \alpha qq'''bq't$. We can argue as before that now Rule $q8$ must be applied, which not only deletes q^v but also fixes the position of the inserted string Yq^v, so that we arrive at $w_8 = \alpha qYq'''bq't$. The scenery is now completely symmetric to the p-case, and analogous arguments prove that in fact the rule $q : X \to Yb$ of G has to be faithfully simulated.

Case 2: Assume that $w \in \{EA, EC\}^* N_{S'} \{BF, DF\}^* T^*$. As the presence or absence of symbols from $\{B, D, F\}$ was never an issue in the analysis presented in Case 1, nor was the question if $X \in N_S$ (or $X \in N_{S'}$), only the fact that $X \in N'$ was of interest, all arguments given under Case 1 transfer to this case. Notice that now, $Y \in N_{S'}$ is necessarily the case.

Case 3: Assume that $w \in \{EA, EC\}^* \{\lambda, EF\} \{BF, DF\}^* T^*$. The main subcase distinction is whether w matches (i.e., $w \in \{EA, EC\}^* \{EABF, ECDF, EF\} \{BF, DF\}^* T^*$ or not (i.e., $w \in \{EA, EC\}^* \{EADF, ECBF\} \{BF, DF\}^* T^*$), or if $w \in T^*$ (terminal case). In each case, we have the following argument: If we apply $r1$, $r2$, $r3$ (in this order, as explored above) corresponding to a context-free rule r, there is no continuation with $r4$, because the left-hand side of Rule r is obviously missing. We still

might apply Rule $r6$, but now we are finally stuck: neither can we delete any markers anymore, nor can we introduce some nonterminal (which would be the really bad case, because then all of a sudden some continuation might be possible).

The matching case: Without loss of generality, consider $w = \alpha EABF\beta t$, with $\alpha \in \{EA, EC\}^*$, $\beta \in \{BF, DF\}^*$, $t \in T^*$. If we try to apply, say $h1$ (followed necessarily by $h2$ and $h3$, as we will see with a similar discussion on f soon), we arrive at $h''h''' \triangleright h' \triangleright h \triangleright \alpha EABF\beta t$. Then, we need to delete h'' before being able to continue, but this requires as permitting contexts hE, Eh' and $h'F$, which means that w must have EF as a subword, which is not the case.

Hence, we have to apply one of the Rules $f1$, $f2$, or $f3$. If we apply $f2$ first, then we cannot apply $f1$ thereafter as all markers are blocked in $f1$, but as all deleting rules require f to be present, such a derivation cannot terminate, as in particular f' cannot be deleted. The same argument is valid when we start with $f3$. Hence, $w \Rightarrow_{f1} w_1 \in f \triangleright w$ is enforced. We could apply Rule $f9$ now, but as $w_1 \Rightarrow_{f9} w$, we will see no progress, so we can omit discussing this case. If we apply Rule $f3$ immediately, we cannot delete f'', as this requires the presence of f'. Hence $w_1 \Rightarrow_{f2} w_2 \in f' \triangleright w_1$ is enforced. As we cannot delete f' due to the absence of f''', nor can we delete f in the presence of f', we must apply Rule $f3$ next. This leads us to $w_3 \in f''f''' \triangleright w_2$. Now, deletion rules must be applied. Notice that currently $f''f''' \in sub(w_3)$. This disables all deletion rules but Rule $f4$. However, applying this rule also fixes the places where the insertions occurred, because there is only one way how all the subwords $fA, Af', f'B, Bf''$ could have been produced. Namely, $w_3 = \alpha fAf'Bf''f'''\beta t$. Notice that due to ssSGNF, we do not have the possibility of longer sequences of A's or B's occurring as subwords of w. Therefore, cases like $w_3 = \alpha fAAf'Bf''f'''\beta t$ are impossible, which is good, as otherwise Rule $f6$ might be applied twice (after still applying $f4$ and $f5$ first), leading to malicious derivations.

Hence, $w_3 \Rightarrow_{f4} w_4 = \alpha fAf'Bf'''\beta t$. Clearly, no insertion rules are applicable. The subword Bf''' enables exactly one deletion rule, which is Rule $f5$. By our general observation, $w_4 \Rightarrow_{f5} w_5 = \alpha fAf'f'''\beta t$ is enforced, because we can delete only the occurrence of B in the correct context. Similarly, the subwords Af' and $f'f'''$ enforce $w_5 \Rightarrow_{f6} w_6 = \alpha ff'f'''\beta t$, while subwords ff' and $f'f'''$ enforce $w_6 \Rightarrow_{f7} w_7 = \alpha ff'''\beta t$. Finally, subword ff''' triggers $w_7 \Rightarrow_{f8} w_8 = \alpha f\beta t$. It might be tempting to leap into another simulation cycle by continuing with Rule $f2$, as marker f is already present, but as our previous analysis shows, f should be placed to the left of an A (as otherwise the deletion rules would not work), but this is not possible, as β does not contain any occurrences of A. Therefore, we need to continue with Rule $f9$, yielding $w_8 \Rightarrow_{f9} w_9 = \alpha\beta t$. Obviously, the (enforced) transition from w to w_9 models the non-context-free deletion rule $AB \to \lambda$.

The non-matching case: Without loss of generality, consider $w = \alpha EADF\beta t$, with $\alpha \in \{EA, EC\}^*$, $\beta \in \{BF, DF\}^*$, $t \in T^*$. Notice that according to our previous discussions, we should start by applying an insertion rule like $f1$, $g1$, or $h1$. Once we applied, say, $f1$, the presence of the marker f blocks other marker insertions, like $g2$. However, also as argued above, if we start inserting markers f and f', we also have to insert $f''f'''$, as otherwise no deletion rules would apply. Yet, after applying Rules $f1$, $f2$ and $f3$, the contexts $f'f'''$ or ff''' required by the deletion rules $f7$ or $f8$, respectively, require to delete f'' first. This in turn means that we need subwords Af' and $f'B$, i.e., we find the subword $Af'B$ as we could not have introduced two occurrences of f'. However, this implies that $AB \in sub(w)$, which is not true in our case. A similar contradiction, this time conerning the conclusion $CD \in sub(w)$, could be drawn when assuming that we applied Rules $g1$, $g2$ and $g3$.

The terminal case: We have to discuss possible re-starts with insertion rules and prove that they get (finally) stuck. We already discussed the rules stemming from simulating context-free rules. Alternatively, if we use $f1$, $f2$ and $f3$, we will also get stuck, because any other rules will try to look for A or B, which are not present within $w \in T^*$. This is likewise true for the other non-context-free erasing rules.

Altogether, by induction, this shows that the simulation is correct.

The second claim, which is $\mathrm{SSC}_{2,1}\mathrm{ID}(2,0,0;1,1,0) = \mathrm{RE}$, follows again with Theorem 2 from the previous considerations. □

Next, we recall from Ivanov and Verlan (2015) that $\mathrm{SC}_{1,1}\mathrm{ID}(1,1,0;2,0,0) \neq \mathrm{RE}$. In the following we show that computational completeness can be achieved if we increase the degree of the system from (1, 1) to (2, 1), even when maintaining simplicity.

Theorem 5 $SSC_{2,1}ID(1,1,0;2,0,0) = SSC_{2,1}ID\ (1,0,1;2,0,0) = RE$.

The reader might wonder why we could not deduce this result by sequentializing the construction of Theorem 3 or even by starting from a $\mathrm{SSC}_{2,1}\mathrm{ID}(2,0,0;2,0,0)$ system. In fact, as long as special symbols like rule labels are introduced as in rule $p1$ in Fig. 1a, where a string of two rule labels is inserted (in this example pp') we might do the following. First, introduce the left one of them (in this example it is p) with the context conditions of the previous simulation (in this example it is $\mathcal{M}'''$), and then introduce the right one (in this example it is p') in the context of the left one (in this example it is p). One can avoid repetitions

by having this newly introduced marker (in this example it is p') in the forbidden context. This trick can only work if we do not expect that this symbol (that we now check for not showing up in the string) may not already be present in the string. In our example we do not expect p' to be present before we introduced it, so we can sequentialize $p1$ in the described way. However, this expectation is not met, for instance, when trying to sequentialize rule $p3$ in Fig. 1a in a similar fashion. Here, we would need different ideas. In more general terms, this prevents us from starting out from a $\mathrm{SSC}_{2,1}\mathrm{ID}(2,0,0;2,0,0)$ system in our simulation for proving the claimed computational completeness result for $\mathrm{SSC}_{2,1}\mathrm{ID}(1,1,0;2,0,0)$. Hence, we now show a different simulation, starting from type-0 grammars in SGNF again.

Proof Consider a type-0 grammar $G = (N, T, P, S)$ in SGNF. The rules of P are labelled uniquely by numbers $[1 \ldots |P|]$. We construct an SSCID system $\Pi = (V, T, \{S\}, R)$ of degree (2, 1) and ID size (1, 1, 0; 2, 0, 0) as follows such that $L(\Pi) = L(G)$. The alphabet of Π is $V \subset N \cup T \cup \mathcal{M}''$. The set of rules R of Π is given as follows:

1. For every rule of type $p : X \to bY$ in G, the simulating rules are stated in Fig. 4a.
2. For every rule of type $q : X \to Yb$ in G, the simulating rules are stated in Fig. 4b.
3. Rules of type $f : AB \to \lambda$ is simulated by the SSCID rule $f1 = [(\lambda, AB, \lambda)_{del}, \emptyset, \emptyset]$.

We first explain the idea behind the construction of q rule simulation in Π as follows. We introduce three markers q, q', q'' in order to have $qq'q''$ present in the string. The X of N' is deleted before q' is introduced. So, the effect of executing $q1$ through $q4$ is the same as that of applying the rewriting rule $X \to qq'q''$. Then, Y is inserted in between q, q' and b is inserted in between q' and q''. Note that b cannot be introduced for a second time, as the string will be having $q'bq''$ and not $q'q''$ (see rule $q5$). On deleting the markers, first q is deleted in the presence of the Yq' and bq'' to ensure that Y and b are correctly introduced. Then, the markers q' and q'' are deleted in this order. The order of deletion is important since otherwise, the rules $q3$ and/or $q4$ can be applied again and a malicious string can be obtained by using the rules $q5$ and/or $q6$.

One can show that $L(G) \subseteq L(\Pi)$ by an inductive argument. The anchor being trivially given by construction. Consider some string w that is derivable both in G and in Π (by induction hypothesis). If w was produced in phase I of the SGNF grammar G, then $w = \alpha X \beta$ for some $\alpha, \beta \in (T \cup N'')^*$ and $X \in N'$.

Simulation of $q : X \to Yb$: The application of $q : X \to Yb$ in phase I is simulated by rules of Π as follows:

$$
\begin{array}{l}
p1 = [(X, p, \lambda)_{ins}, \emptyset, \mathcal{M}''] \\
p2 = [(\lambda, X, \lambda)_{del}, \{p\}, \emptyset] \\
p3 = [(p, p', \lambda)_{ins}, \emptyset, N' \cup (\mathcal{M}'' \setminus \{p\})] \\
p4 = [(p', p'', \lambda)_{ins}, \emptyset, N' \cup (\mathcal{M}'' \setminus \{p, p'\})] \\
p5 = [(p', Y, \lambda)_{ins}, \{p'p''\}, \emptyset] \\
p6 = [(p, b, \lambda)_{ins}, \{pp'\}, \emptyset] \\
p7 = [(\lambda, p, \lambda)_{del}, \{bp', Yp''\}, \emptyset] \\
p8 = [(\lambda, p', \lambda)_{del}, \emptyset, M] \\
p9 = [(\lambda, p'', \lambda)_{del}, \emptyset, M \cup M']
\end{array}
$$

(a) Simulating $p : X \to bY$

$$
\begin{array}{l}
q1 = [(X, q, \lambda)_{ins}, \emptyset, \mathcal{M}''] \\
q2 = [(\lambda, X, \lambda)_{del}, \{q\}, \emptyset] \\
q3 = [(q, q', \lambda)_{ins}, \emptyset, N' \cup (\mathcal{M}'' \setminus \{q\})] \\
q4 = [(q', q'', \lambda)_{ins}, \emptyset, N' \cup (\mathcal{M}'' \setminus \{q, q'\})] \\
q5 = [(q', b, \lambda)_{ins}, \{q'q''\}, \emptyset] \\
q6 = [(q, Y, \lambda)_{ins}, \{qq'\}, \emptyset] \\
q7 = [(\lambda, q, \lambda)_{del}, \{Yq', bq''\}, \emptyset] \\
q8 = [(\lambda, q', \lambda)_{del}, \emptyset, M] \\
q9 = [(\lambda, q'', \lambda)_{del}, \emptyset, M \cup M']
\end{array}
$$

(b) Simulating $q : X \to Yb$

Fig. 4 Simulation of context-free rules of SGNF by $\mathrm{SSC}_{2,1}\mathrm{ID}(1,1,0;1,0,0)$

$$\alpha X\beta \Rightarrow_{q1} \alpha Xq\beta \Rightarrow_{q2} \alpha q\beta \Rightarrow_{q3} \alpha qq'\beta \Rightarrow_{q4} \alpha qq'q''\beta \Rightarrow_{q5} \alpha qq'bq''\beta \Rightarrow_{q6} \alpha qYq'bq''\beta \Rightarrow_{q7} \alpha Yq'bq''\beta \Rightarrow_{q8} \alpha Ybq''\beta \Rightarrow_{q9} \alpha Yb\beta.$$

Clearly, rules of type p work alike. The rule $S' \to \lambda$ that transfers to phase II and the phase II rules $AB \to \lambda$ and $CD \to \lambda$ can be directly simulated. By induction, the claim follows.

To show the converse inclusion $L(G) \supseteq L(\Pi)$, consider a string w_0 derivable both in G and in Π. We discuss possible derivations for w_0 in Π and have to show that these either get stuck or correspond to derivation steps in G, which would then entail the claim by induction. Observe that any rules r_j for $j > 1$ require that $sub(w_0) \cap \mathcal{M}'' \neq \emptyset$, either by the permitting context, or because this is a requirement of the ins–del rules themselves. Hence, if $N' \cap sub(w_0) = \emptyset$, i.e., the SGNF grammar G would work in phase II, we have to apply one of $h1, f1, g1$, which directly corresponds to an erasing rule of G.

Therefore, we now consider a sentential form $w_0 = \alpha X \beta$ derivable in Π and G, where $X \in N'$ and $\alpha, \beta \in (N'' \cup T)^*$. The only applicable rules are some rules $q1$ that insert the marker q to the right of X, thus yielding $w_1 = \alpha Xq\beta$. Notice that now (and also within the future discussions) always a marker from $\mathcal{M}''$ is present in the string, which disables applying rule $h1$ prematurely. No rule $r3$ is applicable, as $N' \cap sub(w_1) \neq \emptyset$. For any of the rules $r4$, $r5$, $r6$, $r7$, $r8$ to be applicable, $M' \cap sub(w_1) \neq \emptyset$ is necessary, which is not the case. Similarly, $r9$ is not applicable. Hence, the only applicable rule is $q2$ which deletes X yielding the string $w_2 = \alpha q\beta$. Again, none of the rules $r4$, $r5$, $r6$, $r7$, $r8$ is applicable, as $M' \cap sub(w_1) = \emptyset$. The presence of the marker q disables $r1$ and $r9$. As $N' \cap sub(w_1) = \emptyset$, no rule $r2$ is applicable. Due to the uniqueness of the rule labels, $q3$

is hence the only applicable rule, with $w_2 \Rightarrow_{q3} w_3 = \alpha qq'\beta$. As q, q' are present in w_3, any rule like $r1$, $r3$, $r8$, $r9$ is disabled. The absence of symbols from $N' \cup M''$ disables applying $r2$, $r4$, $r5$, $r7$. Label uniqueness leaves us with applying either $q4$ or $q6$. Hence, if $w_3 \Rightarrow w_4$ in Π, then $w_4 \in \{\alpha qq'q''\beta, \alpha qYq'\beta\}$. If $w_4 = \alpha qYq'\beta$, a case analysis reveals that if $w_4 \Rightarrow w_5$ in Π, then this must be due to applying q_4, i.e., $w_5 = \alpha qYq'q''\beta$. Now, $q5$ is the only applicable rule, so that $w_6 = \alpha qYq'bq''\beta$ is enforced. Alternatively, on $w_4 = \alpha qq'q''\beta$, only rules $q5$ and $q6$ can apply. However, the order of application of $q5$, $q6$ does not matter, because if $q5$ is applied, then only $q6$ can be applied next, and vice versa. Hence, if $w_4 \Rightarrow w_5 \Rightarrow w_6$ in Π, $w_6 = \alpha qYq'bq''\beta$ is again enforced.

The presence of symbols from M, M', M'' and N' in the substring $qYq'bq'$ within w_6 prevents applying any of the insertion rules, as well as of any of $r8$ or $r9$. Because we can assume that $X \neq Y$ in any rule $q : X \to Yb$ or $p : X \to bY$ of G, no rule $r2$ can be applied at this point. The only applicable rule on w_6 is hence $q7$ which deletes the marker q, thus yielding $w_7 = \alpha Yq'bq''\beta$. Let us stress that $q7$ could not have been applied at any earlier point, as it also checks that both Yq' and bq'' are present within the sentential form. Following the application of $q7$, the rules $q8, q9$ are applied in a deterministic way which will delete the markers q', q'', respectively, from w_7 thus finally yielding $w_9 = \alpha Yb\beta$. A case-by-case analysis shows that no other rules are applicable within a derivation $w_7 \Rightarrow w_8 \Rightarrow w_9$ within Π. This series of rule applications yielding $w_9 = \alpha Yb\beta$ from $w_0 = \alpha X\beta$ corresponds to the rewriting rule $X \to Yb$.

The second claim $\mathrm{SSC}_{2,1}\mathrm{ID}(1,0,1;2,0,0) = \mathrm{RE}$ follows now with Theorem 2. □

It is shown in (Ivanov and Verlan 2015, Theorem 4) that $\mathrm{SC}_{1,1}\mathrm{ID}(1,1,0;1,1,1) \neq \mathrm{RE}$. Analogous to the previous theorem, we show in the following that computational completeness of the system with ID (1, 1, 0; 1, 1, 1) can be achieved if we increase the degree of the systems from (1, 1) to (2, 1). We prove the result even for simple semi-conditional ins–del systems. Thus, the ID size in the following result is optimal for degree (2, 1).

Theorem 6 $\mathrm{SSC}_{2,1}\mathrm{ID}(1,1,0;1,1,0) = \mathrm{SSC}_{2,1}\mathrm{ID}(1,0,1;1,0,1) = \mathrm{RE}$.

Proof Clearly, we only have to show that $\mathrm{SSC}_{2,1}\mathrm{ID}(1,1,0;1,1,0) = \mathrm{RE}$, because the other equality follows with Theorem 2. Let us now sketch how to simulate a type-0 grammar in ssSGNF by an SSC ins–del system of ID size (1, 1, 0; 1, 1, 0) and of degree (2, 1). Recall that we only have to describe how to simulate the context-free rules of types p and q, as well as how to simulate the non-context-free deletion rules like $f : AB \to \lambda$. For the first item, we refer to Fig. 4 and the corresponding explanations from the proof of Theorem 5. Also recall how the proper use of markers prohibited mixing the intended strands of simulation with other rule applications. These arguments are also valid in this case, where we now employ the simulation rules listed in Fig. 5a in order to simulate $f : AB \to \lambda$. Let us explain the intended simulation in the following. Consider $w = \alpha AB\beta t$. We expect that $\alpha\beta t$ should be derived after some steps.

$$
\begin{aligned}
w &\Rightarrow_{f1} \alpha fAB\beta t \Rightarrow_{f2} \alpha fAf'B\beta t \Rightarrow_{f3} \alpha fAf'Bf''\beta t \\
&\Rightarrow_{f4} \alpha fAf'Bf''f'''\beta t \Rightarrow_{f5} \alpha fAf'Bf'''\beta t \Rightarrow_{f6} \alpha fAf'f'''\beta t \\
&\Rightarrow_{f7} \\
&\alpha ff'f'''\beta t \Rightarrow_{f8} \alpha ff'''\beta t \Rightarrow_{f9} \alpha f\beta t \Rightarrow_{f10} \alpha\beta t
\end{aligned}
$$

For the converse inclusion, proving that no malicious derivations are possible in the obtained SSC ins–del system, we can again follow the lines of reasoning of the previous theorem as far as the context-free simulation rules are concerned. Concerning the simulation of the non-context-free rules, we can follow the path given by the arguments in the proof of Theorem 4, because the rules from Fig. 5a can be seen as a kind of serialization of the rules from Fig. 3. Therefore, let us only sketch the argument for the matching case in what follows. Yet, notice that in particular the *general observation* from that proof is also valid here.

Consider $w = \alpha EABF\beta t$, with $\alpha \in \{EA, EC\}^*, \beta \in \{BF, DF\}^*, t \in T^*$ that is derivable both in the original ssSGNF grammar and in the constructed SSC ins–del system. We cannot apply the wrong deletion simulation,

$$
\begin{aligned}
f1 &= [(\lambda, f, \lambda)_{ins}, \emptyset, N' \cup \mathcal{M}'''] \\
f2 &= [(A, f', \lambda)_{ins}, \emptyset, N' \cup \mathcal{M}''' \setminus \{f\}] \\
f3 &= [(B, f'', \lambda)_{ins}, \emptyset, N' \cup \mathcal{M}''' \setminus \{f, f'\}] \\
f4 &= [(f'', f''', \lambda)_{ins}, \emptyset, N' \cup \mathcal{M}''' \setminus \{f, f'\}] \\
f5 &= [(\lambda, f'', \lambda)_{del}, \{fA, Af', f'B, Bf'', f''f'''\}, \emptyset] \\
f6 &= [(f', B, \lambda)_{del}, \{fA, Af', f'B, Bf'''\}, \emptyset] \\
f7 &= [(f, A, \lambda)_{del}, \{fA, Af', f'f'''\}, \emptyset] \\
f8 &= [(\lambda, f', \lambda)_{del}, \{ff', f'f'''\}, \emptyset] \\
f9 &= [(\lambda, f''', \lambda)_{del}, \{ff'''\}, \emptyset] \\
f10 &= [(\lambda, f, \lambda)_{del}, \emptyset, N' \cup (\mathcal{M}''' \setminus \{f\})]
\end{aligned}
$$

(a) Simulating $f : AB \to \lambda$ by $\mathrm{SSC}_{2,1}\mathrm{ID}(1,1,0;1,1,0)$

$$
\begin{aligned}
f1 &= [(\lambda, f, \lambda)_{ins}, \emptyset, N' \cup \mathcal{M}'''] \\
f2 &= [(A, f', \lambda)_{ins}, \emptyset, N' \cup \mathcal{M}''' \setminus \{f\}] \\
f3 &= [(B, f'', \lambda)_{ins}, \emptyset, N' \cup \mathcal{M}''' \setminus \{f, f'\}] \\
f4 &= [(f'', f''', \lambda)_{ins}, \emptyset, N' \cup \mathcal{M}''' \setminus \{f, f'\}] \\
f5 &= [(\lambda, f'', \lambda)_{del}, \{fA, Af', f'B, Bf'', f''f'''\}, \emptyset] \\
f6 &= [(\lambda, B, f''')_{del}, \{fA, Af', f'B, Bf'''\}, \emptyset] \\
f7 &= [(\lambda, A, f')_{del}, \{fA, Af', f'f'''\}, \emptyset] \\
f8 &= [(\lambda, f', \lambda)_{del}, \{ff', f'f'''\}, \emptyset] \\
f9 &= [(\lambda, f''', \lambda)_{del}, \{ff'''\}, \emptyset] \\
f10 &= [(\lambda, f, \lambda)_{del}, \emptyset, N' \cup (\mathcal{M}''' \setminus \{f\})]
\end{aligned}
$$

(b) Simulating $f : AB \to \lambda$ by $\mathrm{SSC}_{2,1}\mathrm{ID}(1,1,0;1,0,1)$

Fig. 5 Simulating $f : AB \to \lambda$ by $\mathrm{SSC}_{2,1}\mathrm{ID}(1,1,0;1,y,z)$ with $y, z \in \{0,1\}$ and $y + z = 1$

for instance, trying to delete EF, because we again check that the two symbols that are deleted sit side-by-side. Hence, we apply Rules $f1, f2, f3$ and $f4$, in this order, due to the forbidden contexts. Also, the given permitting contexts prevent deletion rules from being applied prematurely. Hence,

$$w \Rightarrow_{f1} w_1 \Rightarrow_{f2} w_2 \Rightarrow_{f3} w_3 \Rightarrow_{f4} w_4$$

is enforced, with $w_4 \in f''' \triangleright f'' \triangleright f' \triangleright f\alpha EABF\beta t$, where on top the insertion contexts guarantee that $\{Af', Bf''f'''\} \subset sub(w_4)$. The permitting contexts of Rules $f6$ through $f9$, as well as the forbidden context of $f10$, now enforce the deletion of f'', i.e., $w_4 \Rightarrow_{f5} w_5$, and the permitting context conditions of Rule $f5$ imply that $w_5 = \alpha EfAf'Bf'''F\beta t$, because the substring AB only appears once in w, and also because there are no substrings like AA or BB in ssSGNF. From now on, the simulation works exactly as analyzed for Fig. 3. This shows that the intended simulation is enforced.

One could build a formal induction argument from this sketch, proving the claimed result. □

Theorem 7 $SSC_{2,1}ID(1,1,0;1,0,1) = SSC_{2,1}ID\ (1,0,1;1,1,0) = RE$.

Proof Clearly, we only have to show that $\mathrm{SSC}_{2,1}\mathrm{ID}(1,1,0;1,0,1) = \mathrm{RE}$, because the other equality follows with Theorem 2. We now sketch how to simulate a type-0 grammar in ssSGNF by an SSC ins–del system of ID size (1, 1, 0; 1, 0, 1) and of degree (2, 1). As in the previous theorem, we re-cycle the simulation from Fig. 4 for the context-free rules. For the non-context-free rules, we will use the simulation rules listed in Fig. 5b. Notice that they are quite similar to the ones from Fig. 5a, with the only difference being that the context conditions are tested on the other side for the deletion rules. Yet, notice that the only purpose of these contexts is to ensure that the intended symbols are deleted (there are possibly many occurrences of A's or B's in the string). This purpose can be fulfilled by using either left or right context.

These explanations should suffice to convince the reader that a formal proof of the claim could be carried out. □

4 Conclusion and future work

In this paper, we introduced the mechanism of simple semi-conditional restrictions on the application of rules of ins–del systems. We described recursively enumerable languages with simple semi-conditional ins–del systems of degrees (2, 1), as shown in Table 1, ignoring symmetric results obtainable from Theorem 2. From the experience of working in this paper, we find that we could put the simulations of the context-free and non context-free rules of SGNF in two different baskets, say A and B respectively. Pick a simulation from basket A and one from basket B. Let the degree and size of the simulation from A be d_1 and s_1 and from B be d_2 and s_2. We may construct a *stitched* simulation of degree $\max(d_1, d_2)$ and of ID size $\max(s_1, s_2)$ to have a simulation of the SGNF rules. This concept of stitching is discussed in detail in Fernau et al. (2017b) in the domain of graph-controlled ins–del systems. The beauty of this method is that, with $k + p$ simulations in baskets A and B as pieces, one could build kp simulations out of it. However, among them, some stitched simulations may end up with not so interesting results in view of the literature. We have already adopted this stitching concept for our simulations in some cases here, but this might be doable in a more systematic way.

We list below what we consider the most challenging problems in this area.

- While Ivanov and Verlan could prove that semi-conditional ins–del systems of degree (2, 2) and ID size (1, 0, 0; 1, 0, 0) are computationally complete, it is open if *simple* semi-conditional ins–del systems of degree (2, 2) and ID size (1, 0, 0; 1, 0, 0) characterize RE.
- The above-mentioned computational completeness result of Ivanov and Verlan is also interesting, because it is one of the few situations where the simulation starts out from a mechanism completely different from type-0 grammars in (some sort of) Geffert normal form. We were not able to mimic this result with simple semi-conditional ins–del systems.
- Again, Ivanov and Verlan could prove that semi-conditional ins–del systems of degree (1, 1) and ID size (2, 0, 0; 1, 1, 0) are computationally complete, but it is unclear whether *simple* semi-conditional ins–del systems of this size and degree characterize RE. In fact, little is known about degree (1, 1).
- With more limited resources, it seems to be difficult if not impossible to characterize RE. In such situations, it would be good to see if we can at least describe all context-free languages or nice sub-classes thereof, as attempted in similar situations in Fernau et al. (2017a); Fernau et al. (2018b, c).

We also pose the following, a more general, open problem for further study in this domain: Given the degree (i, j) satisfying $i, j \geq 1$ and $3 \leq i + j \leq 4$, with what ID sizes does a simple semi-conditional ins–del system characterize RE?

Although the (altogether eight) numbers are the measures traditionally considered in the area of semi-conditional ins–del systems, there are further measures that could be worth considering. For instance, at present the

number of permitting or forbidden strings per rule is unlimited. Yet, this would lead to two further natural descriptional complexity measures, in the spirit of considerations within random context grammars. We refer to the notion of *restricted grammars* studied in Dassow and Masopust (2012); Masopust (2010), where it was shown, among other results, that every recursively enumerable language can be generated by a random context grammar where each rule can test only either for the presence or for the absence of a single symbol. We are not aware of any studies of this restricted form of regulation in combination with ins–del systems.

We also think that considering (simple) semi-conditional ins–del systems that can only (globally) test permitting or forbidden strings has some good motivation, as it is not required that a certain bio-chemical environment should implement both kinds of global tests. We are therefore considering this scenario in forthcoming projects.

Acknowledgements This research would not have been possible without the visits of L. Kuppusamy during Oct. 2017 & June 2018 and I. Raman during May 2018 to the first author at the University of Trier, Germany, subsidized by CIRT, Center for Informatics Research and Technology and some DFG overhead money. We are also grateful to the comments received from the anonymous referees.

References

Dassow J, Masopust T (2012) On restricted context-free grammars. J Comput Syst Sci 78(1):293–304

Fernau H, Kuppusamy L, Raman I (2017a) Graph-controlled insertion-deletion systems generating language classes beyond linearity. In Pighizzini G, Câmpeanu C (eds) Descriptional complexity of formal systems—19th IFIP WG 1.02 international conference, DCFS, volume 10316 of LNCS. Springer, pp 128–139

Fernau H, Kuppusamy L, Raman I (2017b) On the computational completeness of graph-controlled insertion-deletion systems with binary sizes. Theor Comput Sci 682:100–121

Fernau H, Kuppusamy L, Raman I (2018a) Computational completeness of simple semi-conditional insertion-deletion systems. In Stepney S, Verlan S (eds) Unconventional computation and natural computation (UCNC), volume 10867 of *LNCS*. Springer, pp 86–100

Fernau H, Kuppusamy L, Raman I (2018b) Investigations on the power of matrix insertion-deletion systems with small sizes. Nat Comput 17(2):249–269

Fernau H, Kuppusamy L, Raman I (2018c) On describing the regular closure of the linear languages with graph-controlled insertion-deletion systems. RAIRO Inform Théor Appl/Theor Inform Appl 52(1):1–21

Fernau H, Kuppusamy L, Raman I (2019) On path-controlled insertion-deletion systems. Acta Inform 56(1):35–59

Freund R, Kogler M, Rogozhin Y, Verlan S (2010) Graph-controlled insertion-deletion systems. In McQuillan I, Pighizzini G (eds) Proceedings twelfth annual workshop on descriptional complexity of formal systems, DCFS, volume 31 of EPTCS, pp 88–98

Geffert V (1991) Normal forms for phrase-structure grammars. RAIRO Inform Théor Appl/Theor Inform Appl 25:473–498

Ivanov S, Verlan S (2015) Random context and semi-conditional insertion-deletion systems. Fundam Inform 138:127–144

Kari L, Thierrin G (1996) Contextual insertions/deletions and computability. Inf Comput 131(1):47–61

Krassovitskiy A, Rogozhin Y, Verlan S (2011) Computational power of insertion-deletion (P) systems with rules of size two. Nat Comput 10:835–852

Krishna SN, Rama R (2002) Insertion-deletion P systems. In Jonoska N, Seeman NC (eds) DNA computing, 7th international workshop on DNA-based computers, 2001, revised papers, volume 2340 of LNCS. Springer, pp 360–370

Kuppusamy L, Rama R (2003) On the power of tissue P systems with insertion and deletion rules. Pre-Proc. of Workshop on Membrane Computing, volume 28 of Report RGML. Univ. Tarragona, Spain, pp 304–318

Masopust T (2010) Simple restriction in context-free rewriting. J Comput Syst Sci 76(8):837–846

Meduna A, Svec M (2005) Grammars with context conditions and their applications. Wiley, Hoboken

Păun Gh, Rozenberg G, Salomaa A (1998) DNA computing: new computing paradigms. Springer, New York

Takahara A, Yokomori T (2003) On the computational power of insertion-deletion systems. Nat Comput 2(4):321–336

Verlan S (2007) On minimal context-free insertion-deletion systems. J Autom Lang Comb 12(1–2):317–328

Verlan S (2010) Recent developments on insertion-deletion systems. Comput Sci J Moldova 18(2):210–245