# Computational Completeness of Simple Semi-conditional Insertion-Deletion Systems

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

[1] Fachbereich 4 – Abteilung Informatikwissenschaften, CIRT, Universität Trier, 54286 Trier, Germany
fernau@uni-trier.de
[2] School of Computer Science and Engineering, VIT, Vellore 632 014, India
{klakshma,indhumathi.r}@vit.ac.in
[3] School of Information Technology and Engineering, VIT, Vellore 632 014, India

**Abstract.** Insertion-deletion (or ins-del for short) systems are well studied in formal language theory, especially regarding their computational completeness. The need for many variants on ins-del systems was raised by the computational completeness result of ins-del system with (optimal) size $(1, 1, 1; 1, 1, 1)$. Several regulations like graph-control, matrix and semi-conditional have been imposed on ins-del systems. Typically, computational completeness are obtained as trade-off results, reducing the 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. We show that simple semi-conditional ins-del system, with maximum permitting string length 2 and maximum forbidden string length 1 and sizes $(2, 0, 0; 2, 0, 0)$, $(1, 1, 0; 2, 0, 0)$, or $(1, 1, 0; 1, 1, 1)$, are computationally complete. We also describe RE by a simple semi-conditional ins-del system of size $(1, 1, 0; 1, 1, 0)$ and with maximum permitting and forbidden string lengths 3 and 1, respectively. The obtained results complement the existing results available in the literature.

## 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 [7,14,15].

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


S. Stepney and S. Verlan (Eds.): UCNC 2018, LNCS 10867, pp. 86–100, 2018.
https://doi.org/10.1007/978-3-319-92435-9_7

deleted relative to the context conditions. The potential biological meaning of such a rule should be clear. The main research question is under which restrictions can computational completeness results still be obtained. For instance, it is known [13] 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. [14]. 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 [8,9], sometimes in disguise [4], tissue P systems with ins-del rules [10] and semi-conditional ins-del systems [6].

Meduna and Svec have reported on the use of several variants of context conditions in regulated rewriting in the textbook [11]. Here, (simple) semi-conditional rules are of particular importance. In the semi-conditional case, the conditions are sets of words and 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. 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.

Ivanov and Verlan initiated the study of semi-conditional ins-del systems in [6]. 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. We 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 two-sided single-symbol local context) may be deleted. We finally present a trade-off result for systems of degree $(3, 1)$.

## 2 Preliminaries

Let $\mathbb{N}$ denote the set of non-negative integers, and $[1 \dots k] = \{i \in \mathbb{N} \colon 1 \leq i \leq k\}$. If $\Sigma$ is an *alphabet* (finite set), then $\Sigma^*$ denotes the free monoid generated by $\Sigma$. The elements of $\Sigma^*$ are called *strings* or *words*; $\lambda$ denotes the empty string. The morphism from the monoid $\Sigma^*$ to $\mathbb{N}$ (with addition), defined by $a \mapsto 1$ for $a \in \Sigma$ is called *length* of a word; usually, we write $|w|$. $\Sigma^{\leq i}$ collects all words over $\Sigma$ of length at most $i$. 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 use the *shuffle operation* $\sqcup\!\sqcup$ to describe the effect of insertions at a random position in the string. $w^R$ denotes the reversal of $w \in \Sigma^*$. For the computational completeness results, we are using the fact that type-0 grammars in SGNF are known to characterize the class RE of recursively enumerable languages.

**Definition 1** ([5]). *A type-0 grammar $G = (N, T, P, S)$ 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:*

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

The way the normal form is constructed is described in [5]. 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 tool for proving computational completeness results for (regulated) ins-del systems.

**Definition 2** ([7,12]). *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*, $\eta$ is called the *insertion string*, $\delta$ is called the *deletion string* and $x \in A$ is called an *axiom*. If one of the $u$ or $v$ is $\lambda$ for all the insertion (deletion) contexts, then we call the insertion (deletion) *one-sided*. If both $u, v = \lambda$ for every insertion (deletion) rule, then it means that the corresponding insertion (deletion) can be done freely anywhere in the string and is called *context-free* insertion (context-free deletion). 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 1.

**Definition 3** ([6]). *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 $V^*$.*

**Table 1.** Parameters in the size of ins-del system.

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

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 $\Pi$ 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 $\bigcup_{r \in R} \mathcal{F}_r \subseteq V^{\leq j}$. 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 $\Pi$ 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 $\Rightarrow = \bigcup_{r \in R} \Rightarrow_r$. 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 $\mathrm{SSC}_{i,j}\mathrm{ID}(s)$.

*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 $\Pi$ 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 $R$ are given in Table 2.

**Table 2.** SSCID rules describing $\{a^n b^n c^n \mid n \geq 1\}$.

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

We will now explain the working of the rules in Table 2. 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 (immediately) be 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 the system.

$$abc \Rightarrow_{r1} aaAbbc \Rightarrow_{r2} aaAbbBcc \Rightarrow_{r3} aabbBcc \Rightarrow_{r4} aabbcc = a^2b^2c^2.$$

The above process 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. □

The results of this paper and a sketch on how they complement the existing results of [6] are given in Table 3.

**Table 3.** Comparing the results of [6] and this paper.

| S. No | Result of [6] | Complementing result(s) of this paper | Reference |
|---|---|---|---|
| 1 | $\mathrm{SC}_{2,2}\mathrm{ID}(1,0,0;1,0,0) = \mathrm{RE}$ | $\mathrm{SSC}_{2,1}\mathrm{ID}(2,0,0;2,0,0) = \mathrm{RE}$ | Theorem 2 |
| 2 | $\mathrm{SC}_{1,1}\mathrm{ID}(1,1,0;2,0,0) \subsetneq \mathrm{RE}$ | $\mathrm{SSC}_{2,1}\mathrm{ID}(1,1,0;2,0,0) = \mathrm{RE}$ | Theorem 3 |
| 3 | $\mathrm{SC}_{1,1}\mathrm{ID}(1,1,0;1,1,1) \subsetneq \mathrm{RE}$ | $\mathrm{SSC}_{2,1}\mathrm{ID}(1,1,0;1,1,1) = \mathrm{RE}$ | Theorem 4 |
| 4. | $\mathrm{SC}_{1,1}\mathrm{ID}(2,0,0;1,1,0) = \mathrm{RE}$ | $\mathrm{SSC}_{2,1}\mathrm{ID}(2,0,0;2,0,0) = \mathrm{RE}$ | Theorem 2 |
| | | $\mathrm{SSC}_{3,1}\mathrm{ID}(1,1,0;1,1,0) = \mathrm{RE}$ | Theorem 5 |
| | | $\mathrm{SSC}_{3,1}\mathrm{ID}(1,0,1;1,1,0) = \mathrm{RE}$ | Theorem 6 |

## 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 [4].

**Theorem 1.** *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 $s' = (n, i'', i'; m, j'', j')$, and moreover, $\mathrm{SSC}_{i,j}\mathrm{ID}(s) = \mathrm{RE}$ if and only if $\mathrm{SSC}_{i,j}\mathrm{ID}(s') = \mathrm{RE}$.*

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 where the rules are of the type (i) $p : X \to bY$ (ii) $q : X \to Yb$ (iii) $f : AB \to \lambda$ (iv) $g : CD \to \lambda$ and (v)

$h : S' \to \lambda$, where $p, q, f, g, h \in [1 \dots |P|]$ 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. The simulation of type $g : CD \to \lambda$ rules is similar to the simulation of $f$-type rules. Also, we always simulate the $h$ type rule by $[(\lambda, S', \lambda)_{del}, \emptyset, \mathcal{M}]$, 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

$$\begin{aligned} M &= \{m \mid m \in [1 \dots |P|]\}, & M' &= \{m' \mid m \in [1 \dots |P|]\}, \\ M'' &= \{m'' \mid m \in [1 \dots |P|]\}, & M''' &= \{m''' \mid m \in [1 \dots |P|]\}, \\ \mathcal{M}'' &= M \cup M' \cup M'', & \mathcal{M}''' &= M \cup M' \cup M'' \cup M'''. \end{aligned}$$

We first recall from [6] 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. The computational completeness of $\mathrm{SSC}_{0,2}\mathrm{ID}(2,0,0;2,0,0)$ is under study.

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

*Proof.* Consider a type-0 grammar $G = (N, T, P, S)$ in SGNF in which the rules of $P$ are labelled uniquely by numbers $[1 \dots |P|]$. 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 $\Pi$ is $V \subset N \cup T \cup \mathcal{M}'''$. The set of rules of $R$ in $\Pi$ is given as follows. (i) For every rule of type $p : X \to bY$ in $G$, the simulating rules are stated in Fig. 1(a). (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 $f1 = [(\lambda, AB, \lambda)_{del}, \emptyset, \emptyset]$.

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

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

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

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

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

We now proceed to prove that $L(\Pi) = L(G)$. We initially prove that $L(G) \subseteq L(\Gamma)$ by showing that $\Pi$ 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. The application of $p : X \to bY$ to $\alpha X \beta$ derives $\alpha bY\beta = w$, which is correctly simulated by $\Pi$ as follows:

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

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. 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 from using of the same rule again and also indirectly bring the order among the applications of the rules.

We now prove the converse inclusion $L(\Pi) \subseteq L(G)$ by showing that the rules stated in Fig. 1(a) can only be used in the intended way.

Consider a sentential form $w_0 = \alpha X \beta$ derivable in $\Pi$ 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 $r$ ix $X$) 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 $w_1 \in pp' \sqcup\!\sqcup (\alpha X \beta)$, 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 $h1$ 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 $w_1 \Rightarrow_{p2} w_2 = \alpha p \beta$. 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 $w_2' \in Yp''' \sqcup\!\sqcup \alpha p \beta$ 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 $w_3 \in bp'' \sqcup\!\sqcup \alpha p \beta$, 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 $w_3$,[1] $Yp'''$ is randomly inserted, resulting in $w_4 \in Yp''' \sqcup\!\sqcup bp'' \sqcup\!\sqcup \alpha p \beta$, with $Yp''', bp'' \in sub(w_4)$. A careful case analysis reveals that now $p5$ is the only applicable rule.[2] 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 $w_4$. We will see in the next step that only $r = p$ is

[1] Again, any $r4$ could be applied, but we will soon see that $r = p$ is enforced.
[2] Again, any $r5$ could be applied, but we will soon see that $r = p$ is enforced.

possible, which we will therefore use already in the following to avoid clumsy formulations. (b) Previously, we had the choice 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 $G$.

Consider now a sentential form $w_0$ derivable both in $\Pi$ 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. □

Next, we recall from [6] 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. The computational completeness of $\mathrm{SSC}_{0,2}\mathrm{ID}(1,1,0;2,0,0)$ is open for investigation.

**Theorem 3.** $\mathrm{SSC}_{2,1}\mathrm{ID}(1,1,0;2,0,0) = \mathrm{SSC}_{2,1}\mathrm{ID}(1,0,1;2,0,0) = \mathrm{RE}$.

The reader might wonder why we could not deduce this result by sequentializing the construction of Theorem 2 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. 1(a), 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. 1(a) 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 $\Pi$ is $V \subset N \cup T \cup \mathcal{M}''$. The set of rules $R$ of $\Pi$ is given as follows: (i) For every rule of type $p : X \to bY$ in $G$, the simulating rules are stated in Fig. 2(a), (ii) For every rule of type $q : X \to Yb$ in $G$, the simulating rules are stated in Fig. 2(b), (iii) Rules of type $f : AB \to \lambda$ is simulated by the SSCID rules $f1 = [(\lambda, AB, \lambda)_{del}, \emptyset, \emptyset]$.

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

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

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

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

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

We first explain the idea behind the construction of $q$ rule simulation in $\Pi$ 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 main point is to understand the simulation of a context-free rule, say, of type $q$:

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

To show the converse inclusion $L(G) \supseteq L(\Pi)$, consider a string $w_0$ derivable both in $G$ and in $\Pi$. We discuss possible derivations for $w_0$ in $\Pi$ 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 $\Pi$ 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 $w_1 = \alpha X q \beta$. 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 $w_2 \Rightarrow_{q3} w_3 = \alpha q q' \beta$. 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 $\Pi$, then $w_4 \in \{\alpha q q' q'' \beta, \alpha q Y q' \beta\}$. If $w_4 = \alpha q Y q' \beta$, a case analysis reveals that if $w_4 \Rightarrow w_5$ in $\Pi$, then this must be due to applying $q_4$, i.e., $w_5 = \alpha q Y q' q'' \beta$. Now, $q5$ is the only applicable rule, so that $w_6 = \alpha q Y q' b q'' \beta$ is enforced. Alternatively, on $w_4 = \alpha q q' q'' \beta$, 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 $\Pi$, $w_6 = \alpha q Y q' b q'' \beta$ 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 $r8$ 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 $w_7 = \alpha Y q' b q'' \beta$. 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 $w_9 = \alpha Y b \beta$. A case-by-case analysis shows that no other rules are applicable within a derivation $w_7 \Rightarrow w_8 \Rightarrow w_9$ within $\Pi$. This series of rule applications yielding $w_9 = \alpha Y b \beta$ 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 1. □

It is shown in [6] 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 size in the following result is optimal. The computational completeness of $\mathrm{SSC}_{0,2}\mathrm{ID}(1,1,0;1,1,1)$ is under investigation.

**Theorem 4.** $\mathrm{SSC}_{2,1}\mathrm{ID}(1,1,0;1,1,1) = \mathrm{SSC}_{2,1}\mathrm{ID}(1,0,1;1,1,1) = \mathrm{RE}$.

*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 $(1,1,0;1,1,1)$ as follows such that $L(\Pi) = L(G)$. The alphabet of $\Pi$ is $V \subset N \cup T \cup \mathcal{M}''$. The set of rules $R$ of $\Pi$ is given as follows: (i) For every rule of type $p : X \to bY$ in $G$, the simulating rules are stated in Fig. 2(a). (ii) For every rule of type $q : X \to Yb$ in $G$, the simulating rules are stated in Fig. 2(b). (iii) Rules of type $f : AB \to \lambda$ in $G$ are simulated by rules as stated in Fig. 3.

$$
\begin{array}{|l|}
\hline
f1 = [(\lambda, f, \lambda)_{ins}, \emptyset, N' \cup \mathcal{M}''] \\
f2 = [(A, f', \lambda)_{ins}, \emptyset, N' \cup \mathcal{M}'' \setminus \{f\}] \\
f3 = [(B, f'', \lambda)_{ins}, \emptyset, N' \cup \mathcal{M}'' \setminus \{f, f'\}] \\
f4 = [(f, A, f')_{del}, \emptyset, N'] \\
f5 = [(f', B, f'')_{del}, \emptyset, N'] \\
f6 = [(f, f', f'')_{del}, \emptyset, \emptyset] \\
f7 = [(f, f'', \lambda)_{del}, \emptyset, \{f'\}] \\
f8 = [(\lambda, f, \lambda)_{del}, \emptyset, \{f', f''\}] \\
\hline
\end{array}
$$

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

We now proceed to prove that $L(\Pi) = L(G)$. We initially prove that $L(G) \subseteq L(\Pi)$ by showing that $\Pi$ correctly simulates the application of the rules of the types $p, q, f$. The working of the simulation rules for the cases $p$ and $q$ are already explained in Theorem 3. Hence, we now explain only the working of $f$.

The idea behind the construction of $f$ rules is follows. We want to pin $AB$ with the markers and to obtain a substring of the form $fAf'Bf''$. Though $f$ is inserted at random, the correct position of $f$ insertion is taken care with rule $f4$. Rule $f6$ is applicable only when $A$ is deleted, since $f'$ is inserted to the right of $A$ and $f$ cannot be present to the left of $f'$ unless $A$ is deleted. As only one $f'$ is present in between $A$ and $B$ in the string (see rules $f4$ and $f5$) this makes sure that the $A$ and $B$ that are next to each other only gets deleted. Also, as $f4$ and $f5$ have both left and right context for deleting, we cannot delete more than one $A$ and one $B$. To delete $f'$ the presence of $f''$ is required which ensures the presence of $B$. Finally, the markers $f', f''$ and $f$ are deleted. Note that the permitting sets for all the rules in the simulation of $f$ rule are empty.

Simulation of $f : AB \to \lambda$: The rule $f : AB \to \lambda$ of $G$ is simulated by rules of $\Pi$ as stated in Fig. 3 as follows:

$$
\begin{aligned}
&\alpha AB\beta \Rightarrow_{f1} \alpha fAB\beta \Rightarrow_{f2} \alpha fAf'B\beta \Rightarrow_{f3} \alpha fAf'Bf''\beta \Rightarrow_{f4} \\
&\alpha ff'Bf''\beta \Rightarrow_{f5} \alpha ff'f''\beta \Rightarrow_{f6} \alpha ff''\beta \Rightarrow_{f7} \alpha f\beta \Rightarrow_{f8} \alpha\beta.
\end{aligned}
$$

By induction, this shows that $L(G) \subseteq L(\Pi)$.

To show the reverse inclusion $L(G) \supseteq L(\Pi)$, assume that $w_0$ can be derived both in $G$ and in $\Pi$. Hence, $w_0 \in (N'' \cup T)^*(N' \cup \{\lambda\})(N'' \cup T)^*$. If $N' \cap sub(w_0) \neq \emptyset$, from the perspective of $G$, we are still simulating phase I. We have to work through the explanations and case distinctions considered in Theorem 3 once more. A problem could arise if in a sentential form $w_i$ considered in these discussions, $(\mathcal{M}'' \cup N') \cap sub(w_i) = \emptyset$, as then rules like $f1$ become applicable. However, this is never the case, so that there is no danger in starting a simulation of an $f$- or $g$-rule prematurely (i.e., when still simulating phase I).

Hence, $w_0 \in (N'' \cup T)^*$. If $w_0 \in T^*$, nothing remains to be shown. Hence, w.l.o.g., we consider a sentential form $w_0 = \alpha AB\beta$ in $\Pi$ (and in $G$), where $A, B \in N''$ and $\alpha, \beta \in (N'' \cup T)^*$. At first glance, it may seem that we could start the simulation with one of the three rules $f1$ or $f2$ or $f3$. If we apply $f2$ and $f3$ (in this sequence, as first applying $f3$ would block $f2$, and actually any derivation starting with $f3$ on $w_0$ is immediately blocked), then the only applicable rule is $f5$ which will delete $B$ between $f'$ and $f''$, yielding $f'f''$ as a substring of some $w'''$, with $w_0 \Rightarrow_{f2} w' \Rightarrow_{f3} w'' \Rightarrow_{f5} w'''$. Alternatively, this process yielding $w'''$ can be described by applying the rewriting rule $B \to f'f''$ to $w_0$. The marker $f$ has neither been introduced earlier nor could be inserted later, because its insertion rule $f1$ demands absence of $f', f''$ in particular. But, in the absence of $f$, it is impossible to delete the markers $f', f''$ using the rules $f6$ and $f7$, respectively.

Hence, in order to make a productive move, we have to begin by applying rule $f1$ to $w_0 = \alpha AB\beta$, which randomly inserts the marker $f$. So, if $w_0 \Rightarrow_{f1} w_1$, then $w_1 \in f \sqcup\!\sqcup (\alpha AB\beta)$. Notice that $f1$ cannot be applied again on $w_1$, nor can $g1$ be, as these rules require all rule marker symbols to be absent. This kind of reasoning reminds valid for the whole derivation that we are going to discuss, disabling unwanted premature starts of other simulations throughout. The only rules that are applicable on $w_1$ are $f2$, $f3$, or $f8$. As applying $f8$ simply deletes the $f$ marker introduced in the previous derivation step, this gives no overall progress, so that we can ignore this as an unnecessary detour of the derivation process. Now we apply rules $f2$ and $f3$ to $w_1$ in order, as applying $f3$ first would lead to a blockage of the derivation. We remark here that it is possible that on applying $f2$, the marker $f'$ may be placed after any occurrence of $A$ in $w_1$. Similar is the case with the application of rule $f3$ with respect to $B$. Hence in general, if $w_0 \Rightarrow_{f1} w_1 \Rightarrow_{f2} w_2 \Rightarrow_{f3} w_3$, then $w_3 \in f \sqcup\!\sqcup f' \sqcup\!\sqcup f'' \sqcup\!\sqcup w_0$ with $Af', Bf'' \in sub(w_3)$. By the forbidden context conditions, none of the insertion rules are applicable to $w_3$. In order to apply $f4$, $fAf' \in sub(w_3)$ is necessary, and in order to apply $f5$, $f'Bf'' \in sub(w_3)$. The only way to get rid of the introduced markers again is to apply $f6$, $f7$ and $f8$ (in this order). But before being able to apply $f6$, the substrings $fAf'$ and $f'Bf''$ of $w_3$ have to be transformed to $ff'$ and $f'f''$, respectively, so that $f4$ and $f5$ have to be applied in any order. Hence, we find $w_3 \Rightarrow w_4 \Rightarrow w_5$, with $w_5$ could have been alternatively derived from $w_0$ by applying the rewriting rule $AB \to ff'f''$. Hence, $w_5 = \alpha ff'f''\beta$, because $w_0 = \alpha AB\beta$ was also derivable in $G$, and any such string contains the substring $AB$ only in one place. It is not hard to see that $f6$ is the only applicable rule

now. Application of the rules $f6, f7, f8$ in a deterministic manner (i.e., each time there is no other rule that applies, and there is only one location in the current string that may be transformed) finally yields $w_8 = \alpha\beta$. This series of rule applications, yielding $w_8$ from $w_0 = \alpha AB\beta$, corresponds to applying the rewriting rule $AB \to \lambda$ of $G$. By induction, the claim $L(\Pi) \subseteq L(G)$ follows.

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

In the previous theorem, the insertion had one-sided context and deletion had both the left and right contexts. In this case computational completeness was achieved with degree $(2,1)$. If we further wish to have one-sided context for deletion as well, then computational completeness is achieved with increasing the degree to $(3,1)$. These are the first RE results ever for degree $(3,1)$.

**Theorem 5.** $\mathrm{SSC}_{3,1}\mathrm{ID}(1,1,0;1,1,0) = \mathrm{SSC}_{3,1}\mathrm{ID}(1,0,1;1,0,1) = \mathrm{RE}$.

*Proof.* The proof is very similar to the previous one. We will first show that $\mathrm{SSC}_{3,1}\mathrm{ID}(1,1,0;1,1,0) = \mathrm{RE}$. The second part then follows from Theorem 1.

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 $(3,1)$ and ID size $(1,1,0;1,1,0)$ as follows such that $L(\Pi) = L(G)$. The alphabet of $\Pi$ is $V \subset N \cup T \cup \mathcal{M}''$. The set of rules $R$ of $\Pi$ is given as follows: (i) For every rule of type $p : X \to bY$ in $G$, the simulating rules are stated in Fig. 2(a). (ii) For every rule of type $q : X \to Yb$ in $G$, the simulating rules are stated in Fig. 2(b). (iii) Rules of type $f : AB \to \lambda$ in $G$ are simulated as stated in Fig. 4(a). The idea behind the construction of $f$ rules is very similar to the working of the rules in Fig. 3 and hence omitted. However we now highlight the difference in the two simulations (stated in Figs. 3 and 4(a)). Rules $f1, f2, f3, f7, f8$ in both the simulations are the same. If rules $f4, f5, f6$ of the former simulation deletes a symbol say $\alpha$ between the contexts $c_1$ and $c_2$ using the deletion rule $(c_1, \alpha, c_2)_{del}$, then the same is taken care by the rules $f4, f5, f6$ (respectively) of the latter simulation by their permitting string $c_1\alpha c_2$. Some formal arguments are presented below.

Simulation of $f : AB \to \lambda$: The intended derivation is the same as the one given in Theorem 4. This already shows that $L(G) \subseteq L(\Pi)$ by induction.

To show the reverse inclusion, we consider a sentential form $w_0 = \alpha AB\beta$ in $\Pi$, where $A, B \in N''$ and $\alpha, \beta \in (N'' \cup T)^*$. As in the proof of Theorem 4, we end up applying $f1, f2, f3$, in this order, to arrive at $w_3 \in f \sqcup\!\sqcup f' \sqcup\!\sqcup f'' \sqcup\!\sqcup w_0$ with $Af', Bf'' \in sub(w_3)$. The only way to continue is to apply rules $f4$ or $f5$ (in any order), with $f4$ guaranteeing that $fAf' \in sub(w_3)$ and with $f5$ guaranteeing that $f'Bf'' \in sub(w_3)$. Altogether, if $f4$ and $f5$ could have been applied, then $w_3 = \alpha fAf'Bf''\beta$, as there is only one position in $w_0$ where the substring $AB$ could occur. Now, $w_3 \Rightarrow_{f4} w_4 \Rightarrow_{f5} w_5 = \alpha ff'f''\beta$, and the same result is obtained when first applying $f5$ and then $f4$. A simple case analysis shows that only $f6$ is applicable now, yielding $w_6 = \alpha ff''\beta$. From this point on, the argument continues again as in Theorem 3. □

**Theorem 6.** $\mathrm{SSC}_{3,1}\mathrm{ID}(1,1,0;1,0,1) = \mathrm{SSC}_{3,1}\mathrm{ID}(1,0,1;1,1,0) = \mathrm{RE}$.

*Proof.* Along with the simulations presented in Figs. 2(a) and (b), we present a simulation of $f$ rule in Fig. 4(b) which is a reflection of the simulation stated in Fig. 4(a) in order to prove that $\mathrm{SSC}_{3,1}\mathrm{ID}(1,1,0;1,0,1) = \mathrm{RE}$ and hence we are not giving a formal proof. The claim $\mathrm{SSC}_{3,1}\mathrm{ID}(1,0,1;1,1,0) = \mathrm{RE}$ again follows with Theorem 1. □

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

(a) $\mathrm{SSC}_{3,1}\mathrm{ID}(1,1,0;1,1,0)$

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

(b) $\mathrm{SSC}_{3,1}\mathrm{ID}(1,1,0;1,0,1)$

**Fig. 4.** Simulation of the rule $f : AB \to \lambda$

## 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 $(2,1)$ and $(3,1)$, as shown in Table 3, ignoring symmetric results obtainable from Theorem 1. We list below some 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.
- 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 even with degree $(2,1)$, it is unclear whether *simple* semi-conditional ins-del systems of this size characterize RE.
- 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 [1–3].

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

## References


1. Fernau, H., Kuppusamy, L., Raman, I.: Graph-controlled insertion-deletion systems generating language classes beyond linearity. In: Pighizzini, G., Câmpeanu, C. (eds.) DCFS 2017. LNCS, vol. 10316, pp. 128–139. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-60252-3_10
2. Fernau, H., Kuppusamy, L., Raman, I.: Investigations on the power of matrix insertion-deletion systems with small sizes. Accepted with Natural Computing (2017)
3. Fernau, H., Kuppusamy, L., Raman, I.: On describing the regular closure of the linear languages with graph-controlled insertion-deletion systems. In: RAIRO Informatique théorique et Applications/Theoretical Informatics and Applications (2017, Submitted)
4. Fernau, H., Kuppusamy, L., Raman, I.: On path-controlled insertion-deletion systems. Accepted with Acta Informatica (2017)
5. Freund, R., Kogler, M., Rogozhin, Yu., Verlan, S.: Graph-controlled insertion-deletion systems. In: McQuillan, I., Pighizzini, G., (eds.) Proceedings Twelfth Annual Workshop on Descriptional Complexity of Formal Systems, DCFS, vol. 31. EPTCS, pp. 88–98 (2010)
6. Ivanov, S., Verlan, S.: Random context and semi-conditional insertion-deletion systems. Fundamenta Informaticae **138**, 127–144 (2015)
7. Kari, L., Thierrin, G.: Contextual insertions/deletions and computability. Inf. Comput. **131**(1), 47–61 (1996)
8. Krassovitskiy, A., Rogozhin, Yu., Verlan, S.: Computational power of insertion-deletion (P) systems with rules of size two. Nat. Comput. **10**, 835–852 (2011)
9. Krishna, S.N., Rama, R.: Insertion-deletion P systems. In: Jonoska, N., Seeman, N.C. (eds.) DNA 2001. LNCS, vol. 2340, pp. 360–370. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-48017-X_34
10. Kuppusamy, L., Rama, R.: On the power of tissue P systems with insertion and deletion rules. In: Pre-Proceedings of Workshop on Membrane Computing, vol. 28. Report RGML, pp. 304–318. University of Tarragona, Spain (2003)
11. Meduna, A., Svec, M.: Grammars with Context Conditions and Their Applications. Wiley-Interscience, New York (2005)
12. Păun, Gh., Rozenberg, G., Salomaa, A.: DNA Computing: New Computing Paradigms. Springer, Heidelberg (1998). https://doi.org/10.1007/978-3-662-03563-4
13. Takahara, A., Yokomori, T.: On the computational power of insertion-deletion systems. Nat. Comput. **2**(4), 321–336 (2003)
14. Verlan, S.: On minimal context-free insertion-deletion systems. J. Automata Lang. Comb. **12**(1–2), 317–328 (2007)
15. Verlan, S.: Recent developments on insertion-deletion systems. Comput. Sci. J. Moldova **18**(2), 210–245 (2010)