Natural Computing (2018) 17:249–269
https://doi.org/10.1007/s11047-017-9656-8

Investigations on the power of matrix insertion-deletion systems with small sizes

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

Published online: 5 December 2017

Abstract
Matrix insertion-deletion systems combine the idea of matrix control (a control mechanism well established in regulated rewriting) with that of insertion and deletion (as opposed to replacements). Given a matrix insertion-deletion system, the size of such a system is given by a septuple of integers $(k; n, i', i''; m, j', j'')$. The first integer k denotes the maximum number of rules in (length of) any matrix. The next three parameters n, i', i'' denote the maximal length of the insertion string, the maximal length of the left context, and the maximal length of the right context of insertion rules, respectively. The last three parameters m, j', j'' are similarly understood for deletion rules. In this paper, we improve on and complement previous computational completeness results for such systems, showing that matrix insertion-deletion systems of size (1) (3; 1, 0, 1; 1, 0, 1), (3; 1, 0, 1; 1, 1, 0), (3; 1, 1, 1; 1, 0, 0) and (3; 1, 0, 0; 1, 1, 1) (2) (2; 1, 0, 1; 2, 0, 0), (2; 2, 0, 0; 1, 0, 1), (2; 1, 1, 1; 1, 1, 0) and (2; 1, 1, 0; 1, 1, 1), are computationally complete. Further, we also discuss linear and metalinear languages and we show how to simulate grammars characterizing them by matrix insertion-deletion systems of size (3; 1, 1, 0; 1, 0, 0), (3; 1, 0, 1; 1, 0, 0), (2; 2, 1, 0; 1, 0, 0) and (2; 2, 0, 1; 1, 0, 0). We also generate non-semilinear languages using matrices of length three with context-free insertion and deletion rules.

Keywords Matrix ins-del systems · Matrix control · Descriptional complexity · Computational completeness · (Meta)linear languages

1 Introduction

Inserting or deleting words in between parts of sentences often take place when processing natural languages; such insertions and deletions are usually based on context information. To some surprise, this is also happening in biology, especially, in DNA processing and in RNA editing (see Benne 1993; Biegler et al. 2007; Păun et al. 1998). Based on the insertion operation, Marcus (1969) introduced *external contextual grammars* as an attempt to mathematically model natural language phenomena. A different variety of linguistically motivated contextual grammars are the *semi-contextual grammars* studied by Galiukschov (1981), which can be also viewed as insertion grammars. The deletion operation as a basis of a grammatical derivation process was introduced in Kari (1991), where the deletion was motivated as a variant of the right-quotient operation that does not necessarily happen at the right end of the string. Insertion and deletion together were first studied in Kari and Thierrin (1996). The corresponding grammatical mechanism is called *insertion-deletion system* (abbreviated as ins-del system). Informally, the insertion and deletion operations of an ins-del system are defined as follows: if a string η is inserted between two parts w_1 and w_2 of a string w_1w_2 to get $w_1\eta w_2$, we call the operation *insertion*, whereas if a substring δ is deleted from a string

A preliminary version of this paper appeared in Proceedings of UCNC 2016, LNCS 9726, pp. 35–48, 2016.

✉ Henning Fernau
fernau@uni-trier.de

Lakshmanan Kuppusamy
klakshma@vit.ac.in

Indhumathi Raman
indhumathi.r@vit.ac.in

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

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

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

$w_1 \delta w_2$ to get $w_1 w_2$, we call the operation *deletion*. Suffixes of w_1 and prefixes of w_2 are called *contexts*.

Several variants of ins-del systems have been considered in the literature and among them the important variants (from our perspective) are *ins-del P systems* (Alhazov et al. 2011), tissue P systems with ins-del rules (Kuppusamy and Rama 2003), *context-free ins-del systems* (Margenstern et al. 2005), *graph-controlled ins-del systems* (Fernau et al. 2017a; Freund et al. 2010; Ivanov and Verlan 2017), *matrix insertion systems* (Marcus and Păun 1990), *matrix ins-del systems* (Kuppusamy et al. 2011; Petre and Verlan 2012; Kuppusamy and Mahendran 2016), *random context and semi-conditional ins-del systems* (Ivanov and Verlan 2015), etc. We refer to the survey (Verlan 2010) for more details of several variants of ins-del systems.

Most variants of ins-del systems that we mentioned, even ins-del systems themselves, are known to be computationally complete, i.e., they characterize the family RE of recursively enumerable languages. In such circumstances, in the area of descriptional complexity of formal languages, one aims at investigating which of the resources are really needed to obtain computational completeness. For instance, is it really necessary to permit insertion operations that check out contexts of arbitrary length? Such considerations are also well-motivated from biology, assuming now that an insertion operation should model what is happening in RNA editing (Benne 1993. Clearly, it would be even dangerous for an organism if such operations would be based on checking (parts of) molecules of arbitrary length, as such an operation would inevitably introduce failures. We would also like to mention that from the very beginning of Theoretical Computer Science, there has been a certain interest in designing Turing machines that use little resources, yet achieving computational completeness. For instance, Shannon showed back in 1956 that two internal states are sufficient for Turing machines to remain computationally complete (see Shannon 1956). This result was complemented by Herman (1968) who proved that one-state machines are not computationally complete, even on a multi-dimensional tape. Such results inspired many further studies, for instance on deterministic Turing machines with m tape symbols and n states, see Kudlek (1996) for a 20-year old survey. A more recent overview can be found in Neary and Woods (2012). Even more, the origins of two now well-established conference series in Formal Languages, namely that of MCU and that of DCFS can be traced back to this type of considerations. On the other side, for resource restrictions that do not suffice to achieve computational completeness, one is interested in seeing which known families of languages can be still modeled; we refer to Neary (2017) as a very recent paper on 2-symbol 2-state Turing machines.

Similar considerations have been undertaken for grammatical mechanisms, as well. In that context, restricted models are often known as *normal forms*. Such normal form results are often valuable as a starting point to obtain new descriptional complexity results for other devices. For instance, considering type-0 grammars, important normal form results have been obtained by Kuroda (1964) and Penttonen (1974) and most notably by Geffert (1991a, b). In fact, these normal form have been the basis for obtaining most descriptional complexity results in the area of regulated rewriting, also confer the textbook (Dassow and Păun 1989). This paper follows this tradition. However, notice that each normal form has its own advantages and disadvantages if it comes to the task of using it as a basis of computational completeness results. Therefore, we found it quite surprising that we managed to improve on most existing results by making use only of one particular variant called *Special Geffert Normal Form* (Freund et al. 2010), or SGNF for short, as formally introduced in the next section.

We are now discussing the main topic of our study. In a matrix ins-del system, the insertion-deletion rules are given in matrix form. If a matrix is chosen for derivation, then all the rules in that matrix are applied in order and no rule of the matrix is exempted. In the size $(k; n, i', i''; m, j', j'')$ of a matrix insertion-deletion system, the parameters (from left to right) denote the maximum number of rules in any matrix (also known as the *length* of the matrix), the maximal length of the inserted string, the maximal length of the left context for insertion, the maximal length of the right context for insertion, the maximal length of the deleted string, the maximal length of the left context for deletion, maximal length of the right context for deletion. We denote the language classes generated by matrix ins-del systems of size s by $\mathrm{MAT}(s)$. It is shown in Petre and Verlan (2012) that the following matrix ins-del systems are computationally complete; we provide precise references for each case.

- $\mathrm{MAT}(3; 1, 1, 0; 1, 1, 0)$ (Theorem 2),
- $\mathrm{MAT}(3; 1, 1, 0; 1, 0, 1)$ (Theorem 3),
- $\mathrm{MAT}(2; 1, 1, 0; 2, 0, 0)$ (Theorem 9),
- $\mathrm{MAT}(2; 2, 0, 0; 1, 1, 0)$ (Theorem 8),
- $\mathrm{MAT}(8; 1, 1, 1; 1, 0, 0)$ (Theorem 5) †,
- $\mathrm{MAT}(8; 1, 0, 0; 1, 1, 1)$ (Theorem 6) †.

Notice that all results listed above were also proved using SGNF, except for the ones marked by †. In our paper, we prove that the following classes of languages are also computationally complete:

- $\mathrm{MAT}(2; 1, 0, 1; 2, 0, 0)$ (Theorem 2),

- $\mathrm{MAT}(2;2,0,0;1,0,1)$ (Theorem 2),
- $\mathrm{MAT}(3;1,0,1;1,0,1)$ (Theorem 2),
- $\mathrm{MAT}(3;1,0,1;1,1,0)$ (Theorem 2),
- $\mathrm{MAT}(2;1,1,0;1,1,1)$ (Theorem 5),
- $\mathrm{MAT}(2;1,1,1;1,1,0)$ (Theorem 6),
- $\mathrm{MAT}(3;1,1,1;1,0,0)$ (Theorem 4),
- $\mathrm{MAT}(3;1,0,0;1,1,1)$ (Theorem 3).

The first four results can be seen as relatively easy consequences of results published in Petre and Verlan (2012), listed here mainly to complete the picture concerning descriptional complexity aspects of matrix ins-del systems. The last four results constitute the main results of this paper. The last two results improve on the results shown in Petre and Verlan (2012) by large, reducing the lengths of the matrices from eight to three. This also shows again the power of SGNF, as the previous results were the only (previous) ones that have been derived in a different fashion. In the conference version Fernau et al. (2016), a computational completeness result for matrix ins-del system with size (2; 1, 1, 1; 1, 0, 0) was claimed whose underlying construction turned out to be dubious and hence is replaced by the result on $\mathrm{MAT}(3;1,1,1;1,0,0)$. Notice that the construction from Fernau et al. (2016) was based on Penttonen normal form, while the new construction in the present paper uses SGNF. We like to emphasize that the result concerning $\mathrm{MAT}(2;1,1,0;1,1,1)$ is not improvable in the sense that $\mathrm{MAT}(1;1,1,0;1,1,1)$ is known not to coincide with RE (see Krassovitskiy et al. 2008). However, it remains open if, say, $\mathrm{MAT}(2;1,1,0;1,1,0)$ is computationally complete. All these results are proved in Sect. 4.

Further, we consider two smaller families of (context-free) languages—namely, the *linear* and the *metalinear* languages, where the latter are obtained as unions of concatenations of an arbitrary number of linear languages. We show that the following matrix ins-del systems all strictly contain the families of linear (LIN) and metalinear (MLIN) languages:

- $\mathrm{MAT}(2;2,1,0;1,0,0)$,
- $\mathrm{MAT}(2;2,0,1;1,0,0)$,
- $\mathrm{MAT}(3;1,1,0;1,0,0)$, and
- $\mathrm{MAT}(3;1,0,1;1,0,0)$.

The reader can find proofs and detailed explanations of these results in Sect. 5. Notice that the idea of simulating linear and metalinear grammars with very small sizes of matrix ins-del systems is also a novel direction of research within matrix ins-del systems. This is a very sensible idea whenever it is unknown if the size under consideration allows for a computational completeness result or not. In the cases that we considered, it is even unknown if or how all context-free languages can be achieved. Similar considerations have been undertaken before in the more classical areas of descriptional complexity issues within regulated rewriting (see Dassow and Păun 1985; Păun 1984). Observe that the sizes of the simulating systems do not increase when we generalize the simulation from linear to metalinear. In this respect, matrix control differs from graph control in connection with ins-del systems (see Fernau et al. (2017a, b).

We also like to point the reader to Example 2 (which again complements the conference version). This exhibits that the well-known non-regular language $\{a^n b^n \mid n \geq 1\}$ can be generated by a matrix ins-del system of size (2; 1, 0, 0; 1, 1, 1) in a possibly surprisingly non-trivial fashion.

We also briefly discuss the semilinearity of Parikh images of matrix ins-del languages. Our findings are collected in Sect. 7. In particular, it is shown that already $\mathrm{MAT}(3;1,0,0;1,0,0)$ contains non-semi-linear languages. Also, consequences to computational linguistics are discussed. These considerations have been continued and extended in Fernau and Kuppusamy (2017).

We conclude our paper by presenting several concrete directions of future research in this area of Formal Languages. Further research should be also inspired by the tables that summarize all that is known about matrix ins-del systems concerning computational completeness aspects, as collected in Sect. 6.

2 Preliminaries

We assume that the readers are familiar with the standard notations used in formal language theory. However, we now recall a few notations here that are important for the understanding of the paper.

Let $\mathbb{N}$ denote the set of positive integers, and Σ^* denote the free monoid generated by the *alphabet* (finite set) Σ. The elements of Σ^* are called *strings* or *words*; λ denotes the empty string. For a string $w \in \Sigma^*$, $|w|$ denotes the length of a string w and w^R denotes the *reversal* (also called *mirror image*) of w. Likewise, L^R and $\mathcal{L}^R$ are understood for languages L and language families $\mathcal{L}$. The Kleene closure of a language L is denoted as L^*. The family of recursively enumerable languages, linear and metalinear languages are denoted by RE, LIN and MLIN respectively. We will provide more details on metalinear languages in Sect. 2.3. Occasionally, we use the shuffle operator, written as ш.

For the computational completeness results, we are using the fact that type-0 grammars in the so-called special Geffert normal form are known to characterize the recursively enumerable languages. Consider a type-0 grammar $G=(N,T,S,P)$, where (as usual), N is the nonterminal alphabet, T is the terminal alphabet, $S \in N$ is the start symbol and P is the set of production rules. According to Freund et al. (2010), G is said to be in *special Geffert normal form*, SGNF for short, if

- N decomposes as $N = N' \cup N''$, where $N'' = \{A, B, C, D\}$ and N' contains at least the two nonterminals S and S',
- the only non-context-free rules in P are the two erasing rules $AB \to \lambda$ and $CD \to \lambda$,
- the context-free rules are of the following forms: $X \to Yb$ or $X \to bY$ where $X, Y \in N', X \neq Y$, $b \in T \cup N''$, or $S' \to \lambda$.

How to construct this normal form is described in Freund et al. (2010). This construction is based on the one in Geffert's paper (1991a). From this construction, it is clear that the derivation of a string is performed in two phases. First, the context-free rules are applied repeatedly and the phase I is completed by applying the rule $S' \to \lambda$ in the derivation. In phase II, only the non-context-free erasing rules are applied repeatedly and the derivation ends. Notice that as these context-free rules are more of a linear type, it is easy to see that there can be at most only one nonterminal from N' present in the derivation of G. We exploit this observation in our proofs. Also, note that $X \neq Y$ for $X, Y \in N'$ in the context-free rules.

We will often use labels from $[1 \dots |P|]$ to uniquely address the rules of a grammar in SGNF. Then, such labels (and possibly also primed version thereof) will be used as *rule markers* that are therefore part of the nonterminal alphabet of the simulating matrix ins-del system. A bit surprisingly, context-free rules appear to be harder to simulate by ins-del systems compared to the non-context-free rules. For the ease of reference, we collect in P_{ll} the labels of the context-free rules of the form $X \to Yb$ (which resemble left-linear rules) and in P_{rl} the labels of the context-free rules of the form $X \to bY$ (which resemble right-linear rules).

2.1 Insertion-deletion systems

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

Definition 1 An *insertion-deletion system* is a construct $\gamma = (V, T, A, R)$, where V is an alphabet, $T \subseteq V$ is the terminal alphabet, A is a finite language over V, R is a finite set of triplets of the form $(u, \eta, v)_{ins}$ or $(u, \delta, v)_{del}$, where $(u, v) \in V^* \times V^*$, $\eta, \delta \in V^+$.

The pair $(u,\ v)$ is called the *context*, η is called the *insertion string*, δ is called the *deletion string* and $x \in A$ is called an *axiom*. For all contexts of t where $t \in \{ins, del\}$, if $u = \lambda$ ($v = \lambda$), then we call the operation t to be right context (left context). If $u = v = \lambda$ for a rule, then the corresponding insertion/deletion can be done freely anywhere in the string and is called context-free insertion/deletion. An insertion rule will be of the form $(u, \eta, v)_{ins}$, which means that the string η is inserted between u and v. A deletion rule will be of the form $(u, \delta, v)_{del}$, which means that the string δ is deleted between u and v. Applying $(u, \eta, v)_{ins}$ corresponds to the rewriting rule $uv \to u\eta v$, and $(u, \delta, v)_{del}$ corresponds to the rewriting rule $u\delta v \to uv$.

Consequently, for $x, y \in V^*$ we write $x \Rightarrow y$ if y can be obtained from x by using either an insertion rule or a deletion rule which formally means the following:

1. $x = x_1uvx_2, y = x_1u\eta vx_2$, for some $x_1, x_2 \in V^*$ and $(u, \eta, v)_{ins} \in R$.
2. $x = x_1u\delta vx_2, y = x_1uvx_2$, for some $x_1, x_2 \in V^*$ and $(u, \delta, v)_{del} \in R$.

The language generated by γ is defined by

$$L(\gamma) = \{w \in T^* \mid x \Rightarrow^* w, \text{forsome} x \in A\},$$

where $\Rightarrow^*$ is the reflexive and transitive closure of the relation $\Rightarrow$.

Observe that, if $u = v = \lambda$ for an insertion rule $(u, \eta, v)_{ins}$, then $x \Rightarrow y$ (by applying this rule) implies that $y \in x ш \eta$. In other words, η can now be inserted arbitrarily into x.

2.2 Matrix insertion-deletion systems

In this subsection, we describe the *matrix insertion-deletion systems* as in Kuppusamy et al. (2011) and Petre and Verlan (2012).

Definition 2 A matrix insertion-deletion system is a construct $\Gamma = (V, T, A, R)$ where V is an alphabet, $T \subseteq V$, A is a finite language over V, R is a finite set of matrices $\{r_1, r_2, \dots r_l\}$, where each r_i, $1 \le i \le l$, is a matrix of the form $r_i = [(u_1, \alpha_1, v_1)_{t_1}, (u_2, \alpha_2, v_2)_{t_2}, \dots, (u_k, \alpha_k, v_k)_{t_k}]$ with $t_j \in \{ins, del\}$, $1 \le j \le k$.

For $1 \le j \le k$, the triple $(u_j, \alpha_j, v_j)_{t_j}$ is an ins-del rule. Consequently, for $x, y \in V^*$ we write $x \Rightarrow x' \Rightarrow x'' \Rightarrow \dots \Rightarrow y$, if y can be obtained from x by applying all the rules of a matrix r_i, $1 \le i \le l$, in order; in this case, we write $x \Longrightarrow_{r_i} y$.

At this point, we make a note that in a derivation, the rules of a matrix are applied sequentially one after another in the given order and no rule is used in *appearance* *checking*, as it is often the case in more classical matrix grammars with rewriting rules (see Dassow and Păun 1989). By $w \Longrightarrow_* z$, we denote the relation $w \Longrightarrow_{r_{i_1}} w_1 \Longrightarrow_{r_{i_2}} \ldots \Longrightarrow_{r_{i_k}} z$, where for all $j, 1 \le j \le k$, we have $1 \le i_j \le l$. This notation should help prevent confusion with applications of single ins-del rules. The language $L(\Gamma)$ generated by Γ is defined as follows.

$$L(\Gamma) = \{w \in T^* \mid x \Longrightarrow_* w, \mathrm{for some} x \in A\}.$$

If a matrix ins-del system has at most k rules in a matrix and the size of the underlying ins-del system is $(n, i', i''; m, j', j'')$, then we denote the corresponding class of language by $\mathrm{MAT}(k; n, i', i''; m, j', j'')$.

We now discuss a few examples of matrix ins-del systems. These are used later in proving some theorems. More specifically, the languages used in Examples 1, 3, and 4 are used as witness languages in the hierarchy of families of languages between the linear languages and even beyond the context-free languages (see Fig. 1). The idea used in constructing the matrix rules of Example 2 is used later in the proofs of Theorems 5 and 6.

Example 1 The language $L_1 = \{a^n b^m c^n d^m \mid m, n \ge 1\}$ of cross-serial dependencies can be generated by a binary matrix insertion-deletion system as follows: $\Gamma_1 = (\{a, b, c, d\}, \{a, b, c, d\}, \{abcd\}, R)$, where $R = \{m1, m2\}$ consists of two matrices:

$$m1 = [(a, a, \lambda)_{ins}, (c, c, \lambda)_{ins}] \qquad m2 = [(b, b, \lambda)_{ins}, (d, d, \lambda)_{ins}]$$

We note that the rules $m1' = [(\lambda, a, a)_{ins}, (\lambda, c, c)_{ins}]$, $m2' = [(\lambda, b, b)_{ins}, (\lambda, d, d)_{ins}]$ also generate L_1. This shows that

$$L_1 \in \mathrm{MAT}(2; 1, 1, 0; 0, 0, 0) \cap \mathrm{MAT}(2; 1, 0, 1; 0, 0, 0)\,.$$

We refer to Stabler (2004) for further variants and a discussion of the linguistic relevance of this type of example.

In the next example, we generate a linear language that is not that easily generated with binary matrices having context-free insertion rules; rather, this requires a clever manipulation in constructing the matrix rules (and axiom too).

Example 2 The following non-regular language $L_2 = \{a^n b^n \mid n \ge 1\}$ can be generated by a matrix ins-del system with size (2; 1, 0, 0; 1, 1, 1).

Consider the matrix ins-del system $\Gamma_2 = (\{X, Y, a, b, \#, \$, \dagger\}, \{a, b\}, \{\#\$\}, R)$, where the rule matrices, collected in R, are given as below.'

$$\begin{aligned} m1 &= [(\lambda, X, \lambda)_{ins}, (\lambda, \dagger, \lambda)_{ins}] \\ m3 &= [(\#, \$, X)_{del}] \\ m5 &= [(\lambda, a, \lambda)_{ins}, (a, X, a)_{del}] \\ m7 &= [(\lambda, b, \lambda)_{ins}, (b, Y, b)_{del}] \end{aligned}$$

Let us explain how and why this grammar Γ_2 works. The derivation has to start with the axiom #\$. One has to begin with applying $m1$, as it is easy to check that no other matrix can be applied. This will introduce an X and a † in the string, which must be now from $\#\$ ш X ш \dagger$. Now, only $m1$ and $m2$ are applicable.

To delete an occurrence of the non-terminal †, the matrix $m2$ has to be applied, which will introduce a Y and ensures that the symbol † introduced by $m1$ was actually placed between # and \$ and is now deleted. So, the resulting string is from $Y ш X ш \#\$$, although not all such strings are possible, as no X or Y occurs in between # and \$. More formally, this can be described by using a new letter c and a homomorphism $h : \{X, Y, a, b, c, \dagger\}^* \to \{X, Y, a, b, \#, \$, \dagger\}^*$, given by $c \mapsto \#\$$ and $x \mapsto x$ for $x \in \{X, Y, a, b, \dagger\}$. Namely, the set of derivable words (as explained above) is then $h(X ш Y ш c)$.

Also, this checks that $m1$ was not repeatedly applied, because there is always (at most) only one occurrence of # and one occurrence of \$, which means that other occurrences of † have been inserted elsewhere in the string, where they cannot be deleted anymore.

By a similar argument, we can rule out applying first $m1$ and then $m3$ (which might be enabled if † was not placed in between # and \$); namely, again there is no way to delete the † symbol introduced when applying $m1$ at any time in the future.

So, we have to apply $m1$ and $m2$ in this sequence, and applying $m1$ and then $m2$ produces one occurrence of X and one occurrence of Y in the string. If $m1$ and $m2$ are applied alternatingly for n times, we will get n occurrences of X and n occurrences of Y in the string. More formally, reconsidering the homomorphism h defined above, we obtain at this stage any word from $h(X^n ш Y^n ш c)$, for $n \ge 1$.

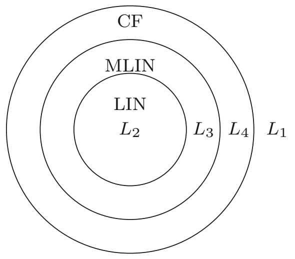

Fig. 1 Witness languages in the hierarchy of LIN, MLIN and CF: L_1, L_2, L_3, and L_4

$L_1 = \{a^n b^m c^n d^m \mid n, m \ge 1\} \in \mathrm{MAT}(2; 1, 1, 0; 0, 0, 0)$
$L_2 = \{a^n b^n \mid n \ge 1\} \in \mathrm{MAT}(2; 1, 0, 0; 1, 1, 1)$
$L_3 = \{a^n b^n c^m d^m \mid n, m \ge 0\} \in \mathrm{MAT}(2; 1, 1, 0; 1, 0, 0)$
$L_4 \in \mathrm{MAT}(1; 2, 0, 0; 0, 0, 0)$, with L_4 described by the context-free grammar with the rules $S \to SS$, $S \to aSb$, and $S \to \lambda$, also known as the Dyck language.

Now, if we want to proceed the derivation, then we have to start applying $m3$. This verifies that at least one occurrence of X should appear immediately to the right of $\#\$$ in the string. Once $m3$ is applied, we cannot apply $m1$ again (neither $m2$); otherwise, there will be at least one $\dagger$ which cannot be deleted by $m2$ (as the symbol $\$$ was already deleted by $m3$). Now, $m4$ can be applied and this forces that the symbol a introduced by $m4$ should be placed after $\#X$ and the X is deleted. Note that $m4$ can be applied only once and cannot be applied further, as now a is to the right of the symbol $\#$. Recall that previously no occurrences of a were in the string, which means that $m5$ was not previously applicable, as the second of its rules requires (at least) two occurrences of a in the string. To make all situations possible at this stage formally more precise, consider a new letter c' and a homomorphism $h' : \{X, Y, a, b, c', \dagger\}^* \to \{X, Y, a, b, \#, \$, \dagger\}^*$, given by $c' \mapsto \#a$ and $x \mapsto x$ for $x \in \{X, Y, a, b, \dagger\}$. Namely, the set of derivable words (as explained above) is then $h'(X^{n-1} ш Y^n ш c')$ for any $n \geq 1$. Now, in order to remove further occurrences of X (if any), $m5$ has to be applied repeatedly and this process ensures that all the occurrences of X were (originally) placed next to each other, so they formed a substring that was placed after the symbol $\$$ in the string. As the a's are later placed where the X's are deleted, they also form a substring. So, the set of words obtainable at this stage can be described as follows. Let c_n be a new symbol. Define the homomorphism $h_n : \{X, Y, a, b, c_n, \dagger\}^* \to \{X, Y, a, b, \#, \$, \dagger\}^*$ by setting $h_n(c_n) = \#a^n$ and $h_n(x) = x$ else. Then, the derivable strings are represented by $h_n(Y^n c_n)$ for any $n \geq 1$.

To move further on with a terminating derivation, now, $m6$ has to be applied ($m7$ cannot be applied before applying $m6$ once, since there are no two occurrences of b available to execute the second rule in $m7$). Applying $m6$ and then repeatedly $m7$ ensures that all the Y's were placed actually one next to each other, and this substring of Y's was originally placed after the last occurrence of X (in the meantime, the X's have been changed to a's). The process of repeated applications of $m7$ also rewrites all the Y to b. The derivation can be completed by applying $m8$, so that the non-terminal $\#$ is also deleted. If $m8$ had already been applied, say, sometimes after applying $m4$, then the derivation ends when all the Y's are changed to b's.

With the details provided we can see that the language is the double agreement language, which is linear, but not regular. A sample derivation about how the rules are applied for the word a^2b^2 is given below.

$$\#\$ \Rightarrow_{m1} \# \dagger \$X \Rightarrow_{m2} \#\$XY \Rightarrow_{m1} \# \dagger \$XX \Rightarrow_{m2} \#\$XXYY$$
$$\Rightarrow_{m3} \#XXYY \Rightarrow_{m4} \#aXYY \Rightarrow_{m8} aXYY \Rightarrow_{m5} aaYY$$
$$\Rightarrow_{m6} aabY \Rightarrow_{m7} aabb.$$

Thus, $L(\Gamma_2) = \{a^n b^n \mid n \geq 1\} \in \mathrm{MAT}(2; 1, 0, 0; 1, 1, 1)$.

Example 3 Let $L_3 = \{a^n b^n c^m d^m \mid n, m \geq 0\}$. Consider the matrix ins-del system

$$\Gamma_3 = (\{\#_1, \#_2, \#_3, \#_4, a, b\}, \{a, b\}, \{\#_1\#_2\#_3\#_4\}, R_3),$$

where R_3 is given by the following rules:

$$m1 = [(\#_1, a, \lambda)_{ins}, (\#_2, b, \lambda)_{ins}] \qquad m2 = [(\lambda, \#_1, \lambda)_{del}, (\lambda, \#_2, \lambda)_{del}]$$
$$m3 = [(\#_3, c, \lambda)_{ins}, (\#_4, d, \lambda)_{ins}] \qquad m4 = [(\lambda, \#_3, \lambda)_{del}, (\lambda, \#_4, \lambda)_{del}]$$

Starting from the axiom $\#_1\#_2\#_3\#_4$, applying $m1$ repeatedly (n times) will yield $\#_1 a^n \#_2 b^n \#_3\#_4$. To terminate the generation of a and b, $m2$ is applied to get the string $a^n b^n \#_3\#_4$. Now, $m1$ is no longer applicable. Then $m3$ is applied repeatedly (m times) to get $a^n b^n \#_3 c^m \#_4 d^m$. Finally, $m4$ is applied for stopping the derivation yielding the terminal string $a^n b^n c^m d^m$. Now, none of the matrices is applicable any more. Notice that applications of $m1$ and $m3$ can mix, but once $m2$ is applied, $m1$ can no longer be applied, and once $m4$ is applied, $m3$ is no longer applicable. With this, it is easy to see that $L(\Gamma_3) = L_3$ and $L_3 \in \mathrm{MAT}(2; 1, 1, 0; 1, 0, 0)$.

Note that in the example above, if we consider the matrices $m1$, $m2$ and $m4$ alone, then, we can generate the language L_2, thus only three matrices are sufficient when contexts are considered for insertion. On the other hand, Example 2 had contexts for deletion, but that did not help much to reduce the number of matrices. Thus, it seems that having contexts for insertion is more useful/powerful than to have on deletion. The following example is a well-known example of a context-free language that is not metalinear (Salomaa 1973). It is also interesting as there are even regular languages that cannot be described by $\mathrm{MAT}(k; 2, 0, 0; 2, 0, 0)$ for any k, (see Petre and Verlan 2012, Theorem 7).

Example 4 Let L_4 be the Dyck language (the set of balanced parenthesis) over the alphabet $\{a, b\}$, with a serving as left and b serving as right parenthesis.

Consider the following matrix ins-del system

$$\Gamma_4 = (\{a, b\}, \{a, b\}, \{\lambda\}, \{m1 = [(\lambda, ab, \lambda)_{ins}]\}).$$

Starting from λ, applying $m1$ for repeated times always yield another string in Dyck language. Thus, $L(\Gamma_4) = L_4$ and hence $L_4 \in \mathrm{MAT}(1; 2, 0, 0; 0, 0, 0)$.

From our examples and discussions, the hierarchical relationship among these families of languages is depicted in Fig. 1. In the figure, also the witness languages from the previous examples are placed.

2.3 Metalinear languages

Recall that a *linear grammar* is a context-free grammar $G = (N, T, S, P)$ whose productions are of the form $A \to x$, where A is a non-terminal symbol, and x is a word over N, with at most one occurrence of a non-terminal symbol. The language class LIN collects all languages that can be described by linear grammars.

LIN is not closed under concatenation. This motivates to consider the class MLIN of *metalinear languages*, which is inductively defined as follows:

- $\mathrm{LIN} \subseteq \mathrm{MLIN}$;
- If $L_1, L_2 \in \mathrm{MLIN}$, then $L_1 \cdot L_2 \in \mathrm{MLIN}$ and $L_1 \cup L_2 \in \mathrm{MLIN}$.
- No other languages are in MLIN.

For a detailed study of metalinear languages, we refer to Kutrib and Malcher (2007). It is known that MLIN is closed under union, concatenation, regular intersection, homomorphism and inverse homomorphism but not under Kleene closure. We now observe that the class MLIN is closed under reversal, as well, basically since $(L_1 \cdot L_2)^R = L_2^R \cdot L_1^R$ for two MLIN languages L_1 and L_2, where R is the reversal operator.

MLIN can be alternatively described via special grammars. We next define metalinear grammars as in Salomaa (1973) and Kutrib and Malcher (2007). To do so, we first define k-linear grammars.

The concept of a linear grammar can be generalized as follows: define a k-linear grammar as a context-free grammar $G = (N, T, S, P)$ such that every production in P has one of the three forms: (1) $A \to u$, (2) $A \to uBv$, (3) $S \to W$, where A, B are non-terminal symbols, not equal to the start symbol S, u, v are terminal words, and W is a word over N with no more than k occurrences of non-terminal symbols, and none of which is the start symbol S. A language is said to be k-linear if it can be generated by a k-linear grammar. Note that a language is 1-linear iff it is linear. A grammar is said to be *metalinear* if it is k-linear for some positive integer k. It should be clear that a language is metalinear if and only if it is generated by some metalinear grammar.

In our simulations, we need another grammatical characterization of metalinear languages (see Fernau et al. 2017c). Just notice that the rules in P_0 (in the description given below) start the linear grammars G_i one by one.

Proposition 1 *Let $L \subseteq T^*$. Then, $L \in \mathrm{MLIN}$ if and only if there are context-free grammars $G^j = (N^j, T, S^j, P^j)$, $j = 1, \ldots, n$, with pairwise disjoint nonterminal alphabets, satisfying $\bigcup_{j=1}^{n} L(G^j) = L$, together with some integer $k \geq 1$, satisfying the following properties.*

- *N^j can be partitioned into $N_0^j, N_1^j, \ldots, N_k^j$, where, for each $i = 1, \ldots, k$, $S_i^j \in N_i^j$.*
- *P^j can be partitioned into $P_0^j, P_1^j, \ldots, P_k^j$* such that *$G_i^j = (N_i^j, T, S_i^j, P_i^j)$ forms a linear grammar for each $i = 1, \ldots, k$.*
- *$P_0^j = \{S^j \to S_1^j(S_2^j)', (S_2^j)' \to S_2(S_3^j)', \ldots, (S_k^j)' \to S_k^j (S_{k+1}^j)', (S_{k+1}^j)' \to \lambda\}$ and $N_0^j = \{S^j, (S_2^j)', \ldots, (S_{k+1}^j)'\}$.*

3 Auxiliary results

In order to simplify the proofs of some of our main results, the following observations are helpful.

Theorem 1 *For all non-negative integers k, n, i', i'', m, j, j'', we have that*

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

Proof To an ins-del rule $r = (x, y, z)_\mu$ with $\mu \in \{ins, del\}$, we associate the reversed rule $\rho(r) = (z^R, y^R, x^R)_\mu$. Let $\Gamma = (V, T, A, R)$ be a matrix insertion-deletion system. Map a matrix $l = [r_1, \ldots, r_k] \in R$ to $\rho(l) = [\rho(r_1), \ldots, \rho(r_k)]$ in $\rho(R)$. Define $\Gamma^R = (V, T, A^R, \rho(R))$. Then, an easy inductive argument shows that $L(\Gamma^R) = (L(\Gamma))^R$. Observing the sizes of the system shows the claim. □

From Theorem 1, we can immediately deduce the following two corollaries:

Corollary 1 *Let k, n, i', m, j' be non-negative integers. The family of languages* MAT *$(k; n, i', i'; m, j', j')$ is closed under reversal.*

Proof By Theorem 1, $\mathrm{MAT}(k; n, i', i'; m, j', j') = [\mathrm{MAT}(k; n, i', i'; m, j', j')]^R$. This is nothing else than the claimed closure property. □

Corollary 2 *Let $\mathcal{L}$ be a language class that is closed under reversal. Then, for all non-negative integers $k, n, i', i'', m, j', j''$, we conclude that*

1. $\mathcal{L} = \mathrm{MAT}(k; n, i', i''; m, j', j'')$ if and only if $\mathcal{L} = \mathrm{MAT}(k; n, i'', i'; m, j'', j')$.
2. $\mathcal{L} \subseteq \mathrm{MAT}(k; n, i', i''; m, j', j'')$ if and only if $\mathcal{L} \subseteq \mathrm{MAT}(k; n, i'', i'; m, j'', j')$.

4 Computational completeness results

Recall that the matrix ins-del systems MAT(2; 1, 1, 0; 2, 0, 0), MAT(2; 2, 0, 0; 1, 1, 0), MAT(3; 1, 1, 0; 1, 1, 0), and MAT(3; 1, 1, 0; 1, 0, 1) are known to equal RE (cf. Petre and Verlan 2012). As RE is closed under reversal, the following theorem holds true as an easy consequence from Corollary 2.

Theorem 2 (1) $\mathrm{MAT}(2;1,0,1;2,0,0) = \mathrm{RE}$, (2) $\mathrm{MAT}(2;2,0,0;1,0,1) = \mathrm{RE}$,
(3) $\mathrm{MAT}(3;1,0,1;1,0,1) = \mathrm{RE}$, (4) $\mathrm{MAT}(3;1,0,1;1,1,0) = \mathrm{RE}$. □

In the following, we discuss further completeness results which are improvements over some of the existing results in terms of the maximal length of the matrix. Therefore, in each of the following theorems we also describe the key ideas and technical issues that brought along these improvements before delving into the details of the constructions. Notice that giving all the details including detailed arguments may look cumbersome, yet it is necessary, as with restricted resources, we are moving within dangerous terrain, i.e., it is very easy to design simulations that look convincing at first glance but fail a detailed analysis. Often enough, small changes in the simulations will make the difference. We will give a concrete account of this delicate issue before we present Theorem 4.

In Petre and Verlan (2012), matrices of maximum length 8 and size (1, 0, 0; 1, 1, 1) were used. This maximum length was reduced to 4 in Fernau et al. (2016), which is further reduced even to 3 in the following theorem.

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

Before we formally construct a matrix ins-del system of size (3; 1, 0, 0; 1, 1, 1) to describe RE, we describe some key issues of our construction.

- At least one of the rules of every matrix has a *rule marker* as a context or the marker itself is deleted. A matrix of this type is said to be *guarded*. The importance of a matrix being guarded is that it can be applied only in the presence of the corresponding rule marker. This will avoid interference of any other matrix application.
- After successful application of every matrix, either a rule marker remains or the intended simulation is completed.

Proof Formally, consider a type-0 grammar $G = (N, T, P, S)$ in SGNF. The rules from P are supposed to be labelled injectively with labels from the set $[1\ldots|P|]$, with label sets P_{ll} and P_{rl} as defined above. Also recall that the nonterminal alphabet decomposes like $N = N' \cup N''$, $N'' = \{A, B, C, D\}$, $S, S' \in N'$, according to the normal form. We construct a matrix insertion-deletion system $\Gamma = (V, T, \{S\}, M)$, where the alphabet of Γ is

$$V = N \cup T \cup \{p, p' \mid p \in P_{rl}\} \cup \{q, q' \mid q \in P_{ll}\} \cup \{f, f', g, g'\}\,.$$

The set of matrices M of Γ consists of the matrices described case-by-case in the following.

We simulate the rule p: $X \to bY$, $X, Y \in N'$, $b \in N'' \cup T$, i.e., $p \in P_{rl}$, by the following matrices:

$$\begin{aligned} p1 &= [(\lambda, p, \lambda)_{ins}, (\lambda, p', \lambda)_{ins}, (p', X, p)_{del}] \\ p2 &= [(\lambda, b, \lambda)_{ins}, (\lambda, Y, \lambda)_{ins}, (b, p, Y)_{del}] \\ p3 &= [(\lambda, p', b)_{del}] \end{aligned}$$

We simulate the rule q: $X \to Yb$, $X, Y \in N'$, $b \in N'' \cup T$, i.e., $q \in P_{ll}$, by the following matrices:

$$\begin{aligned} q1 &= [(\lambda, q, \lambda)_{ins}, (\lambda, q', \lambda)_{ins}, (q, X, q')_{del}] \\ q2 &= [(\lambda, b, \lambda)_{ins}, (\lambda, Y, \lambda)_{ins}, (Y, q, b)_{del}] \\ q3 &= [(b, q', \lambda)_{del}] \end{aligned}$$

We simulate the rule f: $AB \to \lambda$, $A, B \in N''$, by the following two matrices:

$$\begin{aligned} f1 &= [(\lambda, f, \lambda)_{ins}, (\lambda, f', \lambda)_{ins}, (f, A, B)_{del}] \\ f2 &= [(f, B, f')_{del}, (\lambda, f', \lambda)_{del}, (\lambda, f, \lambda)_{del}] \end{aligned}$$

Similarly, we simulate the rule g: $CD \to \lambda$, $C, D \in N''$, by the following matrices:

$$\begin{aligned} g1 &= [(\lambda, g, \lambda)_{ins}, (\lambda, g', \lambda)_{ins}, (g, C, D)_{del}] \\ g2 &= [(g, D, g')_{del}, (\lambda, g', \lambda)_{del}, (\lambda, g, \lambda)_{del}] \end{aligned}$$

Finally, we simulate the rule $h : S' \to \lambda$ by the ins-del rule $[(\lambda, S', \lambda)_{del}]$.

We now proceed to prove that $L(\Gamma) = L(G)$. We initially prove that $L(G) \subseteq L(\Gamma)$ by showing that Γ correctly simulates the application of the rules of the types p, q, f, g, h, as discussed above. We explain the working of the simulation matrices for the cases p and f mainly, as the working of q and g simulation matrices are similar, and as the working of the simulation of the h rule is simple and direct.

Simulation of $p : X \to bY$: Consider the string $\alpha X \beta$ derivable from S in G, with $X \in N'$ and $\alpha, \beta \in (N'' \cup T)^*$. We now show that on applying the matrices introduced for simulating rules from P_{rl}, we can derive $\alpha bY\beta$ within Γ, starting from $\alpha X \beta$. We start by applying the rules of matrix $p1$. The markers p and p' are randomly inserted by the first two rules, leading to a string from $p ш p' ш \alpha X \beta$. However, the third rule of $p1$ is applicable only when p' and p are inserted before and after the non-terminal X. This shows that $\alpha X \beta \Rightarrow_{p1} \gamma$ is possible if and only if $\gamma = \alpha p' p \beta$. After the X has been deleted by the third rule, we note that matrix $p1$ cannot be applied again, since there was only one non-terminal of N' (in this case, X) present in $\alpha X \beta$, and this is deleted. On applying matrix $p2$, b and Y are inserted anywhere, so intermediately we arrive at a string from $b ш Y ш \alpha p' p \beta$; then, p is deleted in the contexts of b and Y. The importance of p' here is to make sure that the left context of p is the introduced b only, so that now we know that we arrived at the string $\alpha p'bY\beta$. Finally, p' is deleted by the matrix $p3$. The right context b in the singleton matrix $p3$ makes it applicable only after applying $p2$. The intended sequence of derivations is hence: $\alpha X\beta \Rightarrow_{p1} \alpha p'p\beta \Rightarrow_{p2} \alpha p'bY\beta \Rightarrow_{p3} \alpha bY\beta$.

Simulation of $f : AB \to \lambda$: Let $\alpha AB\beta$ be a sentential form derivable in G with $A, B \in N''$ and $\alpha, \beta \in (N'' \cup T)^*$. The deletion of AB can be simulated as follows. The application of $f1$ to $\alpha AB\beta$ leads to the string $\alpha fBf'\beta$. Note that, strictly speaking, at this point the (derived) marker f' can be placed anywhere in the string. Also, it is possible to apply $f1$ again, but then there should be some other A and B present together somewhere in the string. Applying now $f2$ forces the f' to be placed after the intended B. At the end of applying all the rules of $f2$, we end up at $\alpha\beta$ as desired.

To prove the reverse relation ($L(\Gamma) \subseteq L(G)$), we observe that the rules of Γ are applied in groups and each group of rules corresponds to one of p, q, f, g, h. Let us indicate the observations necessary to conclude this in the following.

1. It can be seen by induction that, before and after successfully applying a matrix to any sentential form, this sentential form contains at most one symbol of N'.
2. A matrix of the form $p1$ can only be successfully applied to a sentential form of the form $\alpha X\beta$, and applying $p1$ would result in $\alpha p'p\beta$. In particular, the resulting sentential form contains no symbol from N'. Notice that, before and after successfully applying a matrix to any sentential form, this sentential form either does not contain p at all or contains p' immediately left of p, respectively.
3. A matrix of the form $p2$ can only be successfully applied to a sentential form of the form $\alpha p'p\beta$, and applying $p2$ would result in $\alpha p'bY\beta$.
4. A matrix of the form $p3$ can only be successfully applied to a sentential form of the form $\alpha p'\beta$ if β starts with b. Therefore, it can only be applied after $p2$ has been applied, so that $\alpha p'bY\beta$ would be turned into $\alpha bY\beta$.
5. Summarizing the previous three observations, we can conclude that once we start using $p1$, then $p2$ and $p3$ have to follow, and all in all this converts $\alpha X\beta$ into $\alpha bY\beta$, so this corresponds to applying the context-free rule $X \to bY$. Also, there is no other possibility to successfully apply $p2$ or $p3$.
6. We can establish a similar reasoning for the matrices $q1$, $q2$ and $q3$, corresponding to $q \in P_{ll}$.
7. A matrix of the form $f1$ can only be successfully applied to a sentential form like $\alpha AB\beta$. First, f and f' are introduced at some arbitrary positions in the sentential form, and then A is deleted with f and B as its left and right contexts, leading to a sentential form from $f' ш \alpha fB\beta$. Then, applying $f2$ ensures that f' was actually placed just after B and now B and the markers f and f' are deleted. In the case that, before applying $f2$, $f1$ is applied again, then it is applied for some other AB that is present in the string. But, this does not affect applying $f2$ at some other point. As both $f1$ and $f2$ are guarded, it will have no interference with other matrices like $g1$ and $g2$. As the rule marker f (and similarly, g) act as placeholders for A (and similarly, for C), it could well happen that $f1$ and $g1$ are applied several times before removing these rule markers again by applying $f2$ and $g2$. However, by the placeholder observation, such a derivation can only be (finally) terminating if the primed rule markers can match up. In that case, we could as well apply, say, $f2$ immediately after we applied $f1$. This way we can disentangle possibly interleaved applications of $f1$, $f2$, $g1$ and $g2$. With these details provided, we can see that if $\alpha AB\beta$ was a sentential form and on applying $f1$ and $f2$, we will get the sentential form $\alpha\beta$, and this happens without loss of generality.

These observations complete the proof. □

In Fernau et al. (2016), a computational completeness result for matrix ins-del systems with size (2; 1, 1, 1; 1, 0, 0) was claimed. However, as noticed by S. Verlan (personal communication), the proof is prone to error for the following reason that we were unable to circumvent. For this discussion, we refer the reader to the proof sketch in Fernau et al. (2016). When the (deletion) rules are applied in a context-free manner in Penttonen normal form and if a variable (say X) can appear several times in a sentential form, then, an application of a deletion rule for a variable (X) can be misused at some other position where it is present. In order to make sure the result is correct, now, we increase the matrix length from 2 to 3 and also we use SGNF instead of the Penttonen normal form as in the mentioned proof sketch. Thus, in the next theorem, we will prove a (weaker) computational completeness result for matrix ins-del system with size (3; 1, 1, 1; 1, 0, 0). However, this is still an improvement over the computational completeness result of Petre and Verlan (2012), as discussed in the Introduction.

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

Let us again highlight some key features of this construction first.

- Again, we make use of rule markers to guard the simulation of rules applications.

- In order to simulate, say, $AB \to \lambda$, when we use the deletion rules $(\lambda, A, \lambda)_{del}$ and $(\lambda, B, \lambda)_{del}$, it may be the case we are deleting unintended occurrences. Notice that deletions cannot be performed under contexts. So we have to carefully place markers before, after and between the nonterminals A and B in order to check that only the intended nonterminal occurrences are deleted.
- Though we are allowed to use both the contexts for insertion rules, we use only one context (left or right) in most of our insertion rules.

Proof Consider a type-0 grammar $G = (N, T, P, S)$ in SGNF, with the rules uniquely labelled with $[1 \ldots |P|]$. Again, we use the label subsets P_{rl} and P_{ll} as described above. Recall the decomposition $N = N' \cup N''$ by SGNF. We can construct a matrix ins-del system $\Gamma = (V, T, \{S\}, M)$ with alphabet
$V = N \cup T \cup P_{rl} \cup P_{ll} \cup \{f, f', f'', g, g', g''\}$.

The set of matrices M is defined as follows.

We simulate the rule $p : X \to bY$, i.e., $p \in P_{rl}$, by the following matrices:

$$\begin{aligned} p1 &= [(\lambda, p, X)_{ins}, (\lambda, X, \lambda)_{del}] \\ p2 &= [(p, Y, \lambda)_{ins}, (p, b, \lambda)_{ins}, (\lambda, p, \lambda)_{del}] \end{aligned}$$

We simulate the rule $q : X \to Yb$, i.e., $q \in P_{ll}$ by the following matrices:

$$\begin{aligned} q1 &= [(\lambda, q, X)_{ins}, (\lambda, X, \lambda)_{del}] \\ q2 &= [(q, b, \lambda)_{ins}, (q, Y, \lambda)_{ins}, (\lambda, q, \lambda)_{del}] \end{aligned}$$

We simulate the rule $f : AB \to \lambda$ by the following matrices:

$$\begin{aligned} f1 &= [(\lambda, f, A)_{ins}, (B, f'', \lambda)_{ins}] \\ f2 &= [(\lambda, A, \lambda)_{del}, (\lambda, B, \lambda)_{del}, (f, f', f'')_{ins}] \\ f3 &= [(\lambda, f', \lambda)_{del}, (\lambda, f, \lambda)_{del}, (\lambda, f'', \lambda)_{del}] \end{aligned}$$

We simulate the rule $g : CD \to \lambda$ by the following matrices:

$$\begin{aligned} g1 &= [(\lambda, g, C)_{ins}, (D, g'', \lambda)_{ins}] \\ g2 &= [(\lambda, C, \lambda)_{del}, (\lambda, D, \lambda)_{del}, (g, g', g'')_{ins}] \\ g3 &= [(\lambda, g', \lambda)_{del}, (\lambda, g, \lambda)_{del}, (\lambda, g'', \lambda)_{del}] \end{aligned}$$

We simulate the rule $h : S' \to \lambda$ by the matrix $[(\lambda, S', \lambda)_{del}]$.

We now proceed to prove that $L(\Gamma) = L(G)$. We initially prove that $L(G) \subseteq L(\Gamma)$ by showing that Γ correctly simulates the application of the above rules p, q, f, g, h. We discuss the working of the simulation rules p and f mainly as the working of rules q and g are similar to the working of p and f, respectively, and the working of h rule is straightforward.

Simulation of $p : X \to bY$: Consider the string $\alpha X \beta$, $X \in N'$, $\alpha, \beta \in (N'' \cup T)^*$, derivable from S in G. We now show that on applying p-rules, we can derive $\alpha bY\beta$ from $\alpha X \beta$. We start by applying the rules of matrix $p1$. The rule marker p is inserted to the left of X and then X is deleted. Since we are using SGNF, we note that there is only one variable (actually X) of N' present in the string and hence this deletion of X is unique. This also ensures that the matrix $p1$ cannot be applied immediately again. Hence at this point, a single application of $p1$ on the string $\alpha X \beta$ yields $\alpha p \beta$. Next, on applying matrix $p2$, Y and b are inserted after p (in order) and then p is deleted. The matrix $p2$ cannot be applied for a second time as the unique p (one application of $p1$ yielded one p) is deleted and this gives $\alpha bY\beta$. This unique derivation can be represented as $\alpha X \beta \Rightarrow_{p1} \alpha p \beta \Rightarrow_{p2} \alpha bY\beta$.

Simulation of $f : AB \to \lambda$: Let $\alpha AB\beta$ be a sentential form derivable in G, with $A, B \in N''$ and $\alpha, \beta \in (N'' \cup T)^*$. The application of $f1$ (say, k_1 times) to $\alpha AB\beta$ leads to the string $\alpha f^{k_1} AB (f'')^{k_1} \beta$. Note that, strictly speaking, an application of $f1$ can place the marker f'' after any occurrence of B that is not necessarily right of the occurrence of A that we chose to discuss here. Starting for example from $\alpha AB\delta B\beta'$, an application of $f1$ may yield the string $\alpha fAB\delta Bf''\beta'$. In this case, we cannot apply $f2$ and hence also not $f3$, thus leaving the markers f and f'' not deleted from the string. Hence, the markers f and f'' have to be inserted on either sides of one occurrence of AB (as chosen in our discussion) for further continuation. Now applying $f2$, the substring AB (situated between f^{k_1} and $(f'')^{k_1}$) is deleted; if not, i.e., if other occurrences of A and B are deleted, the third rule of $f2$ cannot be applied. Hence on applying $f2$ on the string $\alpha f^{k_1} AB (f'')^{k_1} \beta$, we get $\alpha f^{k_1} f' (f'')^{k_1} \beta$. On applying $f3$ once, one occurrence of all markers is deleted and we obtain $\alpha f^{k_1 - 1} (f'')^{k_1 - 1} \beta$. If $k_1 \geq 2$ at this point, no matrix is applicable. Thus, to have a terminating derivation, k_1 has to be 1, i.e., $f1$ has to be applied only once, corresponding to simulating $AB \to \lambda$. The intended working of the simulation of the f rule is represented as $\alpha AB\beta \Rightarrow_{f1} \alpha fABf''\beta \Rightarrow_{f2} \alpha ff'f''\beta \Rightarrow_{f3} \alpha\lambda\beta$.

To prove the reverse relation ($L(\Gamma) \subseteq L(G)$), we observe that the rules of Γ are applied in groups and each group of rules corresponds to one of p, q, f, g, h. Let us indicate the observations necessary to conclude this in the following. The Items 1.-4. from the proof of Theorem 3 carry over (nearly) literally and are hence omitted. For Item 2., the last sentence should read: Notice that, before successfully applying the $p1$ matrix to any sentential form, this sentential form does not contain p at all; and after successfully applying the $p1$ matrix to any sentential form, this sentential form contains exactly one p. For Item 4., observe that the result of applying $p2$ is $\alpha bY\beta$.

- Summarizing Items 4. and 5., we can conclude that once we start using $p1$, then $p2$ has to follow, and all in all this converts $\alpha X\beta$ into $\alpha bY\beta$, so this corresponds to applying the context-free rule $X \to bY$. Also, there is no other possibility to successfully apply $p2$ in the absence of a $p1$ application.
- We can establish a similar reasoning for the matrices $q1$ and $q2$.
- A matrix of the form $f1$ can be successfully applied to a sentential form like $\alpha A\delta_1 A\delta B\delta_2 B\beta$. First, f and f'' are introduced to the left and right of A and B respectively, leading to the sentential form $\alpha A\delta_1 fA\delta Bf''\delta_2 B\beta$. If $f1$ is applied for the second time, then the string becomes $\alpha fA\delta_1 fA\delta Bf''\delta_2 Bf''\beta$. At this point, no matrix can be applied (specifically, due to the last rule in $f2$ and first rule in $f3$) unless $\delta = \lambda$, if we want to avoid applying $f1$ once more. The only matrix that can handle the newly introduced nonterminals f and f'' is $f2$. On applying $f2$, the string becomes $\alpha fA\delta_1 ff'f''\delta_2 Bf''\beta$. Finally, with the string $\alpha fA\delta_1 ff'f''\delta_2 Bf''\beta$ in hand, the only matrix that can deal with f, f' and f'' is matrix $f3$ which deletes all these markers and yields either $\alpha A\delta_1 ff''\delta_2 B\beta$ or $\alpha fA\delta_1\delta_2 Bf''\beta$. In the former case, there is no possibility to apply any matrix further. In the latter case, matrix $f2$ is applicable if $\delta_1 = \delta_2 = \lambda$, in which case the resultant string is $\alpha ff'f''\beta$. On reapplying $f3$, we get $\alpha\beta$.
- It is important to observe that the above derivation which takes the string $\alpha A\delta_1 A\delta B\delta_2 B\beta$ to $\alpha\beta$ was possible under the assumptions $\delta_1 = \delta = \delta_2 = \lambda$ which implies that the initial string (before the start of simulation) was $\alpha AABB\beta$. The simulation of $\alpha AABB\beta$ to $\alpha\beta$ is intended. Moreover, we could have obtained the same result by the following derivation which uses $f1$, $f2$, $f3$ in this order, which was the way we explained this type of simulation before.

$$\alpha AABB\beta \Rightarrow_{f1} \alpha AfABf''\beta \Rightarrow_{f2} \alpha Aff'f''\beta \Rightarrow_{f3} \alpha AB\beta \Rightarrow_{f1} \ldots \Rightarrow_{f3} \alpha\beta$$

  This idea shows that we can always disentangle derivations that mix $f1$ and $f2$ applications.
- Similar observations apply to $g1$, $g2$ and $g3$. We can even disentangle applications of matrices $f1, f2, g1$ and $g2$ so that $f2$ immediately follows on $f1$, and $g2$ immediately follows $g1$.

The observations above complete our reasoning. □

It was proved in Petre and Verlan (2012) that MAT(3; 1, 1, 0; 1, 1, 0) and MAT(2; 1, 1, 0; 2, 0, 0) both describe RE. As a trade-off between the two, we prove that MAT(2; 1, 1, 0; 1, 1, 1) describe RE in the following theorem. Given a size (1, 1, 0; 1, 1, 1), it is proved in Krassovitskiy et al. (2008) that a ins-del systems (corresponding to matrix ins-del systems of length one) are not computationally complete. Hence one needs at least length 2 for matrix ins-del systems to describe RE. In the following, we show that length 2 is sufficient and hence this length is optimal.

Theorem 5 MAT(2; 1, 1, 0; 1, 1, 1) = MAT(2; 1, 0, 1; 1, 1, 1) = RE.

In this construction, along with the axiom, we introduce two dummy symbols # and \$ in order to restrict certain matrix rules to be applied only once. This trick was crucial to achieve the desired computational completeness results with the given resource restrictions. Notice that a similar unusual use of nonterminal symbols can be found in Example 2

Proof Consider a type-0 grammar $G = (N, T, P, S)$ in SGNF. The rules from P are supposed to be labelled injectively with labels from the set $[1 \ldots |P|]$, with label sets P_{rl} and P_{ll} as introduced above. Recall the decomposition $N = N' \cup N''$ of the nonterminal alphabet N of G. We now construct a matrix ins-del system $\Gamma = (V, T, \{S\#\$\}, M)$ with alphabet

$$V = N \cup T \cup \{\#, \$\} \cup \{x, x', x'', x''' \mid x \in P_{rl} \cup P_{ll}\} \cup \{f, f', g, g'\}.$$

The set of matrices M is defined as follows.

The rules p: $X \to bY$ and q: $X \to Yb$ are simulated by the following set of ins-del matrices shown on the left and right side, respectively:

$$\begin{aligned} p1 &= [(X, p, \lambda)_{ins}, (\#, p', \lambda)_{ins}] & q1 &= [(X, q, \lambda)_{ins}, (\#, q', \lambda)_{ins}] \\ p2 &= [(\lambda, X, p)_{del}, (\#, p'', \lambda)_{ins}] & q2 &= [(\lambda, X, q)_{del}, (\#, q'', \lambda)_{ins}] \\ p3 &= [(p, Y, \lambda)_{ins}, (\#, p''', \lambda)_{ins}] & q3 &= [(q, b, \lambda)_{ins}, (\#, q''', \lambda)_{ins}] \\ p4 &= [(p, b, \lambda)_{ins}, (p''', p'', p')_{del}] & q4 &= [(q, Y, \lambda)_{ins}, (q''', q'', q')_{del}] \\ p5 &= [(\lambda, p, b)_{del}, (p''', p', \$)_{del}] & q5 &= [(\lambda, q, Y)_{del}, (q''', q', \$)_{del}] \\ p6 &= [(\#, p''', \$)_{del}] & q6 &= [(\#, q''', \$)_{del}] \end{aligned}$$

The rules f: $AB \to \lambda$ and g: $CD \to \lambda$ are simulated by the set of matrices shown on the left and right side, respectively:

$$\begin{aligned} f1 &= [(B, f, \lambda)_{ins}, (\#, f', \lambda)_{ins}] & g1 &= [(D, g, \lambda)_{ins}, (\#, g', \lambda)_{ins}] \\ f2 &= [(\lambda, B, f)_{del}, (\lambda, A, f)_{del}] & g2 &= [(\lambda, D, g)_{del}, (\lambda, C, g)_{del}] \\ f3 &= [(\lambda, f, \lambda)_{del}, (\#, f', \$)_{del}] & g3 &= [(\lambda, g, \lambda)_{del}, (\#, g', \$)_{del}] \end{aligned}$$

A rule h: $S' \to \lambda$ is simulated by the matrix $[(\lambda, S', \lambda)_{del}]$. The whole simulation is then terminated by applying the matrix $[(\#, \$, \lambda)_{del}, (\lambda, \#, \lambda)_{del}]$.

If this matrix was not applied at the end, possibly earlier, then the rules p, q, f, g cannot be simulated. Also, once a simulation is started, there will be a rule marker between # and \$, so that the termination matrix cannot be applied in the middle of a simulation.

We now discuss the working of the simulation of rules p and f mainly. The working of the rules q and g are similar to the working of the simulation of rules p and f and hence not discussed. The working of the matrix h is straightforward.

Simulation of $p : X \rightarrow bY$ Let the string $\alpha X\beta\#\$$ be derived from the axiom $S\#\$$ within Γ, with $X \in N'$ and $\alpha, \beta \in (N'' \cup T)^*$, having simulated the derivation of $\alpha X\beta$ from S in G. Using $p1$, the rule marker p is introduced right to X and another marker p' is introduced right to $\#$. In principle, the matrix $p1$ can be repeatedly applied several times, say $k_1 \geq 1$ times, yielding the string $\alpha Xp^{k_1}\beta\#(p')^{k_1}\$$. Now, when $p2$ is applied, it will introduce p'' to the right of $\#$ and delete the only symbol of N', which is X. Thus, $p2$ cannot be applied for a second time, as there will be no X present in the string. With the string of the form $\alpha p^{k_1}\beta\#p''(p')^{k_1}\$$, now, the matrix $p3$ is applied, which will introduce a Y after the marker p, and p''' is introduced after $\#$. Let $p3$ be applied repeatedly for $k_2 \geq 1$ times, yielding possibly $\alpha p^{k_1}Y^{k_2}\beta\#(p''')^{k_2}p''(p')^{k_1}\$$. As $X \neq Y$, $X, Y \in N'$, repeated applications of either of the matrices $p1$, $p2$ is not possible. On applying p_4 once, we have $\alpha p^{k_1}bY^{k_2}\beta\#(p''')^{k_2}(p')^{k_1}\$$. Note that the matrix p_4 cannot be applied for a second time, since there was only one p'' and it was deleted by the application of p_4. The matrix p_5 can be applied to the string $\alpha p^{k_1}bY^{k_2}\beta\#(p''')^{k_2}(p')^{k_1}\$$ only if $k_1 = 1$ due to the presence of p''', with p''' and $\$$ as left and right context of p'', respectively. In this case, the resultant string is $\alpha bY^{k_2}\beta\#(p''')^{k_2}\$$. At this point, the only applicable matrix is $p6$, since the other matrices demand the marker p, but p was already deleted in the previous step. To apply the matrix $p6$, the left and right contexts demand $k_2 = 1$ and the resulting string is $\alpha bY\beta\#\$$. Thus, with the string $\alpha pY\beta\#p'''p''p'\$$, on applying the matrix $p4$, $b \in N'' \cup T$ in the p rule is introduced after p and the marker p'' placed in between p''' and p' is deleted. This will end up with the string $\alpha pbY\beta\#p'''p'\$$ and using the matrix $p5$, the markers p and p' are deleted. Finally, the matrix $p6$ is applied which will delete the marker p''' available in between $\#$ and $\$$ and end up with the string $\alpha bY\beta\#\$$, thus the p rule is simulated. Notice that, after Y is introduced in the derivation, and if Y shows up as the left-hand side of a rule of G, say, $r : Y \rightarrow cZ$, then r cannot be introduced (during the simulation of p rule), by using the matrix $r1$, as there are some primed markers present in between $\#$ and $\$$. Thus, one can notice that the matrices are applied in quite a deterministic way during the simulation. We show the intended whole derivation simulating the application of a p-rule below.

$$\alpha X\beta\#\$ \Rightarrow_{p1} \alpha Xp\beta\#p'\$ \Rightarrow_{p2} \alpha p\beta\#p''p'\$ \Rightarrow_{p3} \alpha pY\beta\#p'''p''p'$$
$$\Rightarrow_{p4} \alpha pbY\beta\#p'''p'\$ \Rightarrow_{p5} \alpha bY\beta\#p'''\$ \Rightarrow_{p6} \alpha bY\beta\#\$.$$

Simulation of $f : AB \rightarrow \lambda$: Let $\alpha AB\beta\#\$$ be a sentential form derivable in Γ, such that $\alpha AB\beta$ is derivable in G with $A, B \in N''$ and $\alpha, \beta \in (N'' \cup T)^*$. We can only start from $f1$ (in order to simulate rule f), as other matrices $f2$ and $f3$ contain some deletion rules based on the symbols f or f' introduced by matrix $f1$. Applying $f1$ introduces the rule marker f to the right of some occurrence of B and another marker f' to the right of $\#$, yielding, for instance in our case, $\alpha ABf\beta\#f'\$$. Though, in principle, $f1$ can be repeatedly applied, a further application of the matrix is thwarted by the use of $f3$; if $f3$ is applied at some other point to delete the introduced markers f and f' (and these markers must be deleted in order to successfully terminate a derivation and can only be deleted by using matrix $f3$), then matrix $f1$ must have been used only once, as otherwise there cannot be a marker f' with $\#$ to its left and $\$$ to its right. Now, if $f2$ is applied, this will delete the occurrence of B that is to the left of the previously introduced marker f and the occurrence of A which is left to the occurrence of B that got deleted by the first rule of $f2$, thus, the correctly chosen occurrences of A and B are deleted. The previously introduced rule markers f and f' are deleted by the matrix $f3$ and this yields the string $\alpha\beta\#\$$. If $f3$ is applied immediately after applying $f1$, this will make no change in the sentential form all in all and such applications are neither useful nor harmful. The above discussed intended working is represented as follows.

$$\alpha AB\beta\#\$ \Rightarrow_{f1} \alpha ABf\beta\#f'\$ \Rightarrow_{f2} \alpha f\beta\#f'\$ \Rightarrow_{f3} \alpha\beta\#\$.$$

The marker deletion matrix $[(\#, \$, \lambda)_{del}, (\lambda, \#, \lambda)_{del}]$ is applied at the end of the derivation of words. Hence on starting at $S\#\$$ and by repeatedly applying the simulations of the rules p, q, f, g, h, and the marker deletion rule we eventually get $S\#\$ \Longrightarrow_* w$. This proves that $L(G) \subseteq L(\Gamma)$.

To prove the reverse relation ($L(\Gamma) \subseteq L(G)$), we observe that the rules of Γ are applied in groups and each group of rules corresponds to one of p, q, f, g, h. We already gave several arguments why the simulation of the rules of G work in quite a deterministic fashion. Let us indicate some further observations necessary to conclude $L(\Gamma) \subseteq L(G)$ in the following.

- It can be seen by induction that, before and after successfully applying a matrix to any sentential form, this sentential form contains at most one symbol of N', or such a derivation cannot terminate (as discussed above).
- As either the rule markers (like p or f) are introduced or used as guards in each matrix (which means that they

serve as contexts or are deleted), interference by mixing-up the simulations is avoided. In particular, the primed rule markers (like p', p'', p''', or f') are meant to be introduced between # and \$, which is tested upon deleting. If more than one primed marker is introduced, trying to open up a derivation that mixes up two simulations, then these primed markers can never be deleted, so that such a derivation can never terminate. This has been explained in details above.

- Once a simulation of rule x has started by applying a matrix $x1$, the matrices $x2$ etc. have to be applied, in this order, as argued above when explaining how the simulation of the rules (should) work.

As RE is closed under reversal, Corollary 2 yields $\mathrm{MAT}(2;1,0,1;1,1,1) = \mathrm{RE}$. □

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

The trick of using auxiliary symbols # and \$ in the axiom carries over from the previous construction. However, notice that allowing one-sided deletions only is distinctively different from allowing one-sided insertions only (as int he previous construction). Important checks have to be undertaken in different parts of the simulation of a rule of the given SGNF grammar.

Proof Consider a type-0 grammar $G = (N, T, P, S)$ in SGNF. In particular, this means that N is split into N' and N''. The rules from P in G are supposed to be labelled injectively with labels from the set $[1 \dots |P|]$, with P_{ll} and P_{rl} being disjoint subsets of labels (as explained above). We now construct a matrix ins-del system $\Gamma = (V, T, \{S\#\$\}, M)$ with

$$V = N \cup T \cup \{\#, \$\} \cup \{x, x', x'', x''' \mid x \in P_{ll} \cup P_{rl}\} \\ \cup \{f, f', g, g'\}.$$

The set of matrices M is defined as follows.

The rules p: $X \to bY$ and q: $X \to Yb$ are simulated by the following ins-del matrices shown on the left and right, respectively:

$$\begin{array}{ll}
p1 = [(\lambda, p, X)_{ins}, (\#, p', \$)_{ins}] & q1 = [(\lambda, q, X)_{ins}, (\#, q', \$)_{ins}] \\
p2 = [(p, X, \lambda)_{del}, (\#, p'', p')_{ins}] & q2 = [(q, X, \lambda)_{del}, (\#, q'', q')_{ins}] \\
p3 = [(\lambda, b, p)_{ins}, (\#, p''', p'')_{ins}] & q3 = [(\lambda, Y, q)_{ins}, (\#, q''', q'')_{del}] \\
p4 = [(b, Y, p)_{ins}, (p''', p'', \lambda)_{del}] & q4 = [(Y, b, q)_{ins}, (q''', q'', q')_{del}] \\
p5 = [(p''', p', \lambda)_{del}] & q5 = [(q''', q', \lambda)_{del}] \\
p6 = [(Y, p, \lambda)_{del}, (\#, p''', \lambda)_{del}] & q6 = [(b, q, \lambda)_{del}, (\#, q''', \lambda)_{del}]
\end{array}$$

The rules f: $AB \to \lambda$ and g: $CD \to \lambda$ are simulated by the following ins-del matrices shown on the left and right, respectively:

$$\begin{array}{ll}
f1 = [(\lambda, f, A)_{ins}, (\#, f', \$)_{ins}] & g1 = [(\lambda, g, C)_{ins}, (\#, g', \$)_{ins}] \\
f2 = [(f, A, \lambda)_{del}, (f, B, \lambda)_{del}] & g2 = [(g, C, \lambda)_{del}, (g, D, \lambda)_{del}] \\
f3 = [(\lambda, f, \lambda)_{del}, (\#, f', \lambda)_{del}] & g3 = [(\lambda, g, \lambda)_{del}, (\#, g', \lambda)_{del}]
\end{array}$$

A rule h: $S' \to \lambda$ is simulated by the matrix $[(\lambda, S', \lambda)_{del}]$. The whole simulation is then terminated by applying the matrix $[(\#, \$, \lambda)_{del}, (\lambda, \#, \lambda)_{del}]$. If this matrix was not applied at the end, possibly earlier, then the rules p, q, f, g cannot be simulated.

We now discuss the working of the simulation rules p and f mainly, as the working of q and g rule are similar to the working of p and f, respectively, and the working of h is direct.

Simulation of $p : X \to bY$ Let the string $\alpha X \beta \# \$$ be derived from the axiom $S\#\$$ in Γ, with $X \in N'$ and $\alpha, \beta \in (N'' \cup T)^*$, corresponding to a derivation $S \Rightarrow^* \alpha X \beta$ in G. We explain how to simulate applying the rule p in G to $\alpha X \beta$, within Γ. As the first rules of the matrices $p2$ to $p6$ involve the introduced rule markers, we have to start from $p1$ and applying it to the string, we obtain $\alpha p X \beta \# p' \$$. The matrix $p1$ cannot be applied again until there is no symbol present between # and \$. Now, the only applicable matrix is $p2$ (as no other matrix can cope with the substring $\#p'\$$), which will delete $X \in N'$ and introduce another marker p'' between # and p'. Now, with the string $\alpha p \beta \# p'' p' \$$, the only applicable rule is $p3$ as all other matrices use p''' either as context or for deletion and p''' is only introduced by $p3$. On applying $p3$, the b that corresponds to the rule $p : X \to bY$ is introduced to the right of p and the marker p''' is inserted between # and p''. In view of the contexts present in the second rule of the matrices $p2$ and $p3$, they cannot be applied again. Now, with the resultant string $\alpha b p \beta \# p''' p'' p' \$$, the only applicable matrix is $p4$ and applying it will introduce the symbol Y of N' that is associated with the p rule and delete the marker p'' placed in between p''' and p'. Now, we have a choice to apply $p5$ or $p6$. However, if $p6$ is applied before applying $p5$, then the marker p' cannot be deleted at all, as that requires the context p''' to be present in the string. Also, we cannot introduce p''' again, as this would require to introduce p'' first, using matrix $p2$, but this is impossible, as X is not present. Yet, if p' is not deleted, then no other simulation is possible (except the h rule) as starting the simulation of p, q, f and g rules demand the substring #\$ to be present in the string. Thus, $p5$ and $p6$ are applied in this order, which results in the string $\alpha b Y \beta \# \$$. We present a sample derivation of the intended simulation of rule p below.

$$\alpha X \beta \# \$ \Rightarrow_{p1} \alpha p X \beta \# p' \$ \Rightarrow_{p2} \alpha p \beta \# p'' p' \$ \Rightarrow_{p3} \alpha b p \beta \# p''' p'' p' \$ \\ \Rightarrow_{p4} \alpha p b Y \beta \# p'' p' \$ \Rightarrow_{p5} \alpha b Y \beta \# p''' \$ \Rightarrow_{p6} \alpha b Y \beta \# \$.$$

Simulation of $f : AB \to \lambda$: Let the string $\alpha AB\beta\#\$$ be derived from the axiom $S\#\$$, with $A, B \in N''$ and $\alpha, \beta \in (N'' \cup T)^*$, corresponding to a derivation $S \Rightarrow^* \alpha AB\beta$ in G. As the first rules of the matrices $f2$ and $f3$ involve the rule marker f, we have to start from $f1$ and on applying it to the string, we obtain $\alpha fAB\beta\#f'\$$. At this point, if we apply $f3$, we get back to the starting point. Even if f was introduced in front of some other occurrence of A (e.g., $\alpha fA\delta AB\beta\#f'\$$) by applying $f1$, then the only applicable matrix is $f3$, which takes us to the starting point again. Hence to proceed further, the matrix $f2$ is applied, which in turn yields $\alpha f\beta\#f'\$$. On applying $f3$, the resultant string is the intended one, $\alpha\beta\#\$$.

$$\alpha AB\beta\#\$ \Rightarrow_{f1} \alpha fAB\beta\#f'\$ \Rightarrow_{f2} \alpha f\beta\#f'\$ \Rightarrow_{f3} \alpha\beta\#\$.$$

The marker deletion matrix $[(\#, \$, \lambda)_{del}, (\lambda, \#, \lambda)_{del}]$ is applied at the end of the derivation of words. Hence on starting at S and by repeatedly applying the simulations of the rules p, q, f, g, h, and the marker deletion rule we eventually get $S \Longrightarrow_* w$. This proves that $L(G) \subseteq L(\Gamma)$.

To prove the reverse relation ($L(\Gamma) \subseteq L(G)$), we observe the points discussed while proving the reverse part ($L(\Gamma) \subseteq L(G)$) of Theorem 5 and also that the rules of Γ are applied in groups and each group of rules corresponds to one of p, q, f, g, h. Details are similar to the previous proof and hence omitted. □

5 Simulating (meta-)linear grammars

Theorem 4 states that a matrix ins-del systems of size (3; 1, 1, 1; 1, 0, 0) can describe *RE*. If we further desire to have a one-sided context for insertion, then a matrix ins-del system of the desired size can simulate linear and metalinear grammars, as we will show now. However, whether or not one can simulate general context-free grammars with MAT(3; 1, 1, 0; 1, 0, 0) is left open. However, we believe that this type of study comes along in a quite natural way due to the special (linear) structure of the context-free rules in SGNF grammars.

Theorem 7 LIN$\subsetneq$MAT(3; 1, 1, 0; 1, 0, 0) $\cap$ MAT(3; 1, 0, 1; 1, 0, 0)

Proof Consider a linear grammar $G = (N, T, S, P)$. The rules from P in G are supposed to be labelled injectively with labels from the set $[1\ldots|P|]$. We construct a matrix insertion-deletion system $\Gamma = (V, T, \{S\}, M)$ where the alphabet of Γ is $V = N \cup T \cup [1\ldots|P|]$. The set of matrices of M of Γ is defined as follows.

We simulate the rule $p : X \to aY$ by the following two ins-del matrices:

$$p1 = [(X, p, \lambda)_{ins}, (\lambda, X, \lambda)_{del}]$$
$$p2 = [(p, Y, \lambda)_{ins}, (p, a, \lambda)_{ins}, (\lambda, p, \lambda)_{del}]$$

Similarly, we simulate the rule q: $X \to Ya$ by the following ins-del matrices:

$$q1 = [(X, q, \lambda)_{ins}, (\lambda, X, \lambda)_{del}]$$
$$q2 = [(q, a, \lambda)_{ins}, (q, Y, \lambda)_{ins}, (\lambda, q, \lambda)_{del}]$$

We simulate the rule f: $X \to a$ by the matrix $f = [(X, a, \lambda)_{ins}, (\lambda, X, \lambda)_{del}]$. The working of the matrices above is rather simple and straightforward. Observe the use of rule markers for simulating rules p and q. As there is never more than one occurrence of a nonterminal in the sentential form of G, repeated applications of $x1$ are impossible, and as $x2$ deletes the rule marker again, also repeated applications of $x2$ are impossible. So, an application of matrix $x1$ has to be immediately followed by an application of matrix $x2$, which corresponds altogether to an application of rule x, where x is of type p or q as shown above. Hence, $L(\Gamma) = L(G)$. The second part of the theorem (LIN$\subsetneq$MAT(3; 1, 0, 1; 1, 0, 0)) follows from Corollary 2 and the strictness of the inclusion follows from Example 1. □

Theorem 8 LIN$\subsetneq$MAT(2; 2, 1, 0; 1, 0, 0) $\cap$ MAT(2; 2, 0, 1; 1, 0, 0)

Proof We only give the matrix rules that work similar to the rules of Theorem 7.

Rules of the form $p : X \to aY$ are simulated by the following ins-del matrices:

$$p1 = [(X, p, \lambda)_{ins}, (\lambda, X, \lambda)_{del}]$$
$$p2 = [(p, aY, \lambda)_{ins}, (\lambda, p, \lambda)_{del}]$$

and rules of the form q: $X \to Ya$ by the following ins-del matrices:

$$q1 = [(X, q, \lambda)_{ins}, (\lambda, X, \lambda)_{del}]$$
$$q2 = [(q, Ya, \lambda)_{ins}, (\lambda, q, \lambda)_{del}]$$

We simulate the rule f: $X \to a$ by the matrix $f = [(X, a, \lambda)_{ins}, (\lambda, X, \lambda)_{del}]$. The second part of the theorem (LIN$\subsetneq$MAT(2; 2, 0, 1; 1, 0, 0)) follows from Corollary 2 and the strictness of the inclusion follows from Example 1. □

Theorem 9 MLIN$\subsetneq$MAT(3; 1, 1, 0; 1, 0, 0) $\cap$MAT(3; 1, 0, 1; 1, 0, 0)

Proof We assume that the metalinear language $L \subseteq T^*$ is described by a context-free grammar $G = (N, T, S, P)$ in the form described in Proposition 1. Taking over the notations of this proposition, L is the union of the concatenation of k linear languages $L(G_1^j), \ldots, L(G_k^j)$, with

$G_i^j = (N_i^j, T, S_i^j, P_i^j)$ for $j = 1, \dots, n$ and $i = 1, \dots, k$. Moreover, $N = \bigcup_{j=1}^n \bigcup_{i=0}^k N_i^j$, where $N_0^j = \{S^j, (S_2^j)', \dots, (S_k^j)'\}$. If $k = 1$, then $L(G) \in \mathrm{LIN}$ and the theorem follows from Theorem 7. Henceforth, we assume $k \geq 2$.

We now formally construct a matrix ins-del system $\Gamma = (V, \Sigma, \{S_1^j(S_2^j)' \mid j = 1, \dots, n\}, R)$ for G. For $1 \leq i \leq k$, let V_i^j be the alphabet resulting from the construction of matrix ins-del system Γ_i^j for G_i^j according to Theorem 7. Let $V = \bigcup_{j=1}^n \bigcup_{i=1}^k \left(V_i^j \cup \{(S_i^j)'\} \cup \{t_i^j, (t_i^j)'\}\right)$. For all i, $1 \leq i \leq k-1$, starting with the axiom $S_i^j(S_{i+1}^j)'$ (for $i = k$, the axiom is S_k^j), all strings of $L(G_i^j)$ are derived from S_i^j similar to Theorem 7. We present the simulation of rules of P_i^j in the following.

For $1 \leq i \leq k$, a rule of the form $p : X \to aY$ in P_i^j is simulated by the following set of matrices

$$p.1 = [(X, p, \lambda)_{ins}, (\lambda, X, \lambda)_{del}]$$
$$p.2 = [(p, Y, \lambda)_{ins}, (p, a, \lambda)_{ins}, (\lambda, p, \lambda)_{del}]$$

For $1 \leq i \leq k$, a rule of the form $q : X \to Ya$ in P_i^j is simulated by the following set of matrices

$$q.1 = [(X, q, \lambda)_{ins}, (\lambda, X, \lambda)_{del}]$$
$$q.2 = [(q, a, \lambda)_{ins}, (q_i, Y, \lambda)_{ins}, (\lambda, q_i, \lambda)_{del}]$$

For $1 \leq i \leq k-2$, a rule of the form $f : X \to a$ in P_i^j is simulated as follows:

$$f.1 = [(X, f, \lambda)_{ins}, (f, a, \lambda)_{ins}, (\lambda, X, \lambda)_{del}]$$
$$f.2 = [(\lambda, f, \lambda)_{del}, ((S_{i+1}^j)', (S_{i+2}^j)', \lambda)_{ins}, ((S_{i+1}^j)', f', \lambda)_{ins}]$$
$$f.3 = [(\lambda, f', \lambda)_{del}, ((S_{i+1}^j)', f'', \lambda)_{ins}, ((S_{i+1}^j)', S_{i+1}^j, \lambda)_{ins}]$$
$$f.4 = [(\lambda, f'', \lambda)_{del}, (\lambda, (S_{i+1}^j)', \lambda)_{del}]$$

A rule of the form $f : X \to a$ in P_{k-1}^j is simulated as follows:

$$f.1 = [(X, f, \lambda)_{ins}, (f, a, \lambda)_{ins}, (\lambda, X, \lambda)_{del}]$$
$$f.2 = [(\lambda, f, \lambda)_{del}, ((S_k^j)', S_k^j, \lambda)_{ins}, ((S_k^j)', f' \lambda)_{ins}]$$
$$f.3 = [(\lambda, f', \lambda)_{del}, (\lambda, (S_k^j)', \lambda)_{del}]$$

A rule of the form $f : X \to a$ in P_k^j is simulated as follows:

$$f.1 = [(X, f, \lambda)_{ins}, (f, a, \lambda)_{ins}, (\lambda, X, \lambda)_{del}]$$
$$f.2 = [(\lambda, f, \lambda)_{del}]$$

We now prove that $L(G) \subseteq L(\Gamma)$. For simplicity, we omit the index j indicating the choice of the specific grammar in the following, but we add the index i to the rule labels for clarity to indicate the number of the grammar within the row of concatenated grammars that we simulate. Consider a sentential form $w_1 \cdots w_{i-1} \alpha X \beta S'_{i+1} = w' \alpha X \beta S'_{i+1}$ (by induction), where $w_1 \in L(G_1)$, ..., $w_{i-1} \in L(G_{i-1})$, and $\alpha, \beta \in T^*$ such that the sentential form $\alpha X \beta$ is derivable in G_i, starting from S_i.

Simulation of $p_i : X \to aY$ Applying now $p_i.1$ to the string $w' \alpha X \beta S'_{i+1}$ yields $w' \alpha p_i \beta S'_{i+1}$. The matrix $p_i.1$ is applicable exactly once, as there is at most one nonterminal in a linear grammar. Applying now $p_i.2$ to the string $w' \alpha p_i \beta S'_{i+1}$ yields $w' \alpha a Y \beta S'_{i+1}$ as intended.

Simulation of $q_i : X \to Ya$ The working is similar to p_i rules.

Simulation of $f_i : X \to a$ The simulating f_i matrices actually do more than only simulating f_i. It simulates $X \to a$ as well as the transition rule (1) $S'_{i+1} \to S_{i+1} S'_{i+2}$ for $1 \leq i \leq k-2$ and (2) $S'_k \to S_k$.

Consider the rule $f_i : X \to a$ from P_i. Applying $f_i.1$ to the string $w' \alpha X \beta S'_{i+1}$ results in $w' \alpha f_i a \beta S'_{i+1}$ for any $1 \leq i < k$ (Case (1)). The only matrix applicable at this point is $f_i.2$ which yields (a) $w' \alpha a \beta S'_{i+1} f'_i S'_{i+2}$ for $1 \leq i \leq k-2$, or (b) $w' \alpha a \beta S'_k f'_{k-1} S_k$ for $i = k-1$. The only matrix that works with the newly introduced marker f'_i is $f_i.3$. On applying $f_i.3$, we get (a) $w' \alpha a \beta S'_{i+1} S_{i+1} f''_i S'_{i+2}$ for $1 \leq i \leq k-2$ or (b) $w' \alpha a \beta S_k$ for $i = k-1$. In the former case, the only applicable matrix that could tackle the new f''_i is $f_i.4$ whose application yields $w' \alpha a \beta S_{i+1} S'_{i+2}$ for $1 \leq i \leq k-2$. In Case (2), applying $f_k.1$ to the string $w' \alpha X \beta$ results in $w' \alpha f_k a \beta$, so that applying $f_k.2$ gives the desired string $w' \alpha a \beta$. The argument above shows that $L(G) \subseteq L(\Gamma)$.

Conversely, in Γ we have

$$w' \alpha X \beta S'_{i+1} \Longrightarrow_{f_i.1} w' \alpha f_i a \beta S'_{i+1} \Longrightarrow_{f_i.2} w' \alpha a \beta S'_{i+1} f'_i S'_{i+2} \Longrightarrow_{f_i.3} w' \alpha a \beta S'_{i+1} S_{i+1} f''_i S'_{i+2} \Longrightarrow_{f_i.4} w' \alpha a \beta S_{i+1} S'_{i+2}.$$

for any $1 \leq i < k-1$, and similarly in the two remaining cases. This derivation naturally corresponds to a derivation step in G. Notice that the special symbols f_i, f'_i, f''_i that are introduced and checked in the matrices $f_i.\ell$ prevent any other sequence of matrix applications from happening but the intended one, as explained above. So, once such a simulation is started, it cannot be interrupted by another simulation. The argument above shows that $L(G) \supseteq L(\Gamma)$.

Hence, $\mathrm{MLIN} \subseteq \mathrm{MAT}(3; 1, 1, 0; 1, 0, 0)$. As MLIN is closed under reversal, Corollary 2 yields $\mathrm{MLIN} \subsetneq \mathrm{MAT}(3; 1, 0, 1; 1, 0, 0)$. The claimed strictness of inclusion follows from Example 1 and hence the theorem follows. □

Notice that in the previous proof, a lot of technicalities were introduced by the fact that metalinear languages can be represented as a finite union of languages that are again finite concatenations of linear languages. However, since the implicit union construction performed in the previous proof can be easily adapted to the situation described in the next theorem, we will simplify our considerations by assuming that a metalinear language is the concatenation of k linear languages. For such languages, a simplified version of Proposition 1 can be shown (see Fernau et al. 2017c), which is used in the following theorem to avoid the mentioned complications.

Theorem 10 $\mathrm{MLIN} \subsetneq \mathrm{MAT}(2;2,1,0;1,0,0) \cap \mathrm{MAT}(2;2,0,1;1,0,0)$.

Proof We refer to Proposition 1 (in its simplified version as discussed above). We now formally construct a matrix ins-del system $\Gamma = (V, T, \{S_1 S_2'\}, R)$ for G. For $1 \le i \le k$, let V_i be the alphabet resulting from the construction of matrix ins-del system Γ_i for G_i according to Theorem 8. Let $V = \bigcup_{i=1}^{k} \left(V_i \cup \{S_i'\} \cup \{t_i, t_i', t_i''\}\right)$. Let P_i be the rule set of G_i. For all i, $1 \le i \le k-1$, starting with the axiom $S_i S_{i+1}'$ (for $i = k$, the axiom is S_k), all strings of $L(G_i)$ are derived from S_i similar to Theorem 8. We present the simulation of the P_i rules in details now.

For $1 \le i \le k$, a rule $p_i : X \to aY$ in P_i of G_i, is simulated by the matrix

$$p_i.1 = [(X, p_i, \lambda)_{ins}, (\lambda, X, \lambda)_{del}]$$
$$p_i.2 = [(p_i, aY, \lambda)_{ins}, (\lambda, p_i, \lambda)_{del}]$$

For $1 \le i \le k$, a rule $q_i : X \to Ya$ in P_i is simulated by the matrix

$$q_i.1 = [(X, q_i, \lambda)_{ins}, (\lambda, X, \lambda)_{del}]$$
$$q_i.2 = [(q_i, Ya, \lambda)_{ins}, (\lambda, q_i, \lambda)_{del}]$$

For $1 \le i \le k-2$, a rule $f_i : X \to a$ in P_i is simulated as follows:

$$f_i.1 = [(X, f_i a, \lambda)_{ins}, (\lambda, X, \lambda)_{del}]$$
$$f_i.2 = [(\lambda, f_i, \lambda)_{del}, (S_{i+1}', f_i' S_{i+2}', \lambda)_{ins}]$$
$$f_i.3 = [(\lambda, f_i', \lambda)_{del}, (S_{i+1}', S_{i+1} f_i'', \lambda)_{ins}]$$
$$f_i.4 = [(\lambda, f_i'', \lambda)_{del}, (\lambda, S_{i+1}', \lambda)_{del}]$$

The rule $f_{k-1} : X \to a$ in P_{k-1} is simulated as follows:

$$f_{k-1}.1 = [(X, f_{k-1} a, \lambda)_{ins}, (\lambda, X, \lambda)_{del}]$$
$$f_{k-1}.2 = [(\lambda, f_{k-1}, \lambda)_{del}, (S_k', f_{k-1}' S_k, \lambda)_{ins}]$$
$$f_{k-1}.3 = [(\lambda, f_{k-1}', \lambda)_{del}, (\lambda, S_k', \lambda)_{del}]$$

The rule $t_k : X \to a$ in P_k is simulated as follows:

$$f_k.1 = [(X, f_k a, \lambda)_{ins}, (\lambda, X, \lambda)_{del}]$$
$$f_k.2 = [(\lambda, f_k, \lambda)_{del}]$$

A proof of the correctness of this construction is completely analogous to the one given in the preceding theorem and hence omitted. Notice that the rules given in this case only summarize some of the rules of the previous construction.

As MLIN is closed under reversal, Corollary 2 yields the second part $\mathrm{MLIN} \subseteq \mathrm{MAT}(2;2,0,1;1,0,0)$. This completes the proof, as the claimed strictness of the inclusion immediately follows from Example 1. □

6 Summary of the results

In the Tables 1, 2 and 3, we summarize the generative power of matrix ins-del systems of all possible sizes with (1) $n = 1$, $m = 1$; (2) $n = 1$, $m = 2$; (3) $n = 2$, $m = 1$, respectively. This should show the state of the art in the area and help identify open problems in the area.

7 Semi-linearity and mild context-sensitivity

In this section, we discuss how matrix ins-del systems of small sizes can be used to study certain aspects of (non-) semilinear and mildly context-sensitive languages. This discussion provides some connections to aspects of our studies to formal linguistics. In order to facilitate following this type of discussion, we at least provide some background on semilinearity in the following.

A subset $A \subseteq \mathbb{N}^n$ is said to be *linear* if there are $v, v_1, \ldots, v_m \in \mathbb{N}^n$ such that

$$A = \{v + k_1 v_1 + k_2 v_2 + \cdots + k_m v_m \mid k_1, k_2, \ldots, k_m \in \mathbb{N}\}.$$

A subset $A \subseteq \mathbb{N}^n$ is said to be *semilinear* if it is a finite union of linear sets.

A permutation of the coordinates in $\mathbb{N}^n$ preserves semilinearity. Let Σ be a finite set of n elements. A *Parikh mapping* ψ from Σ^* into $\mathbb{N}^n$ is a mapping defined by first choosing an enumeration $a_1, \ldots, a_n$ of the elements of Σ and then defining inductively $\psi(\lambda) = (0, \ldots, 0)$, $\psi(a_i) = (\delta_{1,i}, \ldots, \delta_{n,i})$, where $\delta_{j,i} = 0$ if $i \ne j$ and $\delta_{j,i} = 1$ if $i = j$, and $\psi(au) = \psi(a) + \psi(u)$ for all $a \in \Sigma$, $u \in \Sigma^*$. Any two Parikh mappings from Σ^* into $\mathbb{N}^n$ differ only by a permutation of the coordinates of $\mathbb{N}^n$. Hence, the following concept is well-defined.

Let Σ be a finite set of n elements. A subset $A \subseteq \Sigma^*$ is said to be a *language with the semilinear property*, or *slip* language for short, if $\psi(A)$ is a semilinear subset of $\mathbb{N}^n$ for a Parikh mapping ψ of Σ^* into $\mathbb{N}^n$. Parikh's Theorem (see Parikh 1966) shows that all context-free languages are slip languages.

Kracht and Michaelis have shown in Michaelis and Kracht (1997) that several languages possess features that are non-semilinear. In particular, the example of Old-Georgian seems to be persuasive from a linguistic

Table 1 Overview on the power of matrix ins-del systems of size $(k; 1, i', i''; 1, j', j'')$

Size $(k; 1, i', i''; 1, j', j'')$ $i', i'', j', j'' \in \{0, 1\}$	k	Language family relation	Remarks
$(k;$ 1, 0, 0; 1, 0, 0)	1	$\subset$ *REG*	Krassovitskiy et al. (2008) and Verlan (2007)
$(k;$ 1, 0, 0; 1, 1, 0)	≥ 1	OPEN	
$(k;$ 1, 0, 0; 1, 0, 1)	≥ 1	OPEN	
$(k;$ 1, 0, 0; 1, 1, 1)	3	$=$ RE	Theorem 3
$(k;$ 1, 1, 0; 1, 0, 0)	3	$\supset$ MLIN	Theorem 9
$(k;$ 1, 1, 0; 1, 1, 0)	3	$=$ RE	See Petre and Verlan (2012)
$(k;$ 1, 1, 0; 1, 0, 1)	3	$=$ RE	See Petre and Verlan (2012)
$(k;$ 1, 1, 0; 1, 1, 1)	2	$=$ RE	Theorem 5
$(k;$ 1, 0, 1; 1, 0, 0)	3	$\supset$ MLIN	Theorem 9
$(k;$ 1, 0, 1; 1, 1, 0)	3	$=$ RE	Theorem 2
$(k;$ 1, 0, 1; 1, 0, 1)	3	$=$ RE	Theorem 2
$(k;$ 1, 0, 1; 1, 1, 1)	2	$=$ RE	Theorem 5
$(k;$ 1, 1, 1; 1, 0, 0)	3	$=$ RE	Theorem 4
$(k;$ 1, 1, 1; 1, 1, 0)	2	$=$ RE	Theorem 6
$(k;$ 1, 1, 1; 1, 0, 1)	2	$=$ RE	Theorem 6
$(k;$ 1, 1, 1; 1, 1, 1)	1	$=$ RE	See Takahara and Yokomori (2003)

Table 2 Overview on the power of matrix ins-del systems of size $(k; 1, i', i''; 2, j', j'')$

Size $(k; 1, i', i''; 2, j', j'')$ $i', i'', j', j'' \in \{0, 1\}$	k	Language family relation	Remarks
$(k;$ 1, 0, 0; 2, 0, 0)	1	$\subset$ *REG*	Krassovitskiy et al. (2008) and Verlan (2007)
$(k;$ 1, 0, 0; 2, 1, 0)	≥ 1	OPEN	
$(k;$ 1, 0, 0; 2, 0, 1)	≥ 1	OPEN	
$(k;$ 1, 0, 0; 2, 1, 1)	3	$=$ RE	Follows from Theorem 5
$(k;$ 1, 1, 0; 2, 0, 0)	2	$=$ RE	See Petre and Verlan (2012)
$(k;$ 1, 1, 0; 2, 1, 0)	2	$=$ RE	Follows from Petre and Verlan (2012)
$(k;$ 1, 1, 0; 2, 0, 1)	2	$=$ RE	Follows from Petre and Verlan (2012)
$(k;$ 1, 1, 0; 2, 1, 1)	2	$=$ RE	Follows from Theorem 5
$(k;$ 1, 0, 1; 2, 0, 0)	2	$=$ RE	Theorem 2
$(k;$ 1, 0, 1; 2, 1, 0)	2	$=$ RE	Follows from Theorem 2
$(k;$ 1, 0, 1; 2, 0, 1)	2	$=$ RE	Follows from Theorem 2
$(k;$ 1, 0, 1; 2, 1, 1)	2	$=$ RE	Follows from Theorem 5
$(k;$ 1, 1, 1; 2, 0, 0)	1	$=$ RE	See Păun et al. (1998)
$(k;$ 1, 1, 1; 2, 1, 0)	1	$=$ RE	Follows from Păun et al. (1998)
$(k;$ 1, 1, 1; 2, 0, 1)	1	$=$ RE	Follows from Păun et al. (1998)
$(k;$ 1, 1, 1; 2, 1, 1)	1	$=$ RE	See Takahara and Yokomori (2003)

background. This finding motivated us to look for non-semilinear languages that can be described by small matrix ins-del systems. Conversely, recall that many linguistically important languages can be also described by such systems, as proven in the Introduction.

Hopcroft and Pansiot (Lemma 2.8 in 1979) proved that vector addition systems with states (VASS) can describe the non-semilinear language $L' = \{w \in T^* \mid |w|_b + |w|_c \leq 2^{|w|_a}\}$. In Proposition 2, we show that a matrix ins-del system with size $\mathrm{MAT}(3; 1, 0, 0; 1, 0, 0)$ simulates the VASS $\mathcal{V}$ presented in Hopcroft and Pansiot (1979). Hence, it follows that $\mathrm{MAT}(3; 1, 0, 0; 1, 0, 0)$ contains non-semilinear languages. Before proving the proposition, we define VASS for the sake of keeping the presentation self-contained.

Table 3 Overview on the power of matrix ins-del systems of size $(k; 2, i', i''; 1, j', j'')$

Size $(k; 2, i', i''; 1, j', j'')$ $i', i'', j', j'' \in \{0, 1\}$	k	Language family relation	Remarks
(k; 2, 0, 0; 1, 0, 0)	1	⊂ *REG*	Krassovitskiy et al. (2008) and Verlan (2007)
(k; 2, 0, 0; 1, 1, 0)	2	=RE	See Petre and Verlan (2012)
(k; 2, 0, 0; 1, 0, 1)	2	=RE	Theorem 2
(k; 2, 0, 0; 1, 1, 1)	1	=RE	See Krassovitskiy et al. (2008)
(k; 2, 1, 0; 1, 0, 0)	2	⊃ MLIN	Theorem 10
(k; 2, 1, 0; 1, 1, 0)	2	=RE	Follows from Petre and Verlan (2012)
(k; 2, 1, 0; 1, 0, 1)	2	=RE	Follows from Theorem 2
(k; 2, 1, 0; 1, 1, 1)	1	=RE	Follows from Krassovitskiy et al. (2008)
(k; 2, 0, 1; 1, 0, 0)	2	⊃ MLIN	Theorem 10
(k; 2, 0, 1; 1, 1, 0)	2	=RE	Follows from Petre and Verlan (2012)
(k; 2, 0, 1; 1, 0, 1)	2	=RE	Follows from Theorem 2
(k; 2, 0, 1; 1, 1, 1)	1	=RE	Follows from Krassovitskiy et al. (2008)
(k; 2, 1, 1; 1, 0, 0)	3	=RE	Follows from Theorem 4
(k; 2, 1, 1; 1, 1, 0)	2	=RE	Follows from Theorem 6
(k; 2, 1, 1; 1, 0, 1)	2	=RE	Follows from Theorem 6
(k; 2, 1, 1; 1, 1, 1)	1	=RE	See Takahara and Yokomori (2003)

A *vector addition system with states of dimension n* is described by a tuple $(W, S, \Delta, x_0, p_0, F)$, where W is a finite subset of $\mathbb{Z}^n$, S is a finite set, modelling a state control, x_0 is the starting point and $p_0 \in S$ is the starting state, and $F \subseteq S$ collects the final states. The transitions are collected in a subset Δ of $S \times S \times W$, written like $p \to (q, u)$. We define $(p, v) \Rightarrow_{\mathcal{V}} (q, w)$, for $p, q \in S$ and $v, w \in \mathbb{N}^n$, if there is a transition $p \to (q, u)$ in $\mathcal{V}$ such that $w = v + u$. The *reachability set* of $\mathcal{V}$ (in a sense the multiset language described by $\mathcal{V}$ is $\{x \in \mathbb{N}^n \mid \exists q \in S : ((p_0, x_0) \Rightarrow^*_{\mathcal{V}} (q, x))\}$, where $\Rightarrow^*_{\mathcal{V}}$ is the reflexive transitive closure of $\Rightarrow_{\mathcal{V}}$.

Proposition 2 MAT$(3; 1, 0, 0; 1, 0, 0)$ *contains non-semilinear languages.*

Proof We translate the 3-dimensional vector addition system with states

$$\mathcal{V} = (\{0, 1\} \times \{-1, 0\} \times \{0, 2\}, \{p, q\}, \Delta, (0, 0, 1), p, \{p\})$$

with transitions $t_1 : p \to (p, (0, 1, -1))$, $t_2 : p \to (q, (0, 0, 0))$, $t_3 : q \to (q, (0, -1, 2))$, $t_4 : q \to (p, (1, 0, 0))$ as given by Hopcroft and Pansiot Hopcroft and Pansiot (1979) into some system $\Gamma = (\{A, A', B, B'B'', a, b, c\}, \{a, b, c\}, \{Ac\}, R)$ where R consists of the following matrices:

$m1 = [(\lambda, A, \lambda)_{del}, (\lambda, c, \lambda)_{del}, (\lambda, A', \lambda)_{ins}]$
$m2 = [(\lambda, A', \lambda)_{del}, (\lambda, b, \lambda)_{ins}, (\lambda, A, \lambda)_{ins}]$
$m3 = [(\lambda, A, \lambda)_{del}, (\lambda, B, \lambda)_{ins}]$,
$m4 = [(\lambda, B, \lambda)_{del}, (\lambda, b, \lambda)_{del}, (\lambda, B', \lambda)_{ins}]$
$m5 = [(\lambda, B', \lambda)_{del}, (\lambda, c, \lambda)_{ins}, (\lambda, B'', \lambda)_{ins}]$
$m6 = [(\lambda, B'', \lambda)_{del}, (\lambda, c, \lambda)_{ins}, (\lambda, B, \lambda)_{ins})$
$m7 = [(\lambda, B, \lambda)_{del}, (\lambda, a, \lambda)_{ins}, (\lambda, A, \lambda)_{ins}]$
$m8 = [(\lambda, A, \lambda)_{del}]$

Hence, the language $L(\Gamma)$ is in MAT$(3; 1, 0, 0; 1, 0, 0)$. The states p and q of $\mathcal{V}$ correspond to the nonterminals A and B of Γ, respectively. The matrices $m1$, $m2$ together simulate t_1, $m3$ simulates t_2, $m4$, $m5$, $m6$ together simulate t_3, $m7$ simulates t_4, and $m8$ allows termination. We consider the Parikh mapping ψ defined by $a \mapsto (1, 0, 0)$, $b \mapsto (0, 1, 0)$, $c \mapsto (0, 0, 1)$ and claim that the reachability set of $\mathcal{V}$ equals the Parikh image $\psi(L(\Gamma))$. To see this, we provide some more details on the simulation of $\mathcal{V}$ by Γ.

Starting with the axiom Ac corresponds to the starting state p and the starting point (0, 0, 1). On applying $m1$ and $m2$, we have $Ac \Rightarrow_{m1} A' \Rightarrow_{m2} b ш A$. At this point, we note that while going from Ac to $b ш A$, the number of a's is unaltered, the number of b's increases by one and the number of c's decreases by one and the nonterminal A is retained. Hence the first two matrices $m1$ and $m2$ of Γ simulate the transition $t_1 : q \to (q, (0, 1, -1))$ of $\mathcal{V}$ as claimed.

The matrix $m3$ clearly rewrites A into B ($b ш A \Rightarrow_{m3} b ш B$) which corresponds to the transition t_2 of $\mathcal{V}$. Next, on applying $m4$, $m5$ and $m6$, we have

$$b \sqcup B \Rightarrow_{m4} B' \Rightarrow_{m5} c \sqcup B'' \Rightarrow_{m4} c \sqcup c \sqcup B\,.$$

At this point, while moving from $b ш B$ to $c ш c ш B$, the number of a's is unaltered, the number of b's decreases by one and the number of c's increases by two and the nonterminal B is retained. Hence, the three matrices $m4$, $m5$ and $m6$ of Γ together simulate the transition $t_3 : q \to (q, (0, -1, 2))$ of $\mathcal{V}$.

Next on applying $m7$, we have $c ш c ш B \Rightarrow_{m7} c ш c ш a ш A$ which reflects that the number of a's increases by one, the numbers of b's and c's are unaltered and the nonterminal B is changed to A. Clearly, this matrix $m7$ simulates the transition $t_4 : q \rightarrow (p, (1, 0, 0))$ of $\mathcal{V}$. Lastly, the matrix m_8 only serves to terminate if the simulated system was in a certain state, namely p.

The explanations given so far should suffice to understand that the reachability set of $\mathcal{V}$ is a subset of $\psi(L(\Gamma))$. As the presence or absence of the nonterminals A, A', B, B', B'' uniquely identify a certain stage of the simulation of a transition of $\mathcal{V}$, it can be also seen that no malicious derivations are possible in Γ, i.e., $\psi(L(\Gamma))$ actually equals the reachability set of $\mathcal{V}$. Moreover, $L(\Gamma)$ is the (full) pre-image of the reachability set of $\mathcal{V}$ under ψ, as witnessed by writing, for instance, $c ш c ш B$ to denote the set of all strings derivable in a certain way in our discussion above.

Hence, Lemma 2.8 in Hopcroft and Pansiot (1979) shows that $L(\Gamma) = \{w \in \{a, b, c\}^* \mid |w|_b + |w|_c \leq 2^{|w|_a}\}$. As Γ has size (3; 1, 0, 0; 1, 0, 0), the claimed statement follows. □

These observations motivated us to study Parikh images of languages described by matrix ins-del systems, focusing on context-free insertion-deletion rules, see Fernau and Kuppusamy (2017). (Non-)semilinearity is not the only question that could be raised in connection with matrix ins-del systems of small weight like $\mathrm{MAT}(3; 1, 0, 0; 1, 0, 0)$; mildly context-sensitive languages should extend the context-free languages: Is $CF \subset \mathrm{MAT}(3; 1, 0, 0; 1, 0, 0)$ (similar with other matrix ins-del systems of small weight)? This is an interesting question of future research.

Also, notice that (in particular monotone) insertion-deletion systems with matrix control (also see Fernau and Kuppusamy 2017) can be simulated by restarting automata that jump back to the start of the string after simulating a certain number of insertion or deletion steps (this number is just the length of the matrix that is simulated). These devices are quite important for linguistic purposes, see Otto (2006), so that this provides a further natural link to language processing.

8 Conclusions and further research directions

In this paper, using matrix ins-del systems, we have obtained some (improved) computational completeness results and simulated linear and metalinear grammars with small resource needs. It is interesting to note that with the size used to simulate linear grammars, we were able to simulate all metalinear grammars, as well. We have also given a complete picture of the state of the art of the generative power of the matrix ins-del systems with size $(k; n, i', i''; m, j', j'')$ where $n, m \in \{1, 2\}$ with $n + m \leq 3$ and $i', i'', j', j'' \in \{0, 1\}$.

We now present some further concrete research directions below.

- Proving a non-trivial simulation result for all context-free grammars by context-free matrix ins-del systems with small size is left open. It is even not clear if we can obtain the regular closure of the linear languages, i.e., all languages that can be obtained from linear languages with the help of the regular operations union, concatenation and Kleene star. Notice that this family of languages is a strict subclass of the context-free languages (see Kutrib and Malcher 2007). We are currently working in this direction.
- It would be also interesting to explore closure properties for matrix ins-del systems of small sizes. For instance, it would be interesting to know if the family $\mathrm{MAT}(2; 2, 1, 0; 1, 0, 0)$ is closed under reversal. If this would be true, then $\mathrm{MAT}(2; 2, 1, 0; 1, 0, 0) = \mathrm{MAT}(2; 2, 0, 1; 1, 0, 0)$, which would also mean that the statement of theorems like Theorem 10 would simplify.
- Simulate matrix ins-del systems with other ins-del mechanisms, like graph-controlled ins-del systems, where also many non-trivial computational completeness results are known, with only small descriptional complexities, or vice-versa. One of the drawbacks in this approach is that the computational resources are counted quite differently, so that a small graph-controlled ins-del system would not necessarily lead to a small matrix ins-del system, nor vice versa. Supposedly, this situation would change if other types of descriptional complexity measures would be used. For instance, apart from Kuppusamy et al. (2016) and the literature quoted therein, we are not aware of any studies on the nonterminal complexity of controlled ins-del systems. In the case of controlled context-free grammars, it was then quite easy to transfer results between different forms of regulations, see Fernau (2003); Fernau et al. (2007); Freund and Păun (2001) and the papers quoted therein.
- Do matrix ins-del systems of small weight allow for efficient parsing? We are not aware of any research in this direction. Also this area seems to be largely neglected, although it is clear that this is of much importance if it comes to finally applying these generative devices in language processing.

- What type of relevant linguistic dependencies can be captured by matrix ins-del systems of small weight, beyond the example of cross dependencies we gave above in Example 1?
- The linear growth property (not the same as the semilinearity that we discussed in this paper) is related to some on-going discussion of natural language processing (see Kallmeyer 2010). We think it is interesting to discuss (and relate) this property to matrix ins-del systems.

Acknowledgements We are grateful to Serghei Verlan for his comments on our previous UCNC version, and also for sharing with us his Perl tool (simulating graph-controlled ins-del systems) that facilitates checking some of our results and examples. Some extension part of the work was undertaken during the second author's visit to University of Trier, Germany, in June 2016. Support of this visit by overhead money (from DFG grant FE 560/6-1) is gratefully acknowledged.

References

Alhazov A, Krassovitskiy A, Rogozhin Y, Verlan S (2011) P systems with minimal insertion and deletion. Theoret Comput Sci 412(1–2):136–144

Benne R (ed) (1993) RNA editing: the alteration of protein coding sequences of RNA. Series in molecular biology. Ellis Horwood, Chichester

Biegler F, Burrell MJ, Daley M (2007) Regulated RNA rewriting: modelling RNA editing with guided insertion. Theor Comput Sci 387(2):103–112

Dassow J, Păun Gh (1985) Further remarks on the complexity of regulated rewriting. Kybernetica 21(3):213–227

Dassow J, Păun G (1989) Regulated rewriting in formal language theory, volume 18 of EATCS monographs in theoretical computer science. Springer, Berlin

Fernau H (2003) Nonterminal complexity of programmed grammars. Theor Comput Sci 296:225–251

Fernau H, Kuppusamy L (2017) Parikh images of matrix ins-del systems. In: Gopal TV, Jäger G, Steila S (eds) Theory and applications of models of computation, TAMC (LNCS), vol 10185. Springer, pp 201–215

Freund R, Păun G (2001) On the number of non-terminal symbols in graph-controlled, programmed and matrix grammars. In: Margenstern M, Rogozhin Y (eds) Machines, computations, and universality; 3rd MCU (LNCS), vol 2055, pp 214–225

Fernau H, Freund R, Oswald M, Reinhardt K (2007) Refining the nonterminal complexity of graph-controlled, programmed, and matrix grammars. J Autom Lang Comb 12(1/2):117–138

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 (EPTCS), vol 31, pp 88–98

Fernau H, Kuppusamy L, Raman I (2016) Generative power of matrix insertion-deletion systems with context-free insertion or deletion. In: Amos M, Condon A (eds) Unconventional computation and natural computation conference, UCNC (LNCS), vol 9726. Springer, pp 35–48

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 (LNCS), vol 10316. Springer, pp 128–139

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

Fernau H, Kuppusamy L, Raman I (2017c) Properties of language classes between linear and context-free. Manuscript in preparation

Galiukschov BS (1981) Semicontextual grammars (in Russian). Mat. logica i mat. ling., Kalinin Univ., pp 38–50

Geffert V (1991a) How to generate languages using only two pairs of parentheses. J Inf Process Cybern EIK 27(5/6):303–315

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

Herman GT (1968) The halting problem of one state Turing machines with n-dimensional tape. Z Math Log Grundl Math 14:185–191

Hopcroft JE, Pansiot J-J (1979) On the reachability problem for 5-dimensional vector addition systems. Theor Comput Sci 8:135–159

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

Ivanov S, Verlan S (2017) Universality and computational completeness of controlled leftist insertion-deletion systems. Fundam Inform 155(1–2):163–185

Kallmeyer L (2010) On mildly context-sensitive non-linear rewriting. Res Lang Comput 8:341–363

Kari L (1991) On insertion and deletion in formal languages. PhD thesis, University of Turku, Finland

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

Krassovitskiy A, Rogozhin Y, Verlan S (2008) Further results on insertion-deletion systems with one-sided contexts. In: Martín-Vide C, Otto F, Fernau H (eds) Language and automata theory and applications, second international conference, LATA (LNCS), vol 5196. Springer, pp 333–344

Kudlek M (1996) Small deterministic Turing machines. Theor Comput Sci 168(2):241–255

Kuppusamy L, Rama R (2003) On the power of tissue P systems with insertion and deletion rules. In: Pre-proc of workshop on membrane computing, volume 28 of Report RGML. Univ. Tarragona, Spain, pp 304–318

Kuppusamy L, Mahendran A (2016) Modelling DNA and RNA secondary structures using matrix insertion-deletion systems. Int J Appl Math Comput Sci 26(1):245–258

Kuppusamy L, Mahendran A, Krishna SN (2011) Matrix insertion-deletion systems for bio-molecular structures. In: Natarajan R, Ojo AK (eds) Distributed computing and internet technology—7th international conference, ICDCIT (LNCS), vol 6536. Springer, pp 301–312

Kuppusamy L, Raman I, Krithivasan K (2016) On succinct description of certain context-free languages by ins-del and matrix ins-del systems. Int J Found Comput Sci 27(7):775–786

Kuroda S-Y (1964) Classes of languages and linear-bounded automata. Inf Control (now Inf Comput) 7:207–223

Kutrib M, Malcher A (2007) Finite turns and the regular closure of linear context-free languages. Discret Appl Math 155(16):2152–2164

Marcus S (1969) Contextual grammars. Rev Roum Math Pures et Appl 14:1525–1534

Marcus M, Păun Gh (1990) Regulated Galiukschov semicontextual grammars. Kybernetika 26(4):316–326

Margenstern M, Păun Gh, Rogozhin Y, Verlan S (2005) Context-free insertion-deletion systems. Theor Comput Sci 330(2):339–348

Michaelis J, Kracht M (1997) Semilinearity as a syntactic invariant. In: Retoré C (ed) Logical aspects of computational linguistics, first international conference, LACL'96 (LNCS), vol 1328. Springer, pp 329–345

Neary T (2017) 2-state 2-symbol Turing machines with periodic support produce regular sets. In: Pighizzini G, Câmpeanu C (eds) Descriptional complexity of formal systems—19th IFIP WG 1.02 international conference, DCFS (LNCS), vol 10316. Springer, pp 274–286

Neary T, Woods D (2012) The complexity of small universal Turing machines: A survey. In: Bieliková M, Friedrich G, Gottlob G, Katzenbeisser S, Turán Gy (eds) SOFSEM 2012: theory and practice of computer science—38th conference on current trends in theory and practice of computer science (LNCS), vol 7147. Springer, pp 385–405

Otto F (2006) Restarting automata. In: Ésik Z, Martín-Vide C, Mitrana V (eds) Recent advances in formal languages and applications. Studies in computational intelligence, vol 25. Springer, pp 269–303

Parikh RJ (1966) On context-free languages. J ACM 13(4):570–581

Păun Gh (1984) Six nonterminals are enough for generating each r.e. language by a matrix grammar. Int J Comput Math 15(1–4):23–37

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

Penttonen M (1974) One-sided and two-sided context in formal grammars. Inf Control (now Inf Comput) 25:371–392

Petre I, Verlan S (2012) Matrix insertion-deletion systems. Theor Comput Sci 456:80–88

Salomaa AK (1973) Formal languages. Academic Press, Cambridge

Shannon CE (1956) A universal Turing machine with two internal states. In: Shannon C E, McCarthy J (eds) Automata studies. Annals of mathematics studies, vol 34. Princeton University Press, pp 157–165

Stabler E (2004) Varieties of crossing dependencies: structure dependence and mild context sensitivity. Cognit Sci 28:699–720

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 Mold 18(2):210–245