Universal insertion grammars of size two

Sergey Verlan, Henning Fernau, Lakshmanan Kuppusamy

Universal insertion grammars of size two

Sergey Verlan[a,*], Henning Fernau[b], Lakshmanan Kuppusamy[c]

[a] *Univ Paris Est Creteil, LACL, F-94010 Creteil, France*
[b] *Fachbereich 4 – Abteilung Informatikwissenschaften, CIRT, Universität Trier, D-54286 Trier, Germany*
[c] *School of Computer Science and Engineering, VIT, Vellore-632 014, India*

Abstract

In this paper, we show that pure insertion grammars of size 2 (*i.e.*, inserting two symbols in a left and right context, each consisting of two symbols) can characterize all recursively enumerable languages. This is achieved by either applying an inverse morphism and a weak coding, or a left (right) quotient with a regular $LOC(2)$ language, or an intersection with a $LOC(2)$ language and a weak coding. The obtained results improve the descriptional complexity of insertion grammars and complete the picture of known results on insertion-deletion systems that are motivated from the DNA computing area.

Keywords: insertion grammars, insertion systems, recursively enumerable sets, homomorphic representation.

1. Introduction

An insertion grammar is a pure grammar (*i.e.*, there are no non-terminals as opposed to terminal symbols) having only rules of form $uv \to uxv$. Such grammars originated in [6]; they are inspired by Marcus contextual grammars [17, 26] used in linguistics. Another motivation for their study comes from the area of biology. As pointed out in [27], the process of mismatched annealing of DNA strands can be seen as an insertion or a deletion of a string in a specified context. A similar process happens in the case of RNA editing [2], where the uracil base U is inserted or deleted in some left context, as well as in the case of CRISPR-Cas9 technology that uses insertion and deletion to edit the genome [28, 1]. These observations led to the intense study of *insertion-deletion* systems (considering insertion and deletion operations together) in the framework of DNA computing [13, 29, 11, 18, 19, 30, 31].

There are several related models using a similar principle of insertion or deletion of a string in a specified context. We cite guided-insertion systems [3] used to model RNA editing, leftist grammars [20] used to model accessibility problems in protection systems, restarting automata [10] used to model the analysis by reduction and the insertion operation from [8] introduced as a generalization of the concatenation (and which corresponds to a context-free insertion grammar).

Since an insertion grammar is a pure grammar, an additional squeezing mechanism must be used in order to obtain a final language, as otherwise the described language class will have poor closure properties. Usually some restricted transducers are used for this purpose. Some standard examples are (1) the intersection with the free monoid over a terminal alphabet (used traditionally for Chomsky grammars), (2) projection composed with an inverse morphism, (3) left/right quotient by a regular language, (4) projection composed with intersection with a regular language. In the area of insertion grammars, variant (2) is mostly used, because (1) may only give at most context-sensitive languages [27].

The size of the insertion grammar is naturally defined by the triple (n, m, m'), where n (resp. m, m') is the maximal length of the inserted string (resp. left context, right context). If all parameters coincide, we

*Corresponding author
Email addresses: verlan@u-pec.fr (Sergey Verlan), fernau@uni-trier.de (Henning Fernau), klakshma@vit.ac.in (Lakshmanan Kuppusamy)

just say that the grammar is of *size* n. In [27], it was shown for the first time that insertion grammars of size (4,7,6) together with the squeezing mechanism (2) as described above generate all recursively enumerable languages. This result was improved in [21], where insertion grammars of size (3,5,4) are shown to be sufficient for the same task. Continuing with this race, papers [23, 12] show that insertion grammars of size 3 generate all recursively enumerable languages with squeezing mechanisms (2), (3) and (4). In [24, 22] the squeezing mechanism (4) was thoroughly investigated showing that the result above also holds when some restricted classes of regular languages are used.

Further decrease in size of rules was achieved only by using additional control mechanisms, like graph-control for the size 2 [14, 15] and matrix control [25, 16] for the size (1,2,2) [4].

In this paper, we show that it is possible to generate all recursively enumerable languages using insertion grammars of size 2 with squeezing mechanisms (2), (3) and (4). This settles the question of the computational power of this model when all three parameters are the same, as insertion grammars of size $(n, 1, 1)$ are shown to be context-free [27]. This also proves a remarkable jump when combined with squeezing mechanisms as described above, because the context-free languages are closed under these operations. Hence, while insertion grammars of size 1 with squeezing mechanisms (2), (3) and (4) stay within the context-free languages, size 2 grammars jump up to all recursively enumerable languages. The proof of the main result is greatly simplified by the introduced notion of an independent rule set that allows to formalize the conditions when the application of two insertion rules in a derivation is independent and can be done in any order.

2. Definitions

We assume the reader is familiar with the standard concepts used in formal languages theory. We recall some of them in order to fix the notations. Given an alphabet (a finite set) V, let V^* denote the set of all strings over V, *i.e.*, the free monoid generated by V; the operator symbol $\cdot$ (for concatenation) is mostly omitted. For a string $x \in V^*$, we denote the length of x by $|x|$ and the empty string is written as λ. If not explicitly stated otherwise, the notion of a morphism refers to a (homo)morphism mapping from the free monoid over some alphabet to the free monoid over some (other) alphabet. A weak coding is a special morphism $h : V^* \to W^*$, where $W \subseteq V$ and the restriction of h to W is the identity, and $h(x) = \lambda$ for $x \in V \setminus W$.

The set of prefixes of a string s will be denoted by $\mathrm{Pref}(s)$ and the set of suffixes as $\mathrm{Suff}(s)$. In this paper we consider non-empty prefixes and suffixes, *i.e.*, $\lambda \notin \mathrm{Pref}(s), \lambda \notin \mathrm{Suff}(s)$ for any string s. A proper substring of a string $s = a_1 \dots a_n$ is any sequence of consecutive letters s that is not equal to s: $a_i \dots a_j$, $1 \leq i \leq j \leq n$, with $j - i + 1 < n$. An *internal* substring of a string w is any substring of w that is not prefix or suffix of w.

2.1. Special Geffert Normal Form

A type-0 grammar $G = (N, T, S, P)$ is said to be in *Geffert normal form* [7] if the set of non-terminals N is defined as $N = \{S, A, B, C, D\}$, T is an alphabet (of terminal symbols) and P only contains context-free rules $S \to uSv$ with $u \in \{A, C\}^+$ and $v \in (T \cup \{B, D\})^+$, as well as $S \to uv$ whenever $S \to uSv \in P$, and two (non-context-free) erasing rules $AB \to \lambda$ and $CD \to \lambda$.

We remark that, according to [7], the generation of a string using a grammar in this normal form is performed in three stages. During the first two stages, only context-free rules $S \to uSv$ can be applied; this follows from the fact that $u \in \{A, C\}^+$ and $v \in (\{B, D\} \cup T)^+$. Moreover, the terminal symbols are generated first, so during the first stage, the string can be described by the regular expression $(A|C)^*ST^*$, which changes to $(A|C)^*S(B|D)^*T^*$ during the second stage. Assuming a terminal derivation, switching back to the first stage is not possible. The second stage is finalized by applying a rule of the form $S \to uv$. During the third stage, only non-context-free rules can be applied, because the symbol S is no longer present in the string. Moreover, in each step only one such rule can be applied. Note that the symbols A, B, C, D are treated like terminals during the first two stages and so, each rule $S \to uSv$ is, in a sense, "linear". We would also like to remark that according to the construction from Theorem 1 from [7] it is possible that $u = \lambda$ for some rules, however, v can never be the empty word.

Throughout this paper, we will use the *special Geffert normal form* (SGNF) [5, 9, 25]. This normal form is obtained by transforming "linear" rules from Geffert normal form to a set of "left- and right-linear" rules using a standard approach, see [25] for more details. Additionally, for commodity reasons, the terminals are generated at the left side of the string, which means by the remark above that "linear" rules from Geffert normal form that have to be transformed are of the form $S \to uSv$ where $v = \lambda$ for some rules, while u is never the empty word (†). Hence, for a grammar in SGNF, it is possible to split the set of non-terminals into disjoint sets as follows: $N = N_S \cup N_T \cup N_L \cup N_R$, where $N_T = N_{AC} \cup N_{BD}$, with $N_{AC} = \{A, C\}$, and $N_{BD} = \{B, D\}$, $N_S = \{S, S'\}$, N_L (resp. N_R) is the set of non-terminals appearing in the left-hand-side of "left-linear" (resp. "right-linear") rules except for S. We cite below some of the interesting properties of a grammar G in SGNF:

1. G has only two (non-context-free) erasing rules $AB \to \lambda$ and $CD \to \lambda$, and several "right-linear" and "left-linear" rules of one of the following forms; notice that due to Condition (†) above, we start with generating the non-empty part with "right-linear" rules and then might even skip the "left-linear" rules.
   - $X \to bY$, where $X \in N_R \cup \{S\}, b \in T \cup N_{AC}, Y \in N_R \cup N_L \cup N_S, X \neq Y$,
   - $X \to Yb$, where $X \in N_L, b \in N_{BD}, Y \in N_L \cup N_S, X \neq Y$,
   - $S' \to \lambda$.
2. We can assume that rules rewriting a symbol different from S are deterministic: for each $X \neq S$, there is a single rule $X \to ab \in P$. This property can be achieved, as we can choose different nonterminals in $N_R \cup N_L$ for each "linear" rule $S \to \alpha$ that is going to be simulated by "right-linear" and "left-linear" rules.
3. The computation in G is done in three stages described below.
   - Generating terminals at the left end of the string. The sentential form at this stage can be described by the regular expression $T^*(N \setminus N_T)(N_{BD})^*$. We remark that at this stage only one non-terminal from $N \setminus N_T$ is present in the string.
   - Generation of symbols from N_{AC}. The sentential form at this stage can be described by the regular expression $T^*(N_{AC})^*(N \setminus N_T)(N_{BD})^*$. As above, at this stage only one non-terminal from $N \setminus N_T$ is present in the string.
   - Erasing stage. The sentential form at this stage can be described by the regular expression $T^*(N_{AC})^*(AB|CD)(N_{BD})^*$. We remark that there is only one matching pair AB or CD present in the string.

   The transition from stage 1 to stage 2 is performed when the last rule producing a terminal is applied ($X \to bY$, $b \in T$). The transition from stage 2 to stage 3 is performed by applying the rule $S' \to \lambda$.

2.2. Insertion grammars

First, we define the notion of an insertion rule.

Definition 1. Let V be an alphabet. An *insertion rule* over V is the triple $r = (u, x, v)$, where $u, x, v \in V^*$.

Next, we define different types of relations between insertion rules.

An insertion rule $r = (u, x, v)$ *matches* the string w if uv is a substring of w. When $|uv| > 1$ this implies that an insertion (by using rule r) may happen inside w.

An insertion rule $r_1 = (u_1, x_1, v_1)$ matches an insertion rule $r_2 = (u_2, x_2, u_2)$ if $u_1 v_1 = u_2 v_2$. We also define a notion of a *perfect match* between two rules where additionally to the match it is required that $u_1 = u_2$ and $v_1 = v_2$. In a sense, matching rules compete for being applied on a string that they match.

Definition 2. An insertion rule $r = (u, x, v)$ *overlaps* string w (for a given alphabet V) if there exist strings $w', w'', u', v' \in V^*$ such that $w'ww'' = u'uvv'$ and $|u'u| > |w'|$ and $|vv'| > |w''|$.

The notion of overlap expresses the potential possibility to apply r modifying the string w. It is clear that the notion of overlap makes sense only for $|w| > 1$.

By definition, if $r = (u, x, v)$ overlaps w, then there is a factorisation $w'ww'' = u'uvv'$. Then, the condition $|u'u| > |w'|$ implies that w starts before u ends. Similarly, the condition $|vv'| > |w''|$ implies that v starts before w ends. Hence, there are 4 possible cases for positioning of w within $u'uvv'$ (they are also depicted on Fig. 1):

a. w starts before u starts and ends after v ends, hence uv is a substring of w, possibly $uv = w$,
b. w starts after u starts and ends before v ends, hence w is a proper substring of uv but it is not a proper substring of either u or v,
c. w starts before u starts and ends before v ends, hence $\mathrm{Suff}(w) \cap u\,\mathrm{Pref}(v) \neq \emptyset$,
d. w starts after u starts and ends after v ends, hence $\mathrm{Suff}(u)v \cap \mathrm{Pref}(w) \neq \emptyset$.

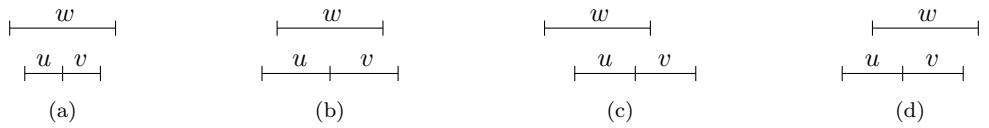

Figure 1: Four possible cases for string overlap: (a) uv is a substring of w, (b) w is a substring of uv, (c) $\mathrm{Suff}(w) \cap u\,\mathrm{Pref}(v) \neq \emptyset$, (d) $\mathrm{Suff}(u)v \cap \mathrm{Pref}(w) \neq \emptyset$.

Definition 3. A (pure) insertion grammar is given by a triple $G = (V, A, R)$, where

- V is an alphabet,
- $A \subseteq V^*$ is the set of axioms,
- R is a finite set of insertion rules over V.

We will also use the term *insertion system* as a synonym for an insertion grammar.

Definition 4. The derivation relation $\Rightarrow \subseteq V^* \times V^*$ of the insertion grammar $G = (V, A, R)$ is defined as follows:

$$w_1uvw_2 \Rightarrow w_1uxvw_2, \text{ if and only if } \exists r = (u, x, v) \in R$$

It can be easily deduced that an insertion rule (u, x, v) corresponds to the rewriting rule $uv \to uxv$. However, there is a small difference with traditional Chomsky grammars, where uv cannot be empty. There is no such restriction for insertion grammars. If $uv = \lambda$, then x can be inserted anywhere in the string, in particular, it can be appended to its left or right end.

We define the language generated by an insertion grammar $G = (V, A, R)$ as the reflexive and transitive closure of the set of axioms:

$$L(G) = \{w \in V^* \mid \exists x \in A : x \Rightarrow^* w\}.$$

We define the *size* of an insertion grammar $G = (V, A, R)$ as the triple (n, m, m'), where

$$n = \max_{(u,x,v)\in R} |x|\,, \quad m = \max_{(u,x,v)\in R} |u| \text{ and } m' = \max_{(u,x,v)\in R} |v|\,.$$

The family of insertion grammars of size (n, m, m') is denoted as $\mathrm{INS}(n, m, m')$.

We now introduce an important notion of the *independent* rule set. The main idea is to capture the conditions that allow rules to be applied in any order (independently of each other). In turn, this allows us to consider particular rule application orders that greatly simplify the proofs.

Definition 5. A set of insertion rules R is called *independent* if, for any two rules $r_1 = (u_1, x_1, v_1)$, $r_2 = (u_2, x_2, v_2)$ from R, one of the following two conditions holds:

- r_1 and r_2 perfectly match,
- r_1 does not overlap u_2v_2 and r_2 does not overlap u_1v_1.

The independence property says that for two insertion rules $r_1 = (u_1, x_1, v_1)$ and $r_2 = (u_2, x_2, v_2)$, r_1 (resp. r_2) can never insert x_1 (resp. x_2) inside the site u_2v_2 (resp. u_1v_1), except the case when they perfectly match (and then $u_1 = u_2$ and $v_1 = v_2$). Neglecting symmetric cases, this leaves us with only 3 possibilities for the relation between contexts and they are depicted in Fig. 2.

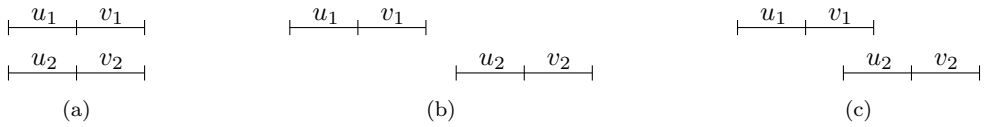

Figure 2: Three possible rule relations for an independent rule set (we recall that these conditions are symmetric with respect to rules): (a) rules perfectly match, (b) rules do not overlap and $\mathrm{Suff}(u_1v_1) \cap \mathrm{Pref}(u_2v_2) = \emptyset$, (c) rules do not overlap and $\mathrm{Pref}(u_2) \cap \mathrm{Suff}(v_1) \neq \emptyset$.

We now show that the independence property allows rules to be applied in any order.

Proposition 1. *Consider an insertion grammar $(V, \{w_1ww_2\}, R)$. If any rule $r = (u, x, v)$ from R does not overlap w and if w is not a proper substring of u or v, then*

- *w is preserved in any derivation,* i.e.*, no symbols will be inserted inside w;*
- *any derivation involving w_1 is independent of any derivation involving w_2,* i.e.*, if $w_1ww_2 \Rightarrow^* w_1'ww_2'$ then $w_1w \Rightarrow^* w_1'w$ and $ww_2 \Rightarrow^* ww_2'$.*

Proof. Since no rule overlaps w, this implies that no insertion may happen inside w. The second condition ensures that no rule contains w in its left or right context. This implies that w acts as a barrier for rules as they cannot check any letter beyond it. This implies the two claims. □

Proposition 2. *Consider an insertion grammar (V, A, R) having an independent rule set. Then for any rule $r = (u, x, v) \in R$ and any derivation $w_1uvw_2 \Rightarrow^* w_1'uvw_2'$ that does not apply r or any rule that perfectly matches it for the highlighted site uv, the evolution of w_1 and w_2 is independent,* i.e.*, $w_1u \Rightarrow^* w_1'u$ and $vw_2 \Rightarrow^* vw_2'$.*

Proof. This proposition is a consequence of Proposition 1, as the independence condition implies that any rule $r' = (u', x, v')$ different from r (and its perfectly matching variants) will not overlap u and v and also u and v will not be proper substrings of u' or v'. □

We would like to remark that for an insertion grammar with an independent rule set each substring uv corresponding to rule contexts marks a portion in the string that cannot be altered until the corresponding rule (or any rule perfectly matching it) is applied. When several such contexts are present in the string, it is possible to apply the corresponding rules in any order. This is formalized by the following theorem.

Theorem 1. *Let $G = (V, A, R)$ be an insertion grammar with an independent rule set. Then, for any derivation the rule application order is not important,* i.e.*, a rule can be applied to some part of the string at any time starting from the moment it becomes applicable and yielding the same result.*

Proof. Indeed, consider any derivation applying a rule $r = (u, x, v)$:

$$w \Rightarrow^* w_1uvw_2 \Rightarrow^* w_1'uvw_2' \Rightarrow_r w_1'uxvw_2' \Rightarrow^* w' . \tag{1}$$

Because R is independent, by Proposition 2 we have:

$$w_1u \Rightarrow^* w_1'u$$
$$vw_2 \Rightarrow^* vw_2'$$

Then the initial derivation can be reorganized as follows, applying r immediately after the substring uv appears for the first time in a string of the derivation sequence:

$$w \Rightarrow^* w_1uvw_2 \Rightarrow_r w_1uxvw_2 \Rightarrow^* w_1'uxvw_2' \Rightarrow^* w' .$$

Since derivation (1) does not impose any constraints on word w_1uvw_2, it may correspond to the first appearance of the substring uv or not. This allows us to conclude that rule r can be applied either immediately after substring uv is introduced, or deferred for the application at a later time. To conclude the proof we remark that because of rule independency the substring uv cannot be "broken" by an application of a rule inside it. □

Taking into account the last theorem, we remark that one can always choose the leftmost (resp. rightmost) site in the string for the rule application, this corresponds to the leftmost (resp. rightmost) derivation.

3. Mark and migration technique

For the proof of the main result, we use a variant of the *mark-and-migration* technique introduced in [27] and that is commonly used to obtain computational completeness results in the area of insertion systems. This technique is based on a simulation of a type-0 grammar. Since it is not possible to delete symbols in the derivation string, the main idea is to simulate the deletion of a non-terminal symbol X by adding a special marker \$ to its left (in [27] two markers # and \$ were used). Such a sequence can be later easily filtered by an inverse morphism or other squeezing mechanism. After the introduction of the \$ marker, the corresponding symbol X is treated as non-existing. We will also call a symbol X with a \$ to its left a *dead symbol*. By contrast, a symbol X without \$ to the left will be called *active*, or non-dead. When simulating context-free rules, dead symbols can be easily ignored by choosing insertion contexts that never allow an insertion between \$ and X. However, for context-sensitive rules like $AB \to \lambda$, it may happen that symbols A and B are separated by a sequence of dead symbols. To address this issue, the "migration" of B towards the left until it is placed right to its partner A is performed. The migration is performed by "jumping" over a dead symbol, *i.e.*, a substring $\$XB$ is transformed to $Bw_1\$Xw_2\$B$, with $w_1, w_2 \in (\$Z)^*$, with Z being an alphabet of some additional symbols. The iteration of this process allows us to obtain an adjacent pair of symbols A and B that can be further used for the simulation of an application of the corresponding rule $AB \to \lambda$, by turning the neighboring symbols A, B into dead symbols.

As shown in [27, 21, 23, 12], the migration of B over $\$X$ can be implemented in two phases: "jumping" over $\$X$ and marking the rightmost B as dead. However, for such an approach contexts of length at least 3 are required, as with contexts of length 2 it is not possible anymore to "jump" over the sequence $\$X$ (as the sequence $\$XB$ should be in the right context of some insertion rule). So, in order to be able to decrease the length of the context more phases and a "jump" inside the sequence $\$X$ are required. But this can pose integrity problems, as temporarily X is not a dead symbol anymore. The papers [14, 15, 4] solve these problems by using an additional control (graph or matrix) for the rule application that allows to apply only specific sequences of rules.

In this paper we show that by carefully choosing the contexts (of length 2) the temporary unmarking of a dead symbol cannot produce wrong derivations, so no additional control is required to achieve the claimed computational completeness. To achieve this, a barred copy of all non-terminal symbols (including the marker \$) is used. They are inserted in between \$ and X in order to split the "jump" and they also play the role of \$ symbol by not allowing X to be active, thus blocking wrong derivations. The "jump" of B over $\$X$ is performed in three phases. First a new symbol $\bar{B}_1$ is inserted between \$ and X and then B and X are marked as dead. Hence, the substring $\$XB$ is transformed to $\$\bar{B}_1\$X\$\nabla\$B$, where ∇ is a kind of separator symbol. Next, $\$\bar{B}_1\$X\$\nabla\$B$ is transformed to $B\$\nabla\$\bar{B}_2\$\bar{B}_1\$X\$\nabla\$B$, finishing the migration of B. Since such a migration introduces sequences of form $\$\bar{B}_i$, $1 \leq i \leq 2$ a symmetrical group of rules that allows to perform a migration over sequences $\$\bar{X}$ is added to the system. Finally, we remark that the construction from the current paper has an independent rule set, see the appendix for more details.

4. Main Results

Theorem 2. *For each recursively enumerable language L, there exists a morphism h, a weak coding g and a language $L_1 \in \mathrm{INS}(2,2,2)$ such that $L = g(h^{-1}(L_1))$.*

Proof. Let $G = (N, T, P, S)$ be a type-0 grammar in SGNF. We construct the insertion grammar $G_1 = (V, \{\emptyset\emptyset S\$S\}, P_1)$, where

$$V = N \cup T \cup \{K_{AB}, K_{CD}\} \cup \left\{X, \bar{X} \mid X \in \{\$, \nabla, B_1, D_1\}\right\} \cup \{\emptyset\}.$$

The set of rules P_1 is constructed as follows. Consider the following sets:

$$\begin{aligned}
\mathcal{L} &= T \cup \{A, C, \emptyset\},\\
\mathcal{N} &= N \cup \{K_{AB}, K_{CD}, B_1, D_1, \nabla\},\\
\bar{\mathcal{N}} &= \{\bar{B}_1, \bar{D}_1, \bar{\nabla}\},\\
\mathcal{D} &= \{\$X \mid X \in \mathcal{N}\} \cup \{\bar{\$}\bar{Y} \mid \bar{Y} \in \bar{\mathcal{N}}\},\\
\mathcal{F} &= \mathcal{L}^2 \cup \mathcal{D},\\
\mathcal{S} &= V \setminus \{\$, \bar{\$}\} = \mathcal{N} \cup \bar{\mathcal{N}} \cup T \cup \{\emptyset\}.
\end{aligned}$$

- For any rule $k : X \to bY \in P$ we add the following rules to P_1 (we recall that $b \in T \cup \{A, C\}$):

$$r_{k,1} = (\mathcal{L}, bY, X), \qquad r_{k,2} = (Y, \$, X)$$

- For any rule $k : X \to Yb \in P$ we add the following rules to P_1 (we recall that $b \in \{B, D\}$):

$$r_{k,3} = (\mathcal{L}, Y, X), \qquad r_{k,4} = (Y, b\$, X)$$

- Rule $S' \to \lambda$ is simulated by the following rule in P_1:

$$r_{S'} = (\mathcal{L}, \$, S').$$

- The move of symbols B and D to the left of the sequence $\$X$, $X \in \mathcal{N}$ is performed using the following rules in P_1 (we remark that rules $r_{D,i}$ are obtained from rules $r_{B,i}$ by replacing B by D):

$$\begin{array}{llll}
r_{B,1} = (\mathcal{S}\$, \bar{B}_1, XB) & r_{B,2} = (\bar{B}_1X, \nabla\$, B\$) & r_{D,1} = (\mathcal{S}\$, \bar{D}_1, XD) & r_{D,2} = (\bar{D}_1X, \nabla\$, D\$)\\
r_{B,3} = (\$\bar{B}_1, \$, X\nabla) & r_{B,4} = (\$X, \$, \nabla\$) & r_{D,3} = (\$\bar{D}_1, \$, X\nabla) & r_{D,4} = (\$X, \$, \nabla\$)\\
r_{B,5} = (\mathcal{F}, B\$, \$\bar{B}_1) & r_{B,6} = (\$\$, \bar{\$}, \bar{B}_1\$) & r_{D,5} = (\mathcal{F}, D\$, \$\bar{D}_1) & r_{D,6} = (\$\$, \bar{\$}, \bar{D}_1\$)\\
r_{B,7} = (B\$, \nabla, \$\bar{\$}) & r_{B,8} = (\nabla\$, \nabla, \bar{\$}\bar{B}_1) & r_{D,7} = (D\$, \nabla, \$\bar{\$}) & r_{D,8} = (\nabla\$, \nabla, \bar{\$}\bar{D}_1)
\end{array}$$

- The move of symbols B and D to the left of the sequence $\bar{\$}\bar{Y}$, $\bar{Y} \in \bar{\mathcal{N}}$ is performed using following rules in P_1 (we remark that rules $\bar{r}_{D,i}$ are obtained from rules $\bar{r}_{B,i}$ by replacing B by D):

$$\begin{array}{llll}
\bar{r}_{B,1} = (\mathcal{S}\bar{\$}, B_1, \bar{Y}B) & \bar{r}_{B,2} = (B_1\bar{Y}, \nabla\$, B\$) & \bar{r}_{D,1} = (\mathcal{S}\bar{\$}, D_1, \bar{Y}D) & \bar{r}_{D,2} = (D_1\bar{Y}, \nabla\$, D\$)\\
\bar{r}_{B,3} = (\bar{\$}B_1, \bar{\$}, \bar{Y}\nabla) & \bar{r}_{B,4} = (\bar{\$}\bar{Y}, \$, \nabla\$) & \bar{r}_{D,3} = (\bar{\$}D_1, \bar{\$}, \bar{Y}\nabla) & \bar{r}_{D,4} = (\bar{\$}\bar{Y}, \$, \nabla\$)\\
\bar{r}_{B,5} = (\mathcal{F}, B\$, \bar{\$}B_1) & \bar{r}_{B,6} = (\$\bar{\$}, \$, B_1\$) & \bar{r}_{D,5} = (\mathcal{F}, D\$, \bar{\$}D_1) & \bar{r}_{D,6} = (\$\bar{\$}, \$, D_1\$)\\
\bar{r}_{B,7} = (B\$, \nabla, \bar{\$}\$) & \bar{r}_{B,8} = (\nabla\bar{\$}, \bar{\nabla}, \$B_1) & \bar{r}_{D,7} = (D\$, \nabla, \bar{\$}\$) & \bar{r}_{D,8} = (\nabla\bar{\$}, \bar{\nabla}, \$D_1)
\end{array}$$

We remark that rules $r_{B,i}$ (resp. $r_{D,i}$), $1 \le i \le 8$ are very similar to rules $\bar{r}_{B,i}$ (resp. $\bar{r}_{D,i}$) — they are obtained one from another by removing the bar (if present) or adding the bar (if not present) to symbols inserted or used as contexts inside the sequence $\$X$ or $\bar{\$}\bar{Y}$.

- Rules $AB \to \lambda$ and $CD \to \lambda$ are simulated by the following rules in P_1:

$$r_{AB,1} = (\mathcal{L}A, K_{AB}\$, B\$) \qquad r_{AB,2} = (\mathcal{L}, \$, AK_{AB}) \qquad r_{AB,3} = (\$A, \$, K_{AB}),$$
$$r_{CD,1} = (\mathcal{L}C, K_{CD}\$, D\$) \qquad r_{CD,2} = (\mathcal{L}, \$, CK_{CD}) \qquad r_{CD,3} = (\$C, \$, K_{CD}).$$

Finally, let $h : \mathcal{S} \to V^*$ be the morphism defined by

$$h(x) = \begin{cases} \$x, & x \in \mathcal{N}, \\ \bar{\$}x, & x \in \bar{\mathcal{N}}, \\ x, & x \in T, \\ \emptyset, & x = \emptyset. \end{cases}$$

and $g : \mathcal{S} \to T$ be the weak coding defined by

$$g(x) = \begin{cases} x, & x \in T, \\ \lambda, & \text{otherwise.} \end{cases}$$

Now we show that $L(G) = g(h^{-1}(L(G_1)))$.

First we show the inclusion $L(G) \subseteq g(h^{-1}(L(G_1)))$. A sentential form $w = w_1 \dots w_n$ of G will be represented by a word of form $\emptyset\emptyset\mathcal{D}^* w_1 \mathcal{D}^+ w_2 \mathcal{D}^+ \dots w_n \mathcal{D}^+$ in G_1. In particular, for the axiom, we find $\emptyset\emptyset S\$S \in \emptyset\emptyset\mathcal{D}^* S \mathcal{D}^+$, as it represents the sentential form S of G. To simplify the presentation, we suppose that there exists a function c that, applied to a word $w = w_1 \dots w_n$, gives the set of all strings of form $\mathcal{D}^* w_1 \mathcal{D}^* w_2 \mathcal{D}^* \dots w_n \mathcal{D}^+$. Hence, the axiom of G_1, $\emptyset\emptyset S\$S$, belongs to $\emptyset\emptyset\, c(S)$. We remark that the rules from G_1 ensure that there is at least one symbol different from \$ and $\bar{\$}$ after the starting $\emptyset\emptyset$ sequence. We highlight this in what follows by writing the sentential form uv in G as $\emptyset\emptyset\, uc(v)$ in G_1.

Consider an arbitrary derivation in G. As pointed out above, this derivation can be split into three phases. During the first two phases only rules of type $X \to bY$ or $X \to Yb$ can be executed and during the third phase, only rules $AB \to \lambda$ and $CD \to \lambda$ can be performed. The transition between the second and the third phases is done by the application of the rule $S' \to \lambda$. Consider the part of the derivation corresponding to the first or second phase. Suppose that the rule $k : X \to bY$ is applied:

$$uXv \Rightarrow ubYv$$

Then in G_1 there is the following derivation:

$$\emptyset\emptyset\, uXc(v) \Rightarrow_{r_{k,1}} \emptyset\emptyset\, ubYXc(v) \Rightarrow_{r_{k,2}} \emptyset\emptyset\, ubY\$Xc(v)$$

We remark that the first derivation is possible as $\emptyset\emptyset\, u \subseteq \mathcal{L}^+$, because uXv is of the form $T^*(N_{AC})^*(N \setminus N_T)(N_{BD})^*$. Also, it is clear that $\emptyset\emptyset\, ubY\$Xc(v) \subseteq \emptyset\emptyset\, c(ubYv)$ that corresponds to $ubYv$ in G.

Now suppose that the rule $k : X \to Yb$ is applied in G:

$$uXv \Rightarrow uYbv$$

Then in G_1 there is the following derivation:

$$\emptyset\emptyset\, uXc(v) \Rightarrow_{r_{k,3}} \emptyset\emptyset\, uYXc(v) \Rightarrow_{r_{k,4}} \emptyset\emptyset\, uYb\$Xc(v)$$

As before, the first derivation is possible as $\emptyset\emptyset\, u \in \mathcal{L}^+$.

Let us consider now that the rule $S' \to \lambda$ is applied:

$$uS'v \Rightarrow uv$$

Then in G_1 there is the following derivation:

$$\emptyset\emptyset\, uS'c(v) \Rightarrow_{r_{S'}} \emptyset\emptyset\, u\$S'c(v)$$

Again, we remark that this derivation is possible as $\emptyset\emptyset\, u \in \mathcal{L}^+$.

Now consider the third phase of the derivation in G. We recall that during this phase exactly one of rules $AB \to \lambda$ or $CD \to \lambda$ is applicable. We will consider below only the first case, the other one being symmetrical. Hence, consider the derivation $uABv \Rightarrow uv$ in G. Suppose that corresponding symbols A and B from $\emptyset\emptyset\, uc(ABv)$ are adjacent to each other. Then there exists the following derivation in G':

$$\emptyset\emptyset\, uABc(v) \Rightarrow_{r_{AB,1}} \emptyset\emptyset\, uAK_{A,B}\$Bc(v) \Rightarrow_{r_{AB,2}} \emptyset\emptyset\, u\$AK_{AB}\$Bc(v) \Rightarrow_{r_{AB,3}} \emptyset\emptyset\, u\$A\$K_{AB}\$Bc(v)$$

We remark that the first two rules are applicable, because for a grammar in SGNF it holds that $u \in \mathcal{L}^+$, hence $\emptyset\emptyset\, u \in \mathcal{L}^+$ as well.

Now suppose that symbols A and B are not adjacent. In this case, consider the leftmost B that is not dead. It is clear that there are only dead symbols between A and that B. We will show below how it is possible to transfer B over a dead symbol $\$X$. We suppose that the two symbols ab to the left of $\$X$ correspond either to a dead symbol ($\$Y$ or $\bar{\$}\bar{Y}$) or to a pair of symbols from $\mathcal{L}$. This corresponds to $ab \in \mathcal{F}$. We will show below that the transformation we perform keeps this property.

$$\begin{aligned} ab\$XB &\Rightarrow_{r_{B,1}} ab\$\bar{B}_1XB \Rightarrow_{r_{B,2}} ab\$\bar{B}_1X\nabla\$B \Rightarrow_{r_{B,3}} ab\$\bar{B}_1\$X\nabla\$B \Rightarrow_{r_{B,4}} \\ &\Rightarrow_{r_{B,4}} ab\$\bar{B}_1\$X\$\nabla\$B \Rightarrow_{r_{B,5}} abB\$\$\bar{B}_1\$X\$\nabla\$B \Rightarrow_{r_{B,6}} abB\$\$\$\bar{\$}\bar{B}_1\$X\$\nabla\$B \Rightarrow_{r_{B,7}} \\ &\Rightarrow_{r_{B,7}} abB\$\nabla\$\$\bar{\$}\bar{B}_1\$X\$\nabla\$B \Rightarrow_{r_{B,8}} abB\$\nabla\$\nabla\$\bar{\$}\bar{B}_1\$X\$\nabla\$B \end{aligned}$$

We remark that in the last string all symbols to the right of the first B are dead, that ensures that for the migration of the next symbol B it will encounter the left context from $\mathcal{F}$.

By repeating the above process it is possible to migrate to the left all symbols B and D that are not dead. Then because the derivation in G is terminal, all these symbols will be marked as dead together with the corresponding symbols A and C. Hence, for any terminal derivation $S \Rightarrow^* u$ in G we obtain the derivation $\emptyset\emptyset S\$S \Rightarrow^* \emptyset\emptyset uv$, with $v \in \mathcal{D}^+$. Hence, $L(G) \subseteq g(h^{-1}L(G_1))$.

Now let us show the converse inclusion, $g(h^{-1}L(G_1)) \subseteq L(G)$.

First, we remark that all rules from G_1 are independent, see the appendix for more details. This implies that the rules can be applied at any moment as soon as the corresponding context appears in the string.

Next we remark that the computation in G_1 can be split into two phases: in the first phase only rules $r_{k,*}$ are applied and in the second phase only rules $r_{Z,*}, \bar{r}_{Z,*}$, $Z \in \{B, D, AB, CD\}$ are applied. Indeed, the initial configuration is $\emptyset\emptyset S\$S$, so no rule from the second group can be applied. The application of rules $r_{k,1}$ or $r_{k,3}$ leaves in the string a single occurrence of $\mathcal{L}ZX\$$, $Z, X \in N \setminus N_T$. This substring can only be matched by a rule $r_{k,2}$ or $r_{k,4}$. Because the rules are independent, we can immediately perform the corresponding rule. Because of the form of the rules, there are only symbols from $\mathcal{L}$ at the left of the leftmost (active) nonterminal from $N \setminus N_T$ and no symbols from $\mathcal{L}$ at the right of it. Hence, the sentential form can be described by the expression $\emptyset\emptyset T^*\{A, C\}^*(N \setminus N_T)w$, with $w \in (N \setminus N_{AB})^*$. Because of the rule independence, by Theorem 1, we can first apply again $r_{k,1}$ or $r_{k,3}$, etc. Hence, it is possible to reorganize the derivation and to execute first rules from the first group and after that the rule $r_{S'}$ and after that use only rules from the second group. This means that the first part of the derivation follows the following rule sequence: $(\bigcup_{k:X\to bY} r_{k,1}r_{k,2} \cup \bigcup_{k:X\to Yb} r_{k,3}r_{k,4})^* r_{S'}$. To conclude, we finally remark that the rule $r_{S'}$ should be applied in a successful derivation, otherwise the string cannot pass the filter h^{-1}.

Consider now the second phase. At the beginning of it, the current string is of the following form: $\emptyset\emptyset T^* N_{AC}^* \$S'(\$(N \setminus N_T)^* N_{BD})^* \$S$ (T and N_{AC} followed by a sequence containing dead nonterminals from $N \setminus N_T$ mixed with symbols from N_{BD}). We will show now how the migration is performed. For simplicity, let us suppose that the rightmost symbol of the N_{AC}^* part is equal to A and that the leftmost alive symbol from N_{BD} is B. Since it is not adjacent to A it should be migrated to the left.

Let us consider the migration of B over $\$X$, $X \in N \setminus N_T$. It is performed by rules $r_{B,*}$. Since the rules are independent, by Theorem 1, we chose to apply them in the rightmost manner. Denote by ab the two symbols at the left of $\$X$. If $b \in \{\$, \bar{\$}\}$ then no rule is applicable. So, we suppose that $b \in \mathcal{S}$ (which is initially true). We will also show that there is always a $\$$ to the right of B. So, we have the following

derivation

$$ab\$XB\$ \Rightarrow_{r_{B,1}} ab\$\bar{B}_1XB\$ \Rightarrow_{r_{B,2}} ab\$\bar{B}_1X\nabla\$B\$ \Rightarrow_{r_{B,3}} ab\$\bar{B}_1\$X\nabla\$B\$ \Rightarrow_{r_{B,4}} ab\$\bar{B}_1\$X\$\nabla\$B\$$$

At this moment if $ab \notin \mathcal{F}$ the derivation stops. So, we suppose that $ab \in \mathcal{F}$ (this is true in particular at the beginning of phase 2). Then the derivation continues as follows:

$$ab\$\bar{B}_1\$X\$\nabla\$B\$ \Rightarrow_{r_{B,5}} abB\$\$\bar{B}_1\$X\$\nabla\$B\$ \Rightarrow_{r_{B,6}} abB\$\$\bar{\$}\bar{B}_1\$X\$\nabla\$B\$ \Rightarrow_{r_{B,7}}$$
$$\Rightarrow_{r_{B,7}} abB\$\nabla\$\bar{\$}\bar{B}_1\$X\$\nabla\$B\$ \Rightarrow_{r_{B,8}} abB\$\nabla\$\nabla\$\bar{\$}\bar{B}_1\$X\$\nabla\$B\$$$

So, we have shown that $ab\$XB\$ \Rightarrow^* abB\$\nabla\$\nabla\$\bar{\$}\bar{B}_1\$X\$\nabla\$B\$$. We observe that all symbols at the right of B are dead and that there is a \$ to the right of first B.

A similar computation happens for the case $\bar{\$}\bar{Y}B$, as well as for the migration of symbol D.

After several migration steps, B becomes adjacent to the last alive A. At this point, only rule $r_{AB,1}$ is applicable, which yields the following sequence of applications ($a \in \mathcal{L}$):

$$aAB\$ \Rightarrow_{r_{AB,1}} aAK_{AB}\$B\$ \Rightarrow_{r_{AB,2}} a\$AK_{AB}\$B\$ \Rightarrow_{r_{AB,3}} a\$A\$K_{AB}\$B\$$$

This simulates one application of the rule $AB \to \lambda$ from G. We remark that we can choose to apply the last three rules in a row, because the set of all rules is independent. The case of the rule $CD \to \lambda$ is similar.

A successful derivation in G_1 implies that all symbols except terminals are dead. We remark that the sequence of rules in phase 1 always leaves one non-terminal that is not dead. Hence, according to the discussion above, in a successful derivation all rules from phase 1 need to be executed. Next, at the beginning of phase 2 there are always symbols from $\{A, B, C, D\}$ that are not dead. We remark that the application of rule sequences of type $r_{Z,*}, \bar{r}_{Z,*}$, $Z \in \{B, D\}$ does not decrease the number of dead symbols. Only sequences of rules of type $r_{AB,*}$ and $r_{CD,*}$ can decrease this number (by two). In order to apply these last rules, corresponding symbols B and D need to be migrated to the left and then matched to corresponding symbols A and C. Hence, the sequence of used rules is described by the language

$$\left(\bigcup_{k:X\to bY} r_{k,1}r_{k,2} \cup \bigcup_{k:X\to Yb} r_{k,3}r_{k,4} \right)^* r_{S'} \left((z_{M,1} \dots z_{M,8})^+ r_{PM,1}r_{PM,2}r_{PM,3} \right)^*,$$

where $z \in \{r, \bar{r}\}$, $PM \in \{AB, CD\}$. Then we can construct the derivation in G yielding to the same terminal string as follows (we recall that there is at most one non-terminal from $N \setminus \{A, B, C, D\}$ or a sequence AB or CD in any sentential form of G):

- for each rule $r_{k,1}$ or $r_{k,3}$, execute rule r_k in G,
- for the rule $r_{S'}$, execute the rule $S' \to \lambda$ in G,
- for the rule $r_{AB,1}$ (resp. $r_{CD,1}$) execute rule $AB \to \lambda$ (resp. $CD \to \lambda$) in G.

This concludes the proof. □

Corollary 1. *For each recursively enumerable language L, there exists a regular language $L_1 \in LOC(2)$, a weak coding g and a language $L_2 \in \mathrm{INS}(2, 2, 2)$ such that $L = g(L_1 \cap L_2)$.*

Proof. Theorem 2 shows how to construct an insertion grammar $G_1 = (V, A, R)$ that for any $x \in L$ will generate $\emptyset\emptyset xw$, $w \in \mathcal{D}^+$, where $\mathcal{D}$ is defined as above. We add to G_1 a rule $(x, \$, y)$, $x, y \in T \cup \{\emptyset\}$ and change the axiom to $\$\emptyset\emptyset S\$S$. Then, the corresponding grammar will generate $\$\emptyset\$\emptyset\$x_1 \dots \$x_n w$, if $x_1 \dots x_n \in L$. Consider $L_1 = (\mathcal{D} \cup \{\$x \mid x \in \mathcal{L}\})^*$. Clearly, $L_1 \in LOC(2)$. The intersection $L_1 \cap L(G_1)$ will keep only strings that have all symbols dead; this corresponds to a successful derivation. To conclude the proof, we take the same mapping g as in Theorem 2. □

Corollary 2. *For each recursively enumerable language L, there exists a regular language $L_1 \in LOC(2)$, and a language $L_2 \in \mathrm{INS}(2,2,2)$ such that $L = L_2/L_1$.*

Proof. Let $G = (N, T, S, P)$ be the grammar describing L. We can suppose that it is in SGNF. Theorem 2 shows how starting from G it is possible to construct an insertion grammar $G_1 = (V, A, R)$ that for any $x \in L$ will generate $\emptyset\emptyset xw$, $w \in \mathcal{D}^+$, where $\mathcal{D}$ is defined as above. Since w is already in $LOC(2)$ form, $L(G_1)/L_1 = \{\emptyset\emptyset x \mid x \in L\}$, where $L_1 = \mathcal{D}^*$. To eliminate the sequence $\emptyset\emptyset$, it is sufficient to observe that it is used as a rule context only during the first steps, when no terminals are yet generated. This means that we can replace the axiom $\emptyset\emptyset S\$S$ by a set of axioms where the two first terminals are already generated: $abS\$Sw_1\$Sw_2 \ldots \$Sw_n\$S$, where $S \Rightarrow^* abw_1 \ldots w_n$ in G and $a, b \in T$, $w_1, \ldots, w_n \in N$. Since G is in SGNF, all corresponding strings can be obtained by considering derivations in G where the number of applications of "right-linear" rules ($X \to bY$) is two. Since in any derivation there cannot be infinite sequences of "left-linear" rules, there is a finite number of such strings. To conclude, we shall also add all strings of length 1 and 0 from L to the set of axioms. Then, no sequence $\emptyset\emptyset$ is present anymore at the beginning of the string and this concludes the proof. □

Corollary 3. *For each recursively enumerable language L, there exists a regular language $L_1 \in LOC(2)$, and a language $L_2 \in \mathrm{INS}(2,2,2)$ such that $L = L_1 \backslash L_2$.*

The left quotient proof is a consequence of the previous corollary, as we can reverse all the rules and axioms.

5. Conclusions

In this paper, we have shown computational completeness of insertion grammars of size 2 enriched with different squeezing mechanisms. Since insertion grammars of size $(n, 1, 1)$ are known to be context-free [27], only cases $(1, 2, 2)$, (n, m, p), $1 \leq n \leq 2$, $0 \leq p \leq 1$, $m \geq 0$ as well as their symmetric variants remain to be investigated for computational completeness.

The proof of the result was greatly simplified using the concept of an independent rule set. We have checked that the rules given in Theorem 2 are independent and then by Theorem 1 we could use them in the order we preferred in order to achieve a simpler proof. We think that this notion can be very useful in the area of insertion grammars and insertion-deletion systems from DNA computing [31] and it can help obtain results improving the existing ones. Another challenging application of the independence property is related to the modelling of the CRISPR-Cas9 mechanism [1] used for genome editing and that is very close to insertion grammars. Ensuring the independence of the target sequences can lead to a better control of the iteration of the editing process.

References

[1] Adli, M., 2018. The CRISPR tool kit for genome editing and beyond. Nature Communications 9, 1–13.

[2] Benne, R. (Ed.), 1993. RNA Editing: The Alteration of Protein Coding Sequences of RNA. Series in Molecular Biology, Ellis Horwood, Chichester, UK.

[3] Biegler, F., Burrell, M.J., Daley, M., 2007. Regulated RNA rewriting: Modelling RNA editing with guided insertion. Theoretical Computer Science 387, 103–112.

[4] Fernau, H., Kuppusamy, L., Verlan, S., 2017. Universal matrix insertion grammars with small size, in: Patitz, M.J., Stannett, M. (Eds.), Unconventional Computation and Natural Computation - 16th International Conference, UCNC 2017, Fayetteville, AR, USA, June 5-9, 2017, Proceedings, Springer. pp. 182–193. URL: https://doi.org/10.1007/978-3-319-58187-3_14, doi:10.1007/978-3-319-58187-3_14.

[5] 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, pp. 88–98.

[6] Galiukschov, B., 1981. Semicontextual grammars. Matem. Logica i Matem. Lingvistika , 38–50 Tallin University, (in Russian).

[7] Geffert, V., 1991. Normal forms for phrase-structure grammars. RAIRO Informatique théorique et Applications / Theoretical Informatics and Applications 25, 473–498.

[8] Haussler, D., 1983. Insertion languages. Information Sciences 31, 77–89. doi:10.1016/0020-0255(83)90023-3.

[9] Ivanov, S., Verlan, S., 2015. Universality of graph-controlled leftist insertion-deletion systems with two states, in: Durand-Lose, J., Nagy, B. (Eds.), Machines, Computations, and Universality - 7th International Conference, MCU, Springer. pp. 79–93.
[10] Jančar, P., Mráz, F., Plátek, M., Vogel, J., 1995. Restarting automata, in: Reichel, H. (Ed.), Fundamentals of Computation Theory, FCT, pp. 283–292.
[11] Kari, L., Păun, Gh., Thierrin, G., Yu, S., 1999. At the crossroads of DNA computing and formal languages: Characterizing recursively enumerable languages using insertion-deletion systems, in: Rubin, H., Wood, D.H. (Eds.), DNA Based Computers III. AMS. volume 48 of *DIMACS Series in Discrete Mathematics and Theretical Computer Science*, pp. 329–338.
[12] Kari, L., Sosík, P., 2008. On the weight of universal insertion grammars. Theoretical Computer Science 396, 264–270.
[13] Kari, L., Thierrin, G., 1996. Contextual insertions/deletions and computability. Information and Computation 131, 47–61.
[14] Krassovitskiy, A., 2009. On the power of insertion P systems of small size, in: Proc. of Seventh Brainstorming Week on Membrane Computing, pp. 29–44.
[15] Krassovitskiy, A., 2011. On the power of small size insertion P systems. Int. J. Comput. Commun. Control 6, 266–277. URL: https://doi.org/10.15837/ijccc.2011.2.2175, doi:10.15837/ijccc.2011.2.2175.
[16] Kuppusamy, L., Mahendran, A., Krishna, S.N., 2011. Matrix insertion-deletion systems for bio-molecular structures, in: Natarajan, R., Ojo, A.K. (Eds.), Distributed Computing and Internet Technology - 7th International Conference, ICDCIT 2011, Bhubaneshwar, India, February 9-12, 2011. Proceedings, Springer. pp. 301–312. doi:10.1007/978-3-642-19056-8_23.
[17] Marcus, S., 1969. Contextual grammars, in: Third International Conference on Computational Linguistics, COLING 1969, Stockholm, Sweden, September 1-4, 1969. URL: https://www.aclweb.org/anthology/C69-4801/.
[18] Margenstern, M., Păun, Gh., Rogozhin, Y., Verlan, S., 2005. Context-free insertion-deletion systems. Theoretical Computer Science 330, 339–348.
[19] Matveevici, A., Rogozhin, Y., Verlan, S., 2007. Insertion-deletion systems with one-sided contexts, in: Durand-Lose, J.O., Margenstern, M. (Eds.), Machines, Computations, and Universality, 5th International Conference, MCU, Springer. pp. 205–217.
[20] Motwani, R., Panigrahy, R., Saraswat, V., Ventkatasubramanian, S., 2000. On the decidability of accessibility problems (extended abstract), in: Proceedings of the Thirty-second Annual ACM Symposium on Theory of Computing, STOC, ACM. pp. 306–315. URL: http://doi.acm.org/10.1145/335305.335341.
[21] Mutyam, M., Krithivasan, K., Reddy, A.S., 2005. On characterizing recursively enumerable languages by insertion grammars. Fundam. Inform. 64, 317–324. URL: http://content.iospress.com/articles/fundamenta-informaticae/fi64-1-4-27.
[22] Okubo, F., Yokomori, T., 2011. Morphic characterizations of language families in terms of insertion systems and star languages. Int. J. Found. Comput. Sci. 22, 247–260. doi:10.1142/S012905411100799X.
[23] Onodera, K., 2003. A note on homomorphic representation of recursively enumerable languages with insertion grammars. Transactions of Information Processing Society of Japan 44, 1424–1427.
[24] Onodera, K., 2009. New morphic characterizations of languages in chomsky hierarchy using insertion and locality, in: Dediu, A., Ionescu, A., Martín-Vide, C. (Eds.), Language and Automata Theory and Applications, Third International Conference, LATA 2009, Tarragona, Spain, April 2-8, 2009. Proceedings, Springer. pp. 648–659. URL: https://doi.org/10.1007/978-3-642-00982-2_55, doi:10.1007/978-3-642-00982-2_55.
[25] Petre, I., Verlan, S., 2012. Matrix insertion-deletion systems. Theoretical Computer Science 456, 80–88. doi:10.1016/j.tcs.2012.07.002.
[26] Păun, Gh., 1997. Marcus Contextual Grammars. Kluwer/Springer. doi:10.1007/978-94-015-8969-7.
[27] Păun, Gh., Rozenberg, G., Salomaa, A., 1998. DNA Computing: New Computing Paradigms. Springer.
[28] Ran, F.A., Hsu, P.D., Wright, J., Agarwala, V., Scott, D.A., Zhang, F., 2013. Genome engineering using the CRISPR-Cas9 system. Nature Protocols 8, 2281–2308. doi:10.1038/nprot.2013.143.
[29] Takahara, A., Yokomori, T., 2003. On the computational power of insertion-deletion systems. Natural Computing 2, 321–336.
[30] Verlan, S., 2007. On minimal context-free insertion-deletion systems. Journal of Automata, Languages and Combinatorics 12, 317–328.
[31] Verlan, S., 2010. Recent developments on insertion-deletion systems. Computer Science Journal of Moldova 18, 210–245. URL: http://www.math.md/files/csjm/v18-n2/v18-n2-(pp210-245).pdf.

Appendix

	$\mathcal{L}\mathcal{X}$	$\mathcal{Y}\mathcal{X}$	$\mathcal{L}S'$	$\mathcal{S}\$X\mathcal{B}$	$\bar{\mathcal{B}}_1X\mathcal{B}\$$	$\$\bar{\mathcal{B}}_1X\nabla$	$\$X\nabla\$$	$\mathcal{F}\$\bar{\mathcal{B}}_1$	$\$\$\bar{\mathcal{B}}_1\$$	$\mathcal{B}\$\$\bar{\$}$	$\nabla\$\$\bar{\mathcal{B}}_1$	$\mathcal{L}\mathcal{A}\mathcal{B}\$$	$\mathcal{L}\mathcal{A}K$	$\$\mathcal{A}K$	$\mathcal{S}\bar{\$}\bar{Y}\mathcal{B}$	$\mathcal{B}_1\bar{Y}\mathcal{B}\$$	$\$\mathcal{B}_1\bar{Y}\nabla$	$\bar{\$}\bar{Y}\nabla\$$	$\mathcal{F}\bar{\$}\mathcal{B}_1$	$\$\bar{\$}\mathcal{B}_1\$$	$\mathcal{B}\$\$\bar{\$}$	$\nabla\$\$\mathcal{B}_1$
$\mathcal{L}\mathcal{X}$	[1], [1]	[], [2]		[], [2]											[], [2]							
$\mathcal{Y}\mathcal{X}$	[2], []	[1, 2], [1, 2]	[2], []	[], [2]											[], [2]							
$\mathcal{L}S'$		[], [2]	[1], [1]	[], [2]											[], [2]							
$\mathcal{S}\$X\mathcal{B}$	[2], []	[2], []	[2], []	[1, 4], [1, 4]	[3], []	[4], []	[3], []	[4], []	[3], []	[], [4]	[4], []	[3], []	[3], []	[3], []	[4], [4]	[3], []	[4], []	[3], []	[4], []	[3], []	[], [4]	[4], []
$\bar{\mathcal{B}}_1X\mathcal{B}\$$				[], [3]	[1], [1]	[], [4]	[], [4]	[4], [4]	[], [4]	[], [3]	[4], []			[], [4]						[], [4]	[], [4]	[], [3]	
$\$\bar{\mathcal{B}}_1X\nabla$				[], [4]	[4], []	[1], [1]	[4], []	[3], []	[4], []		[], [4]	[4], []			[], [4]	[4], []		[4], []		[4], []	[4], []	[], [4]
$\$X\nabla\$$				[], [3]	[4], []	[], [4]	[1, 4], [1, 4]	[], [4]	[4], [4]		[], [3]	[4], []		[], [4]		[4], []		[4], []	[], [4]	[4], [4]	[4], []	
$\mathcal{F}\$\bar{\mathcal{B}}_1$				[], [4]	[4], [4]	[], [3]	[4], []	[1], [1]	[4], []			[4], []			[], [4]	[4], []		[4], []		[4], []	[4], []	[3], []
$\$\$\bar{\mathcal{B}}_1\$$				[], [3]	[4], []	[], [4]	[4], [4]	[], [4]	[1, 4], [1, 4]			[4], []			[], [4]			[4], []		[4], []	[4], [4]	[4], []	
$\mathcal{B}\$\$\bar{\$}$				[4], []	[3], []			[], [4]		[1], [1]		[3], []			[4], []	[3], []	[], [4]	[], [4]	[], [4]	[], [3]		
$\nabla\$\$\bar{\mathcal{B}}_1$				[], [4]	[], [4]	[4], []	[3], []	[], [3]			[1], [1]				[], [4]		[4], []	[3], []	[], [3]			
$\mathcal{L}\mathcal{A}\mathcal{B}\$$				[], [3]		[], [4]	[], [4]	[], [4]	[], [4]	[], [3]		[1], [1]			[], [4]				[], [4]	[], [4]	[], [3]	
$\mathcal{L}\mathcal{A}K$				[], [3]									[1], [1]		[], [3]							
$\$\mathcal{A}K$				[], [3]	[4], []		[4], []		[4], []			[4], []		[1], [1]	[], [3]	[4], []		[4], []		[4], []	[4], []	
$\mathcal{S}\bar{\$}\bar{Y}\mathcal{B}$	[2], []	[2], []	[2], []	[4], [4]		[4], []		[4], []		[], [4]	[4], []		[3], []	[3], []	[1, 4], [1, 4]		[4], []		[4], []		[], [4]	[4], []
$\mathcal{B}_1\bar{Y}\mathcal{B}\$$				[], [3]		[], [4]	[], [4]	[], [4]	[], [4]	[], [3]				[], [4]		[1], [1]			[4], [4]	[], [4]	[], [3]	[4], []
$\bar{\$}\mathcal{B}_1\bar{Y}\nabla$				[], [4]						[4], []	[], [4]				[], [4]		[1], [1]		[3], []			[], [4]
$\bar{\$}\bar{Y}\nabla\$$				[], [3]		[], [4]	[], [4]	[], [4]	[], [4]	[4], []	[], [3]			[], [4]				[1], [1]	[], [4]	[], [4]		
$\mathcal{F}\bar{\$}\mathcal{B}_1$				[], [4]	[4], []		[4], []		[4], []			[4], []			[], [4]	[4], [4]	[], [3]	[4], []	[1], [1]	[4], []	[4], []	[3], []
$\$\bar{\$}\mathcal{B}_1\$$				[], [3]	[4], []	[], [4]	[4], [4]	[], [4]	[4], [4]	[3], []		[4], []		[], [4]		[4], []		[4], []	[], [4]	[1, 4], [1, 4]	[4], []	
$\mathcal{B}\$\$\bar{\$}$				[4], []	[3], []	[], [4]	[], [4]	[], [4]	[], [4]			[3], []		[], [4]	[4], []	[3], []			[], [4]	[], [4]	[1], [1]	
$\nabla\bar{\$}\$\mathcal{B}_1$				[], [4]		[4], []		[], [3]							[], [4]	[], [4]	[4], []		[], [3]			[1], [1]

Notations: $\mathcal{B} = \{B, D\}$, $\mathcal{A} = \{A, C\}$, $K = \{K_{AB}, K_{CD}\}$, $X \in \mathcal{N}$, $Y \in \bar{\mathcal{N}}$, $\mathcal{X}, \mathcal{Y} \in N$.

Common prefixes/suffixes between rules from Theorem 2. At the intersection of row $u = u_1 \dots u_n$ and column $v = v_1 \dots v_k$, there is a pair of lists or empty space if both lists are empty. The first (resp. second) list indicates positions P such that for each $p \in P : u_i = v_{p+i-1}$, $1 \leq i \leq k - p + 1$ (resp. $v_i = u_{p+i-1}$, $1 \leq i \leq n - p + 1$); *e.g.*, if $u = aacaa$ and $v = aac$, then the corresponding cell will contain $[1], [4, 5]$ (see also the picture below).