Computer Science Journal of Moldova, vol.**18**, no.2(53), 2010

Recent Developments on Insertion-Deletion Systems*

Sergey Verlan

Abstract

This article gives an overview of the recent developments in the study of the operations of *insertion* and *deletion*. It presents the origin of these operations, their formal definition and a series of results concerning language properties, decidability and computational completeness of families of languages generated by insertion-deletion systems and their extensions with the graph-control. The basic proof methods are presented and the proofs for the most important results are sketched.

Keywords: insertion-deletion systems, computational completeness, decidability, graph control, P systems.

1 Introduction

In general form, an insertion operation means adding a substring to a given string in a specified (left and right) context, while a deletion operation means removing a substring of a given string being in a specified (left and right) context. An insertion or deletion rule is defined by a triple (u, x, v) meaning that x can be inserted between u and v or deleted if it is between u and v. Thus, an insertion corresponds to the rewriting rule $uv \to uxv$ and a deletion corresponds to the rewriting rule $uxv \to uv$. A finite set of insertion-deletion rules, together with a set of axioms provides a language generating device: starting from the set of initial strings and iterating insertion or deletion operations as defined by the given rules one gets a language. The size of

©2010 by S. Verlan

*This work was supported by Science and Technology Center in Ukraine project Nr. 4032

the alphabet, the number of axioms, the size of contexts and of the inserted or deleted string are natural descriptional complexity measures for insertion-deletion systems.

The idea of insertion of one string into another was firstly considered with a linguistic motivation in [23] and latter developed in [8, 29]. Marcus contextual grammars investigated in above references consider couples $(x, (u, v))$, meaning that words u and v can be adjoined to the word x. This corresponds in some sense to grammars having rules of type $x \to uxv$, *i.e.*, u and v are inserted around the position marked by x. Such grammars are alternative concepts to Chomsky grammars and present the evolution of the descriptive linguistics. Many interesting linguistic properties like ambiguity and duplication can be captured in this framework. The insertion of a string in a specified context was firstly considered in [8].

In [9, 10] the insertion operation and its iterated variant is introduced with a different motivation. The author considers this operation as generalization of Kleene's operations of concatenation and closure [17]. The operation of concatenation would produce a string x_1x_2y from two strings x_1x_2 and y. By allowing the concatenation to happen anywhere in the string and not only at its right extremity a string x_1yx_2 can be produced, *i.e.*, y is inserted into x_1x_2. In [13] the deletion is defined as a right quotient operation which happens not necessarily at the rightmost end of the string. In the same thesis the duality between the insertion and deletion is also highlighted: any insertion system generating a language $\mathcal{L}$ is at the same time a deletion system recognizing $\mathcal{L}$. The operations considered in above works correspond to context-free variants of insertion and deletion operations, because no contexts are used. In the same place several other variants of insertion and deletion are introduced and their closure properties are investigated.

The third inspiration for insertion and deletion operations comes, surprisingly, from the field of molecular biology. In fact they correspond to a mismatched annealing of DNA sequences. We refer to [32] for more details. Such operations are also present in the evolution processes under the form of point mutations as well as in RNA editing, see the discussions in [3], [34] and [32]. This biological motivation of insertion-

deletion operations lead to their study in the framework of molecular computing, see, for example, [6], [14], [32], [35].

This article is organized as follows. After a formal definition given in Section 2, the next section describes existing formal proof methods and presents a recent proof technique. Section 4 considers context-free insertion-deletion systems and their links to the previous results in the formal language theory. After that in Section 5 one-sided insertion-deletion systems are considered. Section 6 investigates the graph-control extension of insertion-deletion systems that permits to increase the computational power for non-complete classes. Finally, Section 7 considers the variant where only the insertion operation is used.

2 Formal definition

We do not present here definitions concerning standard concepts of the theory of formal languages and we refer to [33] for more details.

The empty string is denoted by λ. The *length* of the word $x \in V^*$ is the number of symbols which appear in x and it is denoted by $|x|$. The number of occurrences of a symbol $a \in V$ in $x \in V^*$ is denoted by $|x|_a$. If $x \in V^*$ and $U \subseteq V$, then we denote by $|x|_U$ the number of occurrences of symbols from U in x. For a word $w \in V^*$ we define $Perm(w) = \{w' : |w'|_a = |w|_a \text{ for all } a \in V\}$. The length set of a language L is defined as $|L| = \{|x| : x \in L\}$. The length set of a family of languages $\mathcal{F}$ is defined analogously: $N\mathcal{F} = \{|L| : L \in \mathcal{F}\}$.

The family of matrix languages, *i.e.*, the family of languages generated by matrix grammars without appearance checking is denoted by MAT. The family of recursively enumerable languages is denoted by RE. The *Parikh image* of a language family $\mathcal{F}$ is a family of sets of vectors denoted by $Ps\mathcal{F}$ (we assume a fixed ordering on the alphabet $T = \{a_1, \ldots, a_n\}$), and is defined as follows: $Ps(L) = \{(|w|_{a_1}, \ldots, |w|_{a_n}) : w \in L\}$ and $Ps\mathcal{F} = \{Ps(L) : L \in \mathcal{F}\}$.

An *insertion-deletion system* is a construct $ID = (V, T, A, I, D)$, where V is an alphabet, $T \subseteq V$, A is a finite language over V, and I, D are finite sets of triples of the form (u, α, v), $\alpha \neq \lambda$, where u and v are strings over V. The elements of T are *terminal* symbols

(in contrast, those of $V - T$ are called nonterminals), those of A are *axioms*, the triples in I are *insertion rules*, and those from D are *deletion rules*. An insertion rule $(u, \alpha, v) \in I$ indicates that the string α can be inserted between u and v, while a deletion rule $(u, \alpha, v) \in D$ indicates that α can be removed from the context (u, v). As stated otherwise, $(u, \alpha, v) \in I$ corresponds to the rewriting rule $uv \to u\alpha v$, and $(u, \alpha, v) \in D$ corresponds to the rewriting rule $u\alpha v \to uv$. We denote by $\Longrightarrow_{ins}$ the relation defined by an insertion rule (formally, $x \Longrightarrow_{ins} y$ iff $x = x_1uvx_2, y = x_1u\alpha vx_2$, for some $(u, \alpha, v) \in I$ and $x_1, x_2 \in V^*$) and by $\Longrightarrow_{del}$ the relation defined by a deletion rule (formally, $x \Longrightarrow_{del} y$ iff $x = x_1u\alpha vx_2, y = x_1uvx_2$, for some $(u, \alpha, v) \in D$ and $x_1, x_2 \in V^*$). We refer by $\Longrightarrow$ to any of the relations $\Longrightarrow_{ins}, \Longrightarrow_{del}$, and denote by $\Longrightarrow^*$ the reflexive and transitive closure of $\Longrightarrow$ (as usual, $\Longrightarrow^+$ is its transitive closure).

The language generated by ID is defined by

$$L(ID) = \{w \in T^* \mid x \Longrightarrow^* w, x \in A\}.$$

The complexity of an insertion-deletion system $ID = (V, T, A, I, D)$ is described by the vector $(n, m, m'; p, q, q')$ called *size*, where

$$n = \max\{|\alpha| \mid (u, \alpha, v) \in I\}, \qquad p = \max\{|\alpha| \mid (u, \alpha, v) \in D\},$$
$$m = \max\{|u| \mid (u, \alpha, v) \in I\}, \qquad q = \max\{|u| \mid (u, \alpha, v) \in D\},$$
$$m' = \max\{|v| \mid (u, \alpha, v) \in I\}, \qquad q' = \max\{|v| \mid (u, \alpha, v) \in D\}.$$

We also denote by $INS_n^{m,m'}DEL_p^{q,q'}$ corresponding families of insertion-deletion systems. Moreover, we define the total size of the system as the sum of all numbers above: $\psi = n + m + m' + p + q + q'$.

If some of the parameters n, m, m', p, q, q' is not specified, then we write instead the symbol $*$. In particular, $INS_*^{0,0}DEL_*^{0,0}$ denotes the family of languages generated by *context-free insertion-deletion systems*. If one of numbers from the couples m, m' and/or q, q' is equal to zero (while the other is not), then we say that corresponding families have a one-sided context.

We remark that, historically, another complexity measure called *weight* was used for insertion-deletion systems. It corresponds to 4-tuples $(n, \bar{m}; p, \bar{q})$, where $\bar{m} = \max\{m, m'\}$ and $\bar{q} = \max\{q, q'\}$.

3 Basic simulation principles

In this section we show some important properties of insertion-deletion systems, present some normal forms and indicate basic methods for equivalence proofs used in the rest of the chapter.

We start with the presentation of the normal form for insertion-deletion systems.

Definition 3.1. An insertion-deletion system $ID = (V \cup \{\$\}, T, A, I, D \cup D_2)$ of size $(n, m, m'; p, q, q')$ is said to be in the *normal form* if

- for any $(u, x, v) \in I$ it holds $|u| = m$, $|v| = m'$, $|x| = n$,
- for any $(u, x, v) \in D$ it holds $|u| = q$, $|v| = q'$, $|x| = p$,
- for any $(u, x, v) \in D$ it holds that x contains no letters from T,
- the set D_2 is defined as $D_2 = \{(\lambda, \$, \lambda)\}$.

Theorem 3.2. *For any insertion-deletion system ID it is possible to construct a system ID' in normal form and having same size such that $L(ID) = L(ID')$.*

This affirmation is quite obvious. For the first two conditions it is enough to replace any rule having left or right contexts of a smaller size by a group of rules, where the left (resp. right) context is a string over $V \cup \{\$\}$ of the required size. The same holds for the inserted or deleted symbol and axioms. More precisely, the new symbol \$ permits to fill the context of rules and sizes of axioms up to the desired size.

The third condition can be satisfied as follows. For any terminal symbol $t \in T$ a special non-terminal N_t is considered. All rules and axioms involving t are duplicated and t replaced by N_t. This construction ensures that symbol N_t acts like an alias for the symbol t, *i.e.* for any derivation producing $w_1 t w_2$ there is another derivation producing $w_1 N_t w_2$. Hence there is no difference between erasing t or N_t, therefore all deletion rules involving t can be omitted. A formal proof of the theorem can be found in [2].

Example 3.3.

Consider the system $ID = (\{a, b, C\}, \{a, b\}, \{ab\}, I, D)$ of size $(2, 1, 1; 2, 1, 1)$, with $I = \{(a, aC, b), (a, b, C)\}$ and $D = \{(b, C, b)\}$. Then ID' can be defined as follows: $ID' = (\{a, b, C, \$\}, \{a, b\}, \{ab \sqcup\!\sqcup \$\$\}, I', D' \cup \{(\lambda, \$, \lambda)\})$, where $I' = \{(a, aC, b), (a, \$b, C), (a, b\$, C)\}$ and $D' = \{(b, C\$, b), (b, \$C, b)\}$.

Insertion-deletion systems represent a powerful model of computation. If the size of the system is not bounded, then an arbitrary grammar can be simulated.

Theorem 3.4. *For any type-0 grammar $G = (N, T, S, P)$ there is an insertion-deletion system $ID = (V, T, A, I, D)$ such that $L(G) = L(ID)$.*

Proof. Let $V = N \cup \{\#_i : 1 \leq i \leq |P|\} \cup \{\$\}$. Let $k_1 = \max\{|u|, u \to v \in P\}$ and $k_2 = \max\{|v|, u \to v \in P\}$. Consider $k = \max(k_1, k_2)$. The set A is defined as $A = \{\$^k S \$^k\}$.

For any rule $i : u \to v \in P$ we add insertion rules $(xu, \#_i v, y)$, $x, y \in (N \cup \{\$\})^*$, $|xu| = k$, $|y| = k$, to I and a deletion rule $(x, u\#_i, v)$, $x \in N \cup \{\$\}$ to D. Finally, a rule $(\lambda, \$, \lambda)$ is added to D.

It is not difficult to see that such system simulates G. Indeed, for any derivation $w_1 u w_2 \Longrightarrow w_1 v w_2$ in G there is a following two-step derivation $\$^k w_1 u w_2 \$^k \Longrightarrow \$^k w_1 u \#_i v w_2 \$^k \Longrightarrow \$^k w_1 v w_2 \$^k$ in ID that simulates the corresponding production of G. If $w \in L(G)$ then the string $\$^k w \$^k$ will be obtained in ID. Additional symbols \$ can be deleted at this moment. So $w \in L(ID)$.

For the converse inclusion it is enough to observe that if an insertion rule $(xu, \#_i v, y)$ is used, then no more insertions inside the corresponding site xu can be done. So, the only way to eliminate the symbol $\#_i$ is to perform the corresponding deletion. Hence the computation in ID can be rearranged in such a way that an insertion is followed by the corresponding deletion. This corresponds to a derivation step in G, which completes the proof. □

As one can see from the previous theorem, the basic idea of grammar simulation by insertion-deletion systems is a construction of a set

of related insertion and deletion rules that shall be used in some specified sequence thus performing a grammar rule simulation. Usually, insertion rules introduce new non-terminal symbols in the string which can be deleted only by corresponding deletion rules (like symbols $\#_i$ in the theorem above). If the correct sequence is not performed, then some non-terminal symbols that cannot be deleted will remain in the string. In the subsequent sections different variants of this method are shown permitting to decrease the size of used insertion and deletion rules.

3.1 The method of direct simulation

A simulation of type-0 grammars by insertion-deletion systems is the main method permitting to prove the computational completeness of insertion-deletion systems. However, when several such results are established, it is much easier to prove the computational completeness by simulating another insertion-deletion systems. For example:

Theorem 3.5. $INS_1^{1,1}DEL_2^{0,0} = RE$.

Sketch of Proof. The proof may be done by simulating insertion-deletion systems of size $(1,1,1;1,1,1)$ which are known to be computationally complete, see [35, 36]. In this case it is enough to show how a deletion rule (a,b,c), $a,b,c \in V$ can be simulated using insertion and deletion rules of size $(1,1,1;2,0,0)$. Let $a \neq b \neq c$. Then a deletion rule (a,b,c) with label i may be simulated by a sequence of the following rules: $\{(a,\}_i,b),(b,]_i,c),(a,[_i,\}_i),([_i,\{_i,\}_i),([_i,K_i,\{_i,)\} \subseteq I$ and $(\lambda,\{_i\}_i,\lambda),(\lambda,K_ib,\lambda),(\lambda,[_i]_i,\lambda) \subseteq D$. The simulation is performed as follows (we underline the inserted symbols):

$$w_1abcw_2 \Longrightarrow_{ins} w_1a\underline{\}_i}bcw_2 \Longrightarrow_{ins} w_1a\underline{[_i}\}_ibcw_2 \Longrightarrow_{ins} w_1a[_i\}_ib\underline{]_i}cw_2$$
$$\Longrightarrow_{ins} w_1a[_i\underline{\{_i}\}_ib]_icw_2 \Longrightarrow_{ins} w_1a[_i\underline{K_i}\{_i\}_ib]_icw_2 \Longrightarrow_{del}$$
$$\Longrightarrow_{del} w_1a[_iK_ib]_icw_2 \Longrightarrow_{del} w_1a[_i]_icw_2 \Longrightarrow_{del} w_1acw_2.$$

The idea behind the simulation is the following. Symbols $[_i$ and $]_i$ delimit the deletion site. Symbol K_i performs the deletion of b, while

symbols $\}_i$ and $\{_i$ ensure that K_i is inserted only once after $[_i$ (hence only one b can be deleted). If all the above steps are not performed, then some of additional symbols will remain in the string, hence it will never become terminal. This is a common method of simulation: the working (insertion or deletion) site is delimited by special symbols in order to avoid interactions between several such sites and inside the site the sequence of insertions and deletions permits to simulate exactly one application of the corresponding rule. All additional symbols are related in such a way that the whole sequence of insertions and deletions shall be performed in order to eliminate all of them.

We remark that it would be wrong to simulate a deletion rule (a, b, c) by only rules $\{(a, [_i, b), (b,]_i, c), ([_i, K_i, b)\} \subseteq I$ and $(\lambda, K_i b, \lambda)$, $(\lambda, [_i]_i, \lambda) \subseteq D$, because it is possible to erase several symbols b, which leads to a wrong computation:

$$\begin{aligned} w_1abbcw_2 &\Longrightarrow_{ins} w_1a\underline{[_i}bbcw_2 \Longrightarrow_{ins} w_1a[_ibb]\underline{_i}cw_2 \Longrightarrow_{ins} \\ &\Longrightarrow_{ins} w_1a[_i\underline{K_i}bb]_icw_2 \Longrightarrow_{del} w_1a[_ib]_icw_2 \Longrightarrow_{ins} \\ &\Longrightarrow_{ins} w_1a[_i\underline{K_i}b]_icw_2 \Longrightarrow_{del} w_1a[_i]_icw_2 \Longrightarrow_{del} w_1acw_2. \end{aligned}$$

□

The above approach is very powerful and it permits to establish the computational completeness of the corresponding class of insertion-deletion systems in a much easier way. For example, the proof of Theorem 3.5 in [32] (Theorem 6.3) takes more than two pages. The method is quite generic, in order to use it one should find a computational complete class of insertion-deletion systems having same insertion or deletion parameters. Then, in order to prove the computational completeness, it is sufficient to simulate corresponding deletion or insertion operation. This is significantly easier than the simulation of a Chomsky grammar because only left-hand or only right-hand side of a production $u \to v$ shall be simulated.

Most of the recent results about the universality of insertion-deletion systems are obtained using this technique.

4 Context-free insertion-deletion systems

In this section we present an important class of insertion-deletion systems: systems with context-free rules. This permits to bridge recent results with early investigations from [9] and [13] giving answers to old questions from this area.

4.1 Computational completeness results

We start with the sketch of the proof of the computational completeness for context-free insertion-deletion systems. More details can be found in [24].

Theorem 4.1. $INS_*^{0,0}DEL_*^{0,0} = RE$.

Sketch of proof. Let $G = (N, T, S, P)$ be type-0 Chomsky grammar where N, T are disjoint alphabets, $S \in N$, and P is a finite subset of rules of the form $u \to v$ with $u, v \in (N \cup T)^*$ and u contains at least one letter from N. We assume all rules from P labeled in a one-to-one manner with elements of a set M, disjoint of $N \cup T$.

We construct the following context-free insertion-deletion system: $\gamma = (N \cup T \cup M, T, \{S\}, I, D)$, where

$$I = \{(\lambda, vR, \lambda) \mid R : u \to v \in P,\ R \in M,\ u, v \in (N \cup T)^*\},$$
$$D = \{(\lambda, Ru, \lambda) \mid R : u \to v \in P,\ R \in M,\ u, v \in (N \cup T)^*\}.$$

Two rules $(\lambda, vR, \lambda) \in I, (\lambda, Ru, \lambda) \in D$ as above are said to be *M-related.*

We have the equality $L(G) = L(\gamma)$.

The inclusion $L(G) \subseteq L(\gamma)$ is obvious: each derivation step $x_1ux_2 \Longrightarrow x_1vx_2$, performed in G by means of a rule $R : u \to v$, can be simulated in γ by an insertion operation step $x_1ux_2 \Longrightarrow_{ins} x_1vRux_2$ which uses the rule $(\lambda, vR, \lambda) \in I$, followed by the deletion operation $x_1vRux_2 \Longrightarrow_{del} x_1vx_2$ which uses the rule $(\lambda, Ru, \lambda) \in D$.

Consider now the inclusion $L(\gamma) \subseteq L(G)$. The idea of the proof is to transform any terminal derivation in γ into one in which any two

consecutive (odd, even) derivations steps simulate one production in G. Because the labels of rules from P precisely identify a pair of M-related insertion-deletion rules, and the elements of M are nonterminal symbols for γ, every terminal derivation with respect to γ must involve the same number of insertion steps and deletion steps; moreover, these steps are performed by using pairs of M-related rules from I and D.

For every terminal derivation in γ it is possible to construct an equivalent derivation, using the same rules in a different order, and having only matching pairs of consecutive rules, *i.e.* odd steps $w_i \Longrightarrow_{ins} w_{i+1}$ are performed by a rule $(\lambda, vR, \lambda) \in I$, while even steps $w_{i+1} \Longrightarrow w_{i+2}$ are performed by using the M-related rule $(\lambda, Ru, \lambda) \in D$. Clearly, two consecutive steps of a derivation in γ which use M-related rules $(\lambda, vR, \lambda) \in I, (\lambda, Ru, \lambda) \in D$, correspond to a derivation step in G which uses the rule $R : u \to v$. This implies the inclusion $L(\gamma) \subseteq L(G)$. □

The context control of a type 0 grammar does not really disappear in the corresponding insertion-deletion system (as constructed in Theorem 4.1 above). It rather changes its form, becoming a rigid synchronization of insertions and deletions. In other terms, if a word u represents the context of a word v in a "context-sensitive production" $R : u \to v$, then in the corresponding insertion-deletion system the word v will also be conditioned by the later occurrence of u in a successful derivation (hence u is yet again the context of v). This condition is enforced by the newly introduced symbol R which acts as a "remote context binder". The fact that the context u "seems" to appear after the context-controlled v is of no importance, reflecting the reversal of generative process of the grammar.

Let us denote by $L \leftrightarrows \begin{smallmatrix} L_1 \\ L_2 \end{smallmatrix}$ the operation of insertion-deletion that inserts words from L_1 into L or deletes words belonging to L_2 from L and by $L \leftrightarrows^* \begin{smallmatrix} L_1 \\ L_2 \end{smallmatrix}$ its reflexive and transitive closure. Then the following representation of RE holds:

Theorem 4.2. *Any language $L \in RE$ can be represented in the following form $L = \left(\{S\} \leftrightarrows^* \begin{smallmatrix} L_1 \\ L_2 \end{smallmatrix}\right) \cap T^*$, where L_1 and L_2 are two finite languages and T is an alphabet.*

In the proof of Theorem 4.1, the length of inserted or deleted strings is not bounded, but a bound can be easily found by controlling the length of strings appearing in the rules of the starting type-0 grammar:

Theorem 4.3. $INS_3^{0,0}DEL_3^{0,0} = RE$.

Proof. Let $G = (N, T, S, P)$ be type-0 Chomsky grammar in Kuroda normal form. Then, the rules of the context-free insertion-deletion system constructed in the proof of Theorem 4.1 are of the form $(\lambda,\ \alpha,\ \lambda)$ with $|\alpha| \leq 3$, hence $RE \subseteq INS_3^0DEL_3^0$. □

The total size of the system provided by the proof of Theorem 4.1 is 6. We can improve by one this result, by decreasing by one either the length of the inserted strings or the length of the deleted strings. These proofs can be done by a direct simulation of systems of size $(3, 0, 0; 3, 0, 0)$ using the method presented in Section 3.1.

Theorem 4.4. *[24]* $INS_3^{0,0}DEL_2^{0,0} = RE$.

A counterpart of this result is also true: we can trade-off the length of inserted and deleted strings.

Theorem 4.5. *[24]* $INS_2^{0,0}DEL_3^{0,0} = RE$.

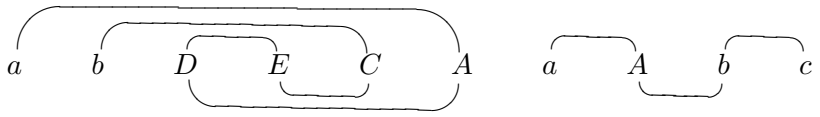
4.2 Non-completeness results

We show below that the above complexity parameters for context-free insertion-deletion systems are optimal. If one of the parameters is further decreased, then the language generated by such systems is included in the family of context-free languages.

The main idea used to obtain this result is that the non-terminal alphabet can be omitted, hence, the deletion can also be omitted.

This can be argued as follows. Consider a derivation of $w \in T^*$ starting from an empty word. Let us mark the corresponding insertion pairs by an overline and the corresponding deletion pairs by an underline. For example, suppose that we insert aA, after that bC in position 1, DE in position 2, aA in position 6 and bc in position 8. After

that suppose that we delete EC, DA and Ab. Then the corresponding marking will be as follows (the resulting word is $w = abac$):



We may interpret symbols as labeled graph nodes and lines as edges. In this case we obtain a graph. It is easy to observe that this graph consists of a set of disjoint linear paths and/or cycles. Indeed, for each node, at most two edges corresponding to an insertion and a deletion may be drawn. Let us also label edges corresponding to insertions by i and edges corresponding to deletions by d. If we take the example above, we obtain:

$$a \overset{i}{\text{——}} A \overset{d}{\text{——}} D \overset{i}{\text{——}} E \overset{d}{\text{——}} C \overset{i}{\text{——}} b$$

$$a \overset{i}{\text{——}} A \overset{d}{\text{——}} b \overset{i}{\text{——}} c$$

We may suppose that the first and the last edge of a path are marked with i. If this is not the case, we add an additional node labeled by λ and we connect this node with the last node by a path labeled by i. In particular, a path containing only one letter a (corresponding to an insertion of a) will be written as $\lambda \overset{i}{\text{——}} a$. Hence, each path consists of sequences of one insertion followed by one deletion.

We observe that for a derivation of a word $w \in T^*$ there can only be paths of the following 4 types: (1) paths that start with a letter $a \in T$ and that end with a letter $b \in T$; (2) Paths that have at one end a terminal letter a and at the other end λ; (3) paths that have λ at both ends; (4) Cycles.

We remark that in Case 1 the path leads to the word ab (*i.e.*, contributes to the production of the subword ab of w), in the second case the path produces the letter a and in the last two cases the path generates the empty word.

Without loss of generality, we may suppose that there are no paths of type 3 and 4, because by eliminating the corresponding insertions

and deletions we obtain the same word.

Suppose that we have a path marked by over- and underlines as above. We shall understand by an interior of the path the set of all positions that are underlined. In the example above, all positions between D and the first A form the interior of the path. It is clear that no other path (of type 1 and 2) may be situated in the interior of some path, because in this case the corresponding deletion cannot be performed. Consequently, all paths are independent of each other, and we may group rules corresponding to each path and compute paths one after another. Moreover, each path contributes to at most two terminal symbols of the resulting word. Therefore, the computation consists of insertion of terminal symbols corresponding to paths ends as well as of deletion of terminal symbols.

Moreover, we can show that it is possible to precompute all possible paths. This may be done by using the following observation. We may assume that each path p has the following property: if $A \overset{i}{-} B$ belongs to p, then p does not contain an insertion that has A in the left-hand side ($A \overset{i}{-} X$) or B in the right-hand side ($Y \overset{i}{-} B$). This assertion is obvious, because if p contains such a pair, for example $p = \cdots \overset{d}{-} A \overset{i}{-} X \overset{d}{-} \cdots \overset{d}{-} A \overset{i}{-} B \cdots$, then we may eliminate the subpath between two A's by obtaining an equivalent path (that leads to the same ends) $p' = \cdots \overset{d}{-} A \overset{i}{-} B \cdots$. So, the length of each path is bounded by $2 \cdot card(V)$, and we may precompute all possible paths.

In a similar manner it can be proved that the nonterminal alphabet is not relevant even in the general case. See [37] for details.

Lemma 4.6. *Let $ID = (V, T, A, I, D)$ be a context-free insertion-deletion system of size $(2, 0, 0; 2, 0, 0)$. Then it is possible to construct a system $ID_2 = (T, T, A_2, I_2, D_2)$ of size $(2, 0, 0; 2, 0, 0)$ such that $L(ID) = L(ID_2)$.*

Moreover, if we consider that the initial system is in the normal form, then there are no deletions of terminal symbols. Hence we obtain that it is sufficient to consider insertion-only systems as $INS_2^{0,0}DEL_2^{0,0} \subseteq INS_2^{0,0}DEL_0^{0,0}$.

We can describe insertion-deletion systems of size $(2,0,0;0,0,0)$ by the following context-free grammar, which is a particular case of a more general result for systems of size $(*,1,1;0,0,0)$ given in [32].

Let $ID = (T, T, A, I, \emptyset)$ be an insertion-deletion system of size $(2,0,0;0,0,0)$. We construct the following context-free grammar $G = (\{Z,S\}, T, Z, P)$. Define $P = P_A \cup P_I \cup \{S \to \lambda\}$, where

$$\begin{aligned} P_A &= \{Z \to Sa_1Sa_2S \dots Sa_nS \mid a_1a_2 \dots a_n \in A\}, \\ P_I &= \{S \to SaSbS \mid (\lambda, ab, \lambda) \in I\} \cup \{S \to SaS \mid (\lambda, a, \lambda) \in I\}. \end{aligned}$$

It is clear that $L(G) = L(ID)$. Indeed, symbol S marks all possible insertion positions and permits the simulation of insertion rules as well.

Consequently, we obtain:

Theorem 4.7. $INS_2^{0,0}DEL_2^{0,0} = INS_2^{0,0}DEL_0^{0,0} \subset CF$.

Proof. The strictness of the inclusion follows from the fact that insertion-deletion systems of size $(2,0,0;0,0,0)$ cannot generate the language $L = \{a^*b^*\}$. Indeed, consider an arbitrary system $ID = (T, T, A, I, \emptyset)$. It is easy to observe that for each word w that belongs to $L(ID)$, words $\{x^*wx^* \mid (\lambda, x, \lambda) \in I\}$ belong to $L(ID)$. Therefore, if we suppose that $L(ID)$ is not finite, then $I \neq \emptyset$, and then for any word $w \in L(ID)$, there are words $\{x^*wx^* \mid (\lambda, x, \lambda) \in I\}$ in $L(ID)$. It is easy to see that L does not have such a property. □

Theorem 4.8. $INS_2^{0,0}DEL_2^{0,0}$ *is incomparable with* REG.

Proof. Previous theorem gives $REG \setminus INS_2^{0,0}DEL_2^{0,0} \neq \emptyset$. It is also clear that the Dyck language D_n may be generated by a context-free insertion system having insertion rules $(\lambda, a_i\bar{a}_i, \lambda)$, $1 \leq i \leq n$. Hence, the assertion is proved. □

From the description above it is clear that languages generated by insertion-deletion systems of size $(2,0,0;2,0,0)$ have a particular structure (below, we denote by $\prod$ the concatenation operation).

Theorem 4.9. *A language L belongs to $INS_2^{0,0}DEL_2^{0,0}$ if and only if it can be represented in the form*

$$L = h\left(T'^* \sqcup \bigcup_{w=a_1\ldots a_n \in A} \prod_{i=1}^{|w|} Da_iD\right),$$

where $A \subseteq T^*$ is a finite set of words, T is an alphabet, D is the Dyck language over an alphabet $T'' \subseteq T$, h is a coding and $T' \subseteq T$.

In a similar way next two results can be obtained. See [37] for more details.

Theorem 4.10. $INS_m^{0,0}DEL_1^{0,0} = INS_m^{0,0}DEL_0^{0,0} \subset CF$, $m > 0$.

Theorem 4.11. $INS_1^{0,0}DEL_p^{0,0} \subset REG$ *for any* $p > 0$.

We collect all results above as well as some other results about the computational power of symmetrical insertion-deletion systems in Table 1.

Table 1. Results on symmetrical insertion-deletion systems

Size	Family	Ref.	Size	Family	Ref.
$(1,2,2;1,1,1)$	RE	[14, 32]	$(3,0,0;2,0,0)$	RE	[24]
$(1,2,2;2,0,0)$	RE	[14, 32]	$(1,1,1;2,0,0)$	RE	[32]
$(2,1,1;2,0,0)$	RE	[14, 32]	$(2,0,0;1,1,1)$	RE	[20]
$(1,1,1;1,2,2)$	RE	[35]	$(1,1,1;1,1,1)$	RE	[35]
$(2,1,1;1,1,1)$	RE	[35]	$(2,0,0;2,0,0)$	$\subsetneq CF$	[37]
$(3,0,0;3,0,0)$	RE	[24]	$(m,0,0;1,0,0)$	$\subsetneq CF$	[37]
$(2,0,0;3,0,0)$	RE	[24]	$(1,0,0;p,0,0)$	$\subsetneq REG$	[37]

5 One-sided insertion-deletion systems

In this section we present results about insertion-deletion systems with one-sided context, *i.e.*, of size $(n, m, m'; p, q, q')$ where either $m+m' > 0$

and $m * m' = 0$, or $q + q' > 0$ and $q * q' = 0$, *i.e.*, one of numbers in some couple is equal to zero.

One-sided insertion-deletion systems present features common to both contextual and context-free insertion-deletion systems. More precisely, an insertion rule having an empty left (or right) context can be applied any number of times like in the case of context-free rules. However, while a context-free insertion can happen anywhere in the string, in the case of a one-sided insertion the context indicates the place where the insertion can happen. Similar properties are exposed by deletion rules.

Example 5.1.

Consider a system $ID = (T, T, \{a\}, I, \emptyset)$, where $T = \{a, b, c, d\}$ and I is defined as follows: $I = \{(a, b, \lambda), (b, c, \lambda), (c, d, \lambda), (d, a, \lambda)\}$.

Let L be the language generated by ID ($L = L(ID)$). It is clear that L can be defined by the following formulas:

$$L = L_1 \quad L_1 = aL_2^* \quad L_2 = bL_3^* \quad L_3 = cL_4^* \quad L_4 = dL_1^*$$

By substituting L_i, for $2 \leq i \leq 4$ into the description of L_{i-1} we obtain:

$$L_1 = a(b(c(dL_1^*)^*)^*)^*$$

Let $R = \{(abcd)^*(dcb)^*\}$. Consider the language $L'' = L \cap R$. Consider the word $w = abcddbc$ from R. This word is generated in L as follows (we underline the inserted symbol):

$$a \Longrightarrow a\underline{b} \Longrightarrow a\underline{b}b \Longrightarrow ab\underline{c}b \Longrightarrow ab\underline{c}cb \Longrightarrow abc\underline{d}cb \Longrightarrow abc\underline{d}dcb$$

We observe that the generation of the second part of w, the subword dcb, is related to the generation of its first part $abcd$, because every letter is inserted two times: first for the second part and after that for the first part. It is also clear that this is the only way to generate the subword dcb. Moreover, it can be easily seen that such a generation leads to a one-to-one correspondence between $abcd$ and dcb. Now, taking w it is possible to insert a after the first letter d and to continue in a similar manner as before and so on, which gives $w_n = (abcd)^n(dcb)^n$, $n \geq 1$. It is also possible to obtain

more copies of $abcd$ by performing insertions of four corresponding letters after d, c, b or a in the first part of w_n. Hence, we finally obtain $L'' = \{(abcd)^i(dcb)^j, j \leq i\}$, which is a non-regular context-free language (by the inverse morphism $\{abcd \to x, dcb \to y\}$ it becomes the well known language $\{x^i y^j, 1 \leq j \leq i\}$). Since the intersection of two regular languages would be regular, we obtain that L is a non-regular context-free language.

5.1 Computational completeness results

Generally, computational completeness proofs for one-sided insertion-deletion systems take into account the above behavior and ensure that additional symbols that potentially can be inserted more than one time are inserted exactly once. This property is usually satisfied by introducing groups of insertion and deletion rules of a special form that can act only if a specified pattern is present in the string. If the pattern is compromised by inserting or deleting more than one additional symbol, then the whole group of rules will fail and non-terminal symbols will remain in the string; moreover, it can be guaranteed that these symbols cannot be eliminated anymore.

The proofs are based on simulation of insertion-deletion systems from Sections 3 and 4 which are known to generate all RE languages. The proof technique is very similar to the one from Theorem 3.5.

We remark that by symmetry, all results for classes $INS_n^{m,m'}DEL_p^{q,q'}$ are also true for classes $INS_n^{m',m}DEL_p^{q',q}$.

We give the sketch of proof for the following theorem.

Theorem 5.2. $INS_1^{1,2}DEL_1^{1,0} = RE$.

Sketch of Proof. The proof is based on the simulation of insertion-deletion systems of size $(1, 1, 1; 1, 1, 1)$ in normal form. Hence, it is sufficient to show how a deletion rule (a, x, b), with $a, b, x \in V$, may be simulated by using rules of the target system, *i.e.*, insertion rules of type $(a', x', b'c')$ and deletion rules of type (a'', y, λ).

Since the system is in normal form, we may assume that $ab \neq \lambda$. Moreover, we may assume that the system has no insertion rules of the form $(a, b, b), a, b \in V$. If this is the case then we replace every such rule

by two insertion rules (a, X, b), (a, b, X), and one deletion rule (b, X, b), where X is a new nonterminal.

A deletion rule $i : (a, x, b)$, where i is the label of the rule, is simulated by two insertion rules (x, X_i, b), (a, D_i, xX_i) and three deletion rules (D_i, x, λ), (D_i, X_i, λ), (a, D_i, λ).

Symbols D_i and X_i act like left and right parentheses that surround x before deleting it. The simulation is performed as follows. First, two insertions are performed:

$$w_1axbw_2 \Longrightarrow_{ins} w_1ax\underline{X_i}bw_2 \Longrightarrow_{ins} w_1a\underline{D_i}xX_ibw_2,$$

and then x is deleted:

$$w_1aD_ixX_ibw_2 \Longrightarrow_{del} w_1aD_iX_ibw_2.$$

At this moment symbols X_i and D_i are deleted:

$$w_1aD_iX_ibw_2 \Longrightarrow_{del} w_1aD_iw_2 \Longrightarrow_{del} w_1abw_2.$$

Hence, every derivation in an insertion-deletion system having the size $(1, 1, 1; 1, 1, 1)$ can be carried out in a system of size $(1, 1, 2; 1, 1, 0)$. On the other hand, we observe that once being inserted, the nonterminals X_i, D_i can be erased only by the rules shown above. Moreover, if they are not deleted, then no symbol can be inserted at the right of a or at the left of b. The rule (D_i, x, λ) can delete at most one x as the pair D_ix is followed by X_ib and $b \neq x$. Thus, there is a one-to-one correspondence between the original and the new systems, which implies that the theorem statement holds. □

In a similar way the results from Table 2 are obtained. We remark that last three results are counterparts of the first three results, where the sizes for insertion and deletion are interchanged. However, in general, systems where insertion parameters are $1, 1, 0$ are simpler than systems having deletion parameters $1, 1, 0$. This is due to the fact that it is easier to control a repeated insertion of symbols by using deletion than a repeated deletion of symbols by using insertion. In the latter case, special "barrier" symbols shall be inserted in order to delimit exactly one symbol to be deleted.

Table 2. Computationally complete one-sided insertion-deletion systems

Size	Ref.	Size	Ref.
(1,1,2;1,1,0)	[20]	(1,1,0;1,1,2)	[26]
(2,0,2;1,1,0)	[20]	(1,1,0;2,0,2)	[26]
(2,0,1;2,0,0)	[20]	(2,0,0;2,0,1)	[26]
(1,2,0;1,0,2)	[21]		

5.2 Non-completeness results

In what follows we show that there are classes of one-sided insertion-deletion systems that are not computationally complete.

We start with the following result.

Theorem 5.3. $REG \setminus INS_1^{1,0}DEL_1^{1,1} \neq \emptyset$.

Sketch of Proof. Consider the regular language $L = \{(ba)^+\}$. We claim that there is no insertion-deletion system ID of size (1,1,0;1,1,1) such that $L(ID) = L$. We can suppose that ID is in normal form.

Let $w_f \in (ba)^+$ be a word generated by ID. Now consider an arbitrary ba block of w_f ($w_f = \beta ba\gamma$, $\beta, \gamma \in (ba)^*$) and take its letter a. Since there are no rules deleting terminal symbols in ID this letter is either inserted by an insertion rule or it was a part of an axiom. We may omit the latter case by taking a derivation that produces a string that is long enough. Now suppose that this letter was inserted using a rule $(z, a, \lambda) \in I$, $z \in V$:

$$w \Longrightarrow^* w_1 z w_2 \Longrightarrow w_1 z a w_2 \Longrightarrow^* \beta ba\gamma = w_f. \tag{1}$$

This means that:

$$\begin{array}{c} w_1 z \Longrightarrow^* \beta b \\ a w_2 \Longrightarrow^* a\gamma \end{array} \tag{2}$$

Now we remark that symbol a might be inserted twice:

$$w \Longrightarrow^* w_1zw_2 \Longrightarrow w_1zaw_2 \Longrightarrow w_1zaaw_2. \tag{3}$$

From (3) and (2) we obtain:

$$w \Longrightarrow^* w_1zaaw_2 \Longrightarrow^* \beta baa\gamma$$

which is a contradiction. □

In way similar to Theorem 5.3 it is possible to show several non-completeness results for one-sided insertion-deletion systems. Table 3 summarizes these results. We remark that systems having smaller parameters, like systems of size $(1, 1, 0; 1, 1, 0)$ are also not complete.

Table 3. Computationally non-complete one-sided insertion-deletion systems

Size	Witness language	Reference
(1,1,0;1,1,1)	$(ba)^+$	[20]
(1,1,1;1,1,0)	a^nb^n, $n \geq 0$	[26]
(1,1,0;2,0,0)	$(ba)^+$	[19]
(2,0,0;1,1,0)	$(ba)^+$	[19]

Moreover, in the case of systems of size $(1, 1, 0; 1, 1, 0)$ it is possible to show that the language generated by such insertion-deletion systems is a particular subclass of the family of context-free languages.

This class of languages is non-trivial because even a smaller subclass, $INS_1^{1,0}DEL_0^{0,0}$, contains non-regular context-free languages, see Example 5.1.

Theorem 5.4. $INS_1^{1,0}DEL_0^{0,0} \cap (CF \setminus REG) \neq \emptyset$.

In [19] it is shown that the effect of deletion rules can be precomputed. This gives the following result.

Theorem 5.5. $INS_1^{1,0}DEL_1^{1,0} \subset CF$.

6 Graph-controlled insertion-deletion systems

In previous sections it was shown that there are classes of insertion-deletion systems that cannot generate RE. Making an analogy to context-free grammars, a natural extension of insertion-deletion systems using the graph-controlled or programmed approach can be done. Such model introduces states (or labels of the program) associated to every insertion or deletion rule. The transition is performed by applying corresponding rule and choosing the new state (thus the rule to be applied) among a specific set of rules. Another definition of this model in the style of [30] or [5] can be done. This definition supposes that there are disjoint groups of insertion and deletion rules (corresponding to *membranes* from [30] or *components* from [5]). The transition is performed by firstly choosing and applying one of applicable rules from the current group and switching to the next group indicated in the rule description.

6.1 Formal definition

A *graph-controlled insertion-deletion system* is a construct

$$\Pi = (V, T, A, H, I_0, I_f, R) \text{ where}$$

- V is a finite alphabet,
- $T \subseteq V$ is the *terminal alphabet*,
- $A \subseteq V^*$ is a finite set of *axioms*,
- H is a set of labels associated (in a one-to-one manner) to the rules in R,
- $I_0 \subseteq H$ is the set of *initial labels*,
- $I_f \subseteq H$ is the set of *final labels*, and
- R is a finite set of rules of the form $l : (r, E)$ where r is an insertion or deletion rule over V and $E \subseteq H$.

As it is common for graph controlled systems, a configuration of Π is represented by a pair (w, i), where i is the label of the rule to be applied and w is the current string. A transition $(w, i) \Rrightarrow (w', j)$ is performed if there is a rule $l : ((u, \alpha, v)_t, E)$ in R such that $w \Longrightarrow_t w'$ by the insertion/deletion rule $(u, \alpha, v)_t$, $t \in \{ins, del\}$, and $j \in E$. The result of the computation consists of all terminal strings reaching a final label from an axiom and the initial label, *i.e.*,

$$L(\Pi) = \{w \in T^* \mid (w', i_0) \Rrightarrow^* (w, i_f) \text{ for some } w' \in A,\ i_0 \in I_0,\ i_f \in I_f\}.$$

We will use another rather similar definition for a graph-controlled insertion-deletion system, thereby assigning groups of rules to *components* of the system:

A *graph-controlled insertion-deletion system with k components* is a construct

$$\Pi = (k, V, T, A, H, i_0, i_f, R) \text{ where}$$

- k is the number of components,
- V, T, A, H are defined as for graph-controlled insertion-deletion systems,
- $i_0 \in [1..k]$ is the initial component,
- $i_f \in [1..k]$ is the final component, and
- R is a finite set of rules of the form $l : (i, r, j)$ where r is an insertion or deletion rule over V and $i, j \in [1..k]$.

The set of rules R may be divided into sets R_i assigned to the *components* $i \in [1..k]$, *i.e.*, $R_i = \{l : (r, j) \mid l : (i, r, j) \in R\}$; in a rule $l : (i, r, j)$, the number j specifies the *target component* where the string is sent from component i after the application of the insertion or deletion rule r. A configuration of Π is represented by a pair (w, i), where i is the number of the *current* component (initially i_0) and w is the current string. We also say that w is *situated* in component i. A transition $(w, i) \Rrightarrow (w', j)$ is performed as follows: first, a rule $l : (r, j)$

from component i (from the set R_i) is chosen in a non-deterministic way, the rule r is applied, and the string is moved to component j; hence, the new set from which the next rule to be applied will be chosen is R_j. More formally, $(w,i) \Rrightarrow (w',j)$ if there is $l : ((u,\alpha,v)_t, j) \in R_i$ such that $w \Longrightarrow_t w'$ by the rule $(u,\alpha,v)_t$; we also write $(w,i) \Rrightarrow_l (w',j)$ in this case. The result of the computation consists of all terminal strings situated in component i_f reachable from the axiom and the initial component, *i.e.*,

$$L(\Pi) = \{w \in T^* \mid (w', i_0) \Rrightarrow^* (w, i_f) \text{ for some } w' \in A\}.$$

It is not difficult to see that graph-controlled insertion-deletion systems with k components are a special case of graph-controlled insertion-deletion systems. Without going into technical details, we just give the main ideas how to obtain a graph-controlled insertion-deletion system from a graph-controlled insertion-deletion system with k components: for every $l : ((u,\alpha,v)_t, j) \in R_i$ we take a rule $l : (i, (u,\alpha,v)_t, Lab(R_j))$ into R where $Lab(R_j)$ denotes the set of labels for the rules in R_j; moreover, we take $I_0 = Lab(R_{i_0})$ and $I_f = Lab(R_{i_f})$. Finally, we remark that the labels in a graph-controlled insertion-deletion system with k components may even be omitted, but they are useful for specific proof constructions. On the other hand, by a standard powerset construction for the labels (as used for the determinization of non-deterministic finite automata) we can easily prove the converse inclusion, *i.e.*, that for any graph-controlled insertion-deletion system we can construct an equivalent graph-controlled insertion-deletion system with k components.

We define the *communication graph* of a graph-controlled insertion-deletion system with k components to be the graph with nodes $1, \ldots, k$ having an edge between node i and j if and only if there exists a rule $l : ((u,\alpha,v)_t, j) \in R_i$. In [30], 5.5, special emphasis is laid on graph-controlled insertion-deletion systems with k components whose communication graph has a tree structure, as we observe that the presentation of graph-controlled insertion-deletion systems with k components given above in the case of a tree structure is rather similar to the definition of insertion-deletion P systems as given in [30]; the main differences are

that in P systems the final component i_f contains no rules and corresponds with the root of the communication tree; on the other hand, in graph-controlled insertion-deletion system with k components, each of the axioms can only be situated in the initial component i_0, whereas in P systems we may situate each axiom in various different components.

Throughout the rest of this section we shall only use the notion of graph-controlled insertion-deletion systems with k components, as they are easier to handle and sufficient to establish computational completeness in the proofs of our main results presented in the succeeding section. By $GCID_k(ins_n^{m,m'}, del_p^{q,q'})$ we denote the family of languages $L(\Pi)$ generated by graph-controlled insertion-deletion systems with at most k components and insertion and deletion rules of size at most $(n, m, m'; p, q, q')$. We replace k by $*$ if k is not fixed. The letter "G" is replaced by the letter "T" to denote classes whose communication graph has a *tree structure*. Some results for the families $TCID_k(ins_n^{m,m'}, del_p^{q,q'})$ can directly be derived from the results presented in [19, 30], for the corresponding families of insertion-deletion P systems $ELSP_k(ins_n^{m,m'}, del_p^{q,q'})$, yet the results we present in the succeeding section either reduce the number of components for systems with an underlying tree structure or else take advantage of the arbitrary structure of the underlying communication graph thus obtaining computational completeness for new restricted variants of insertion and deletion rules.

Example 6.1.

Consider the following graph-controlled insertion-deletion system $\Pi = (3, T, T, \lambda, H, 1, 1, R)$, with $T = \{a, b, c\}$, $H = \{1, 2, 3\}$ and $R = R_1 \cup R_2 \cup R_3$, where $R_1 = \{1 : ((\lambda, a, \lambda)_{ins}, 2)\}$, $R_2 = \{2 : ((\lambda, b, \lambda)_{ins}, 3)\}$, $R_3 = \{3 : ((\lambda, c, \lambda)_{ins}, 1)\}$.

The system is inserting consecutively a, b and c. Therefore it is clear that $L(\Pi) = \{w \in \{a, b, c\}^* : |w|_a = |w|_b = |w|_c\}$, which is not a context-free language.

We remark that using two nodes, it is possible to similarly generate the non-regular language $L = \{w \in \{a, b\}^* : |w|_a = |w|_b\}$. The communication graph has the form of a tree in this case.

6.2 Results

We start with the following result from [1].

Theorem 6.2. $PsTCID_*(ins_1^{0,0}, del_1^{0,0}) \subseteq PsGCID_*(ins_1^{0,0}, del_1^{0,0}) = PsMAT$.

However, in terms of the generated language such systems are not very powerful. Like in the case of context-free insertion-deletion systems there is no control on the position of insertion. Hence, the language $L = \{a^*b^*\}$ cannot be generated, for insertion strings of any size. Hence we obtain:

Theorem 6.3. $REG \backslash GCID_*(ins_n^{0,0}, del_1^{0,0}) \neq \emptyset$, *for any* $n > 0$.

However, there are non-context-free languages that can be generated by such systems (even without deletion). From Example 6.1 we obtain:

Theorem 6.4. $GCID_*(ins_1^{0,0}, del_0^{0,0}) \setminus CF \neq \emptyset$.

A more general inclusion holds:

Theorem 6.5. *[1]* $GCID_*(ins_n^{0,0}, del_1^{0,0}) \subset MAT$, *for any* $n > 0$.

Next theorem shows that graph-controlled insertion-deletion systems are strictly more powerful than ordinary insertion-deletion systems of the same size.

Theorem 6.6. *[21]* $TCID_5(ins_1^{1,0}, del_1^{1,0}) = RE$.

The proof is based on the following idea. Any rule $AB \to CD$ of a type-0 grammar in Kuroda normal form can be simulated in 4 stages: (1) erasing A, (2) erasing B, (3) inserting D and (4) inserting C. Every operation can be done by a dedicated component with the help of an additional symbol that marks the position before A and that is used in all operations. A typical computation may look as follows:

$$w_1ABw_2 \Rrightarrow w_1P_iABw_2 \Rrightarrow w_1P_iBw_2 \Rrightarrow w_1P_iw_2 \Rrightarrow$$
$$w_1P_iDw_2 \Rrightarrow w_1P_iCDw_2 \Rrightarrow w_1CDw_2$$

Other rules of the grammar can be simulated in a similar manner. We leave technical details that can be consulted in [21].

In a similar way it is possible to obtain a characterization of RE languages by the family $TCID_5(ins_1^{1,0}, del_1^{0,1})$, *i.e.* with contexts for insertion and deletion on different sides. Taking also into account the symmetrical cases we get:

Corollary 6.7. $TCID_5(ins_1^{1,0}, del_1^{0,1}) = TCID_5(ins_1^{0,1}, del_1^{1,0}) = TCID_5(ins_1^{0,1}, del_1^{0,1}) = RE.$

Using a similar technique it is possible to prove following theorems (see [22]).

Theorem 6.8. $TCID_5(ins_1^{1,0}, del_2^{0,0}) = TCID_5(ins_1^{0,1}, del_2^{0,0}) = RE.$

Theorem 6.9. $TCID_5(ins_2^{0,0}, del_1^{1,0}) = TCID_5(ins_2^{0,0}, del_1^{0,1}) = RE.$

However, in some cases graph-controlled insertion-deletion systems are still not complete.

Theorem 6.10. *[22]* $REG \setminus GCID_*(ins_2^{0,0}, del_2^{0,0}) \neq \emptyset.$

6.3 Graph-controlled insertion-deletion systems with priorities

A further control can be added to graph-controlled insertion-deletion systems by introducing a priority of deletion over insertion, *i.e.*, if deletion and insertion rules are applicable, then one of deletion rules will be chosen. This condition can also be viewed as a particular case of the graph-controlled insertion-deletion systems if the latter have rules with appearance checking. We denote by $TCID_k(ins_n^{m,m'} < del_p^{q,q'})$ the families of languages generated by corresponding classes.

Using priorities it is possible to further decrease the length of contexts needed for computational completeness. It is quite astonishing that insertion-deletion systems that insert or delete one symbol in a context-free manner can generate $PsRE$. In case of general communication graph this is particularly easy to see: jumping to an instruction

of a register machine corresponds to switching to the associated component, and the entire construction is a composition of graphs shown in Fig. 1. The decrement instruction works correctly because of priority of deletion over insertion. A configuration $(p, x_1, \cdots, x_n)$ of a register machine is encoded by strings $Perm(pA_1^{x_1} \cdots A_n^{x_n})$.

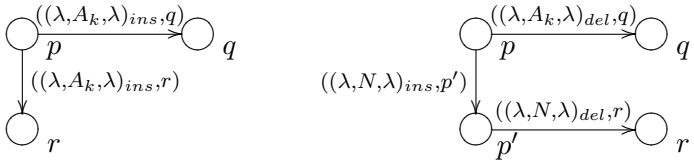


Figure 1. Simulating (p, A_k+, q, r)(left) and (p, A_k-, q, r) (right).

For the tree-like communication graph, the proof is more sophisticated and needs a communication graph depicted at Fig. 2. The main idea is to use a rule $((\lambda, p, \lambda)_{del}, p_1^+)$ if p is an increment instruction or $((\lambda, p, \lambda)_{del}, p_1^-)$ if p is a decrement instruction and redirect the computation to corresponding components that simulate only one instruction of the register machine. This gives:

Theorem 6.11. $PsTCID_*(ins_1^{0,0} < del_1^{0,0}) = PsRE.$

Although the above theorem shows that corresponding systems are quite powerful, they cannot generate RE without control on the place where a symbol is inserted ($REG \backslash GCID_*(ins_n^{0,0} < del_1^{0,0}) \neq \emptyset$ for any $n > 0$, see Theorem 6.3). Once we allow a context in insertion or deletion rules, they can do it.

Theorem 6.12. $TCID_*(ins_1^{0,1} < del_1^{0,0}) = RE.$

In a similar way the following result can be obtained.

Theorem 6.13. $TCID_*(ins_1^{0,0} < del_1^{1,0}) = RE.$

However in this case the proof is more technical and needs additional components, see [1]. A similar can be done with a context-free deletion of two symbols.

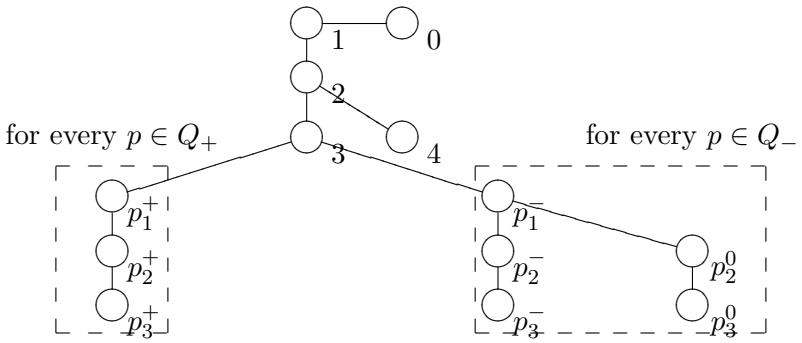


Figure 2. Communication graph for Theorem 6.11. The structures in the dashed rectangles are repeated for every instruction of the register machine.

Theorem 6.14. $TCID_*(ins_1^{0,0} < del_2^{0,0}) = RE$.

We mention that the counterpart of Theorem 6.14 obtained by interchanging parameters of insertion and deletion rules is not true, see Theorem 6.3.

7 Using only insertion

In this section we consider systems which only use the operation of insertion, *i.e.*, there are no deletion rules. We shall use the notation $INS_n^{m,m'}$ in order to denote families of languages generated by insertion-only systems. It is known that the classes of *insertion languages* are incomparable with many known language classes. For example, consider a linear language $\{a^nba^n \mid n \geq 1\}$. This language cannot be generated by any insertion system (see Theorem 6.6 in [32]).

In order to be complete it is possible to use some codings to "interpret" the generated strings. In the literature several types of codings were considered. It is possible to consider the following languages as a result for an insertion system I:

1. $h(L(I) \cap R)$, where h is a morphism and R is a special language (as considered in [28, 31]), or
2. $\varphi(h^{-1}(L(I)))$, where h is a morphism and φ is a weak coding (considered in [25, 32, 15]).

We mention that both types of codings are rather simple and can be simulated by a finite state transducer, provided that R is regular. In some cases R is considered to be the Dyck language.

We start with the following representation of regular languages by using insertion systems and star languages. We recall that the family $STAR = \{A^* \mid A \in FIN\}$ of *star* languages is a subfamily of regular languages.

Theorem 7.1. *[31] Any regular language L can be represented in the form $L = h(L' \cap R)$, where h is a weak coding, $L' \in INS_2^{0,0}$, and R is a star language.*

Let W represent the family of weak codings. We mention that the inclusion $REG \subset W(INS_2^{0,0} \cap STAR)$ is proper, because the Dyck language is in $INS_2^{0,0}$.

A similar characterization of context-free languages by the means of insertion systems can be done.

Theorem 7.2. *[18] A language L is context-free if and only if it can be represented in the form $L = \varphi(h^{-1}(L'))$ where $L' \in INS_3^{1,1}$, φ is a weak coding and h is a morphism.*

We remark that it is important to use a coding:

Theorem 7.3. *[32] $INS_*^{1,1} \subseteq CF$.*

We present below several characterizations of recursively enumerable languages by the means of insertion systems. We start with the following result.

Theorem 7.4. *[15, 27] Each language $L \in RE$ can be written as $L = \varphi(h^{-1}(L'))$, where φ is a weak coding, h is a morphism, and $L' \in INS_3^{3,3}$.*

Sketch. The idea of the proof is to apply "mark and migrate" technique in order to simulate a type-0 grammar. According to this technique, symbols that have been rewritten are marked. In the following a special symbol # called *marking symbol* will be used. We say that a letter a is *marked* in a sentential form waw' if it is followed by #, *i.e.*, $|w'| > 0$, and # is the prefix of w'. For example, in order to simulate a context-free production $A \to BC$, the string $\#BC$ is inserted immediately at the right of A, assuming that A was not marked before. As soon as the derivation of the simulated sentential form is completed, every nonterminal A is marked, and the inverse morphism is applied to the pairs $A\#$.

In order to simulate context-sensitive productions of the form $AB \to CD$, the *migration* of symbols is applied. This means that if a pair AB that should be used by the production is separated by one or more marked symbols, then copies of symbol A are inserted to the right, using the marked symbols as contexts. In this way, the symbol A can migrate to the right and become adjacent to B. When only the terminal symbols are unmarked in the resulted sentential form, the inverse morphism h^{-1} and the weak coding may be applied in order to eliminate marking symbols and nonterminals. □

Corollary 7.5. *[15] Every language $L \in RE$ can be represented in either of the forms $L = L' \setminus R$, $L = L'/R'$, where $L' \in INS_3^{3,3}$, R, R' are regular languages, and $\setminus R, /R'$ denote the left and right quotient with R, R' respectively.*

In a similar way it is possible to obtain a characterization of RE by replacing the inverse morphism h^{-1} by the intersection with a regular language. It is shown in [28] that in order to obtain this characterization it is enough to use strictly $k-$testable languages (denoted by $LOC(k)$), which is a strictly subset of the family of regular languages, for $k \geq 2$. We recall that a language L is a strictly $k-$testable language over T if there are finite sets $Pref, Suf, Int \subseteq T^k$, and for every w, $w \in L$ if and only if (a) the prefix of w of length k belongs to $Pref$, (b) the suffix of w of length k belongs to Suf, and (c) every proper subsequence of w of length k belongs to Int.

Then, the following theorem holds.

Theorem 7.6. *[28] Every language $L \in RE$ can be represented in the form $h(L' \cap R)$, where h is a projection, $L' \in INS_3^{3,3}$, and $R \in LOC(2)$.*

The next theorem considers a different approach showing that insertion systems with context-free rules are quite powerful. Since the *mark and migrate* technique cannot be used in this case, the filtering of sentential forms that have the "proper structure" is performed by an intersection with the Dyck language.

Theorem 7.7. *[31] Every language $L \in RE$ can be represented in the form $L = h(L' \cap \mathcal{D})$, where $L' \in INS_3^{0,0}$, h is a projection, and $\mathcal{D}$ is the Dyck language.*

Finally, we remark that in the case of graph-controlled insertion systems it is possible to decrease the sizes of the contexts.

Theorem 7.8. *[18] Every language $L \in RE$ can be represented in the form $L = \varphi(h^{-1}(L'))$, where φ is a weak coding, h is a morphism, and $L' \in TCID_3(ins_2^{2,2} del_0^{0,0})$.*

8 Bibliographical remarks

Insertion systems, without using the deletion operation, were first considered in [8], however the idea of the context adjoining was exploited long time before by [23]. Context-free insertion systems as a generalization of concatenation were first considered in [9, 10]. A formal language study of both context-free insertion and deletion operations was done in [13], however the operations were considered separately. The articles [7, 11] investigate the power of the insertion and deletion operations. Both operations were first considered together in [16] and related formal language investigations can be found in several places; we mention only [25] and [29]. The biological motivation of insertion-deletion operations leaded to their study in the framework of molecular computing, see, for example, [6], [14], [32], [35]. An interesting study of the deletion operation can be found in [7].

The universality of context-free insertion-deletion systems of size $(2, 0, 0; 3,\ 0, 0)$ and $(3, 0, 0; 2, 0, 0)$ was shown in [24], while the optimality of this result was shown in [37]. The last article suggested to consider the sizes of each context as a complexity measure and not the maximum as it was done before. One-sided insertion-deletion systems were firstly considered in [26] and the graph-controlled variant in [21]. Graph-controlled insertion-deletion systems with priorities were introduced in [1].

Other variants of the insertion operation and different control mechanisms can be found in [13, 12, 4].

References

[1] A. Alhazov, A. Krassovitskiy, Y. Rogozhin, and S. Verlan. P systems with minimal insertion and deletion. In R. Gutiérrez-Escudero, M. A. Gutiérrez-Naranjo, G. Păun, I. Pérez-Hurtado, and A. Riscos-Nunez, editors, *Proc. of Seventh Brainstorming Week on Membrane Computing Sevilla, February 2–6, 2009*. Fénix Editora, Sevilla, 2009, volume I, 9–21. Also accepted to *Theoretical Computer Science*.

[2] A. Alhazov, A. Krassovitskiy, Y. Rogozhin, and S. Verlan. *New Trends in Formal Language Theory Inspired by Natural Computing: Small Size Insertion and Deletion Systems*, Imperial College Press, chapter 9. Mathematics, Computing, Language, and Life: Frontiers in Mathematical Linguistics and Language Theory. 2010, 459–524. In publication.

[3] R. Benne. *RNA-Editing: The Alteration of Protein Coding Sequences of RNA*. Ellis Horwood, Chichester, West Sussex, 1993.

[4] F. Biegler, M. J. Burrell, and M. Daley. Regulated rna rewriting: Modelling rna editing with guided insertion. *Theor. Comput. Sci.*, **387**(2), (2007), 103 – 112. Descriptional Complexity of Formal Systems.

[5] E. Csuhaj-Varjú and J. Dassow. On cooperating/distributed grammar systems. *Elektronische Informationsverarbeitung und Kybernetik*, **26**(1/2), (1990), 49–63.

[6] M. Daley, L. Kari, G. Gloor, and R. Siromoney. Circular contextual insertions/deletions with applications to biomolecular computation. In *SPIRE/CRIWG*. 1999, 47–54.

[7] M. Domaratzki and A. Okhotin. Representing recursively enumerable languages by iterated deletion. *Theoretical Computer Science*, **314**(3), (2004), 451–457.

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

[9] D. Haussler. *Insertion and Iterated Insertion as Operations on Formal Languages*. Ph.D. thesis, Univ. of Colorado at Boulder, 1982.

[10] D. Haussler. Insertion languages. *Information Sciences*, **31**(1), (1983), 77–89.

[11] M. Ito, L. Kari, and G. Thierrin. Insertion and deletion closure of languages. *Theor. Comput. Sci.*, **183**(1), (1997), 3–19.

[12] M. Ito and R. Sugiura. n-insertion on languages. In N. Jonoska, G. Paun, and G. Rozenberg, editors, *Aspects of Molecular Computing*. Springer, 2004, volume 2950 of *Lecture Notes in Computer Science*, 213–218.

[13] L. Kari. *On Insertion and Deletion in Formal Languages*. Ph.D. thesis, University of Turku, 1991.

[14] L. Kari, G. Păun, G. Thierrin, and S. Yu. At the crossroads of dna computing and formal languages: Characterizing RE using insertion-deletion systems. In *Proc. of 3rd DIMACS Workshop on DNA Based Computing*. Philadelphia, 1997, 318–333.

[15] L. Kari and P. Sosík. On the weight of universal insertion grammars. *Theor. Comput. Sci.*, **396**(1-3), (2008), 264–270.

[16] L. Kari and G. Thierrin. Contextual insertions/deletions and computability. *Information and Computation*, **131**(1), (1996), 47–61.

[17] S. C. Kleene. Representation of events in nerve nets and finite automata. In C. Shannon and J. McCarthy, editors, *Automata Studies*, Princeton University Press, Princeton, NJ. 1956, 3–41.

[18] A. Krassovitskiy. On the power of insertion P systems of small size. In *Proc. of Seventh Brainstorming Week on Membrane Computing*. 2009, 29–44.

[19] A. Krassovitskiy, Y. Rogozhin, and S. Verlan. Computational power of insertion-deletion (P) systems with rules of size two. Accepted to *Natural Computing*. 2010, in publication.

[20] A. Krassovitskiy, Y. Rogozhin, and S. Verlan. Further results on insertion-deletion systems with one-sided contexts. In C. Martín-Vide, F. Otto, and H. Fernau, editors, *Language and Automata Theory and Applications, Second International Conference, LATA 2008, Tarragona, Spain, March 13-19, 2008. Revised Papers*. Springer, 2008, volume 5196 of *Lecture Notes in Computer Science*, 333–344.

[21] A. Krassovitskiy, Y. Rogozhin, and S. Verlan. One-sided insertion and deletion: Traditional and P systems case. In E. Csuhaj-Varjú, R. Freund, M. Oswald, and K. Salomaa, editors, *International Workshop on Computing with Biomolecules, August 27th, 2008, Wien, Austria*. Druckerei Riegelnik, 2008, 53–64.

[22] A. Krassovitskiy, Y. Rogozhin, and S. Verlan. Computational power of P systems with small size insertion and deletion rules. In T. Neary, D. Woods, A. K. Seda, and N. Murphy, editors, *Proceedings International Workshop on The Complexity of Simple Programs, Cork, Ireland, 6-7th December 2008*. 2009, volume 1 of *EPTCS*, 108–117.

[23] S. Marcus. Contextual grammars. *Revue Roumaine de Mathmatique Pures et Appliques*, **14**, (1969), 1525–1534.

[24] M. Margenstern, G. Păun, Y. Rogozhin, and S. Verlan. Context-free insertion-deletion systems. *Theoretical Computer Science*, **330**(2), (2005), 339–348.

[25] C. Martín-Vide, G. Păun, and A. Salomaa. Characterizations of recursively enumerable languages by means of insertion grammars. *Theoretical Computer Science*, **205**(1-2), (1998), 195–205.

[26] A. Matveevici, Y. Rogozhin, and S. Verlan. Insertion-deletion systems with one-sided contexts. In J. O. Durand-Lose and M. Margenstern, editors, *Machines, Computations, and Universality, 5th International Conference, MCU 2007, Orléans, France, September 10-13, 2007, Proceedings*. Springer, 2007, volume 4664 of *Lecture Notes in Computer Science*, 205–217.

[27] K. Onodera. A note on homomorphic representation of recursively enumerable languages with insertion grammars. *Transactions of Information Processing Society of Japan*, **44**(5), (2003), 1424–1427.

[28] K. Onodera. New morphic characterizations of languages in Chomsky hierarchy using insertion and locality. In A. H. Dediu, A.-M. Ionescu, and C. Martín-Vide, editors, *Language and Automata Theory and Applications, Third International Conference, LATA 2009, Tarragona, Spain, April 2-8, 2009.* Springer, 2009, volume 5457 of *Lecture Notes in Computer Science*, 648–659.

[29] G. Păun. *Marcus Contextual Grammars*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[30] G. Păun. *Membrane Computing. An Introduction.* Springer–Verlag, 2002.

[31] G. Păun, M. J. Pérez-Jiménez, and T. Yokomori. Representations and characterizations of languages in Chomsky hierarchy by means

of insertion-deletion systems. *Int. J. Found. Comput. Sci.*, **19**(4), (2008), 859–871.

[32] G. Păun, G. Rozenberg, and A. Salomaa. *DNA Computing: New Computing Paradigms*. Springer, 1998.

[33] G. Rozenberg and A. Salomaa. *Handbook of Formal Languages, 3 volumes*. Springer Verlag, Berlin, Heidelberg, New York, 1997.

[34] W. D. Smith. DNA computers in vitro and in vivo. In R. Lipton and E. Baum, editors, *Proceedings of DIMACS Workshop on DNA Based Computers*. Amer. Math. Society, 1996, DIMACS Series in Discrete Math. and Theoretical Computer Science, 121–185.

[35] A. Takahara and T. Yokomori. On the computational power of insertion-deletion systems. In M. Hagiya and A. Ohuchi, editors, *DNA Computing, 8th International Workshop on DNA Based Computers, DNA8, Sapporo, Japan, June 10-13, 2002, Revised Papers*. 2002, volume 2568 of *Lecture Notes in Computer Science*, 269–280.

[36] A. Takahara and T. Yokomori. On the computational power of insertion-deletion systems. *Natural Computing*, **2**(4), (2003), 321–336.

[37] S. Verlan. On minimal context-free insertion-deletion systems. *Journal of Automata, Languages and Combinatorics*, **12**(1-2), (2007), 317–328.

Sergey Verlan,

Received June 10, 2010

LACL, University of Paris Est,
61, av gen. de Gaulle
94010 Creteil, France
Phone: +33145176600
E–mail: $verlan@univ-paris12.fr$

Institute of Mathematics and Computer Science,
Academy of Sciences of Moldova,
Academiei, 5, MD-2028, Chisinau, Moldova