A 15-Year Retrospective on Insertion-Deletion Systems: Progress, Evolution, and Future Directions

Artiom Alhazov, Sergiu Ivanov, Sergey Verlan

Abstract

This paper offers a comprehensive retrospective on the development and results in the field of insertion-deletion systems over the past 15 years, building upon an earlier foundational overview from 2010. We review key theoretical developments, new extensions, and variations that have emerged, examining their impact on the computational power of insertion-deletion systems. This retrospective aims to update the field's understanding of the model, providing insights into the latest progress and identifying promising directions for future research.

Keywords: insertion-deletion systems, computational completeness, decidability, regulated rewriting.

MSC 2020: 68Q42, 68Q45, 68Q05, 68Q10, 03D05.

1 Introduction

The insertion/deletion operation involves adding or removing a substring within a specified context. It is defined by a triple (u, x, v), where x is the substring that can be inserted between u and v or deleted from the substring uxv. So, insertion corresponds to the string rewriting rule $uv \to uxv$, while deletion corresponds to $uxv \to uv$. While these operations can be viewed as part of string rewriting, they have a distinct history due to their relative simplicity: they decouple the action of rewriting that deletes and inserts in one step.

Moreover, these operations naturally occur in various fields, such as linguistics [1], formal language theory [2]–[4], protection systems [5],

doi:10.56415/csjm.v32.18

restarting automata [6], and DNA computing [7]–[11], see Section 2.3 for more details. This last area concentrates much of the research efforts on the operations of insertion and deletion, that are one of the main models in this domain. As pointed out in [12], these operations correspond to the process of mismatched annealing of DNA strands. Similarly, in the case of RNA editing, the Uracyl base (U) is inserted or deleted in some left context [13], giving the inspiration for so-called one-sided insertion-deletion operations. Recently, another biological process has been discovered that uses insertion and deletion to edit the genome forming, being the base for the CRISPR-Cas9 technology [14], [15].

A finite set of insertion-deletion rules, along with a set of axioms, forms an insertion-deletion system. This system functions as a language-generating device: by 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 an insertion-deletion system is defined as the tuple $(n, m, m'; p, q, q')$, where n is the length of the longest inserted string, m is at least the length of the longest left insertion context, and m' is at least the length of the longest right insertion context, whilst the values p, q, and q' describe the same parameters for deletion.

From the very beginning of the research on insertion-deletion systems, the primary focus has been on investigating the computational completeness of the model in relation to various descriptional complexity parameters, mainly with respect to the size. With few exceptions, all symmetric combinations of size parameters have been demonstrated to achieve computational completeness. In 2007, Yurii Rogozhin found the first non-complete insertion-deletion system family having the size $(1, 1, 1; 1, 1, 0)$ [10]. This result, as well as the follow-up result on the non-completeness of systems of size $(1, 1, 0; 1, 1, 1)$ [11], opened a completely new perspective in the study of insertion-deletion systems by allowing to consider such rules in a regulated rewriting framework, e.g., using graph or matrix control mechanisms. This became the main research direction in this area in the last 15 years, see [16] for a detailed overview.

In this paper, we provide a quick overview of the main results in

the area of insertion-deletion systems, focusing on the progress made in the last 15 years. The paper builds on the foundational overview from 2010 [17] that describes main proof techniques in the area (for that time), as well as on the 2022 overview of regulated insertion-deletion systems [16]. Compared to the two mentioned overviews, we aim to create an up-to-date reference of the existing results for the whole area of insertion-deletion systems and to explore some intriguing ideas employed in proofs, as well as to highlight several original models and links to the other areas.

The paper is organized as follows. In Section 2, we recall some basic notions, the definition of insertion-deletion systems and recall the related models. In Section 3, we present new proof methods used in the area of insertion-deletion systems. In Section 4, we recall the best known results on insertion-deletion systems, like matrix, graph-controlled or semi-conditional insertion-deletion systems. Section 5 is devoted to results on regulated insertion-deletion systems. In Section 6, we discuss the model of insertion-deletion with substitutions. Section 7 presents the simulator of insertion-deletion systems. Finally, in Section 8, we draw some conclusions and outline some future research directions.

2 Definitions

In this section, we recall some basic notions and definitions used in formal language theory. For more details on formal language theory, we refer to textbooks like [18].

An *alphabet* V is a finite non-empty set of abstract *symbols*. The set of all strings over the alphabet V is denoted by V^*. The *empty string* is denoted by λ and the set of non-empty strings over V is denoted by V^+. A string x is said to be a substring of a string w if $w = uxv$. The length of a string w is denoted by $|w|$. The cardinality of a set M is denoted by $|M|$.

The family of regular, linear, context-free, monotone, and recursively enumerable languages is denoted by REG, LIN, CF, MON, and RE, respectively.

2.1 Geffert Normal Form

A type-0 grammar $G = (\{S\} \cup N_T, T, S, P)$ is said to be in *Geffert normal form* [19] if the set of non-terminals N_T is defined as $N_T = \{A, B, C, D\}$, T is an alphabet (of terminal symbols) and P only contains context-free rules $S \to uSv$ with $u \in (T \cup \{A, C\})^*$ and $v \in \{B, D\}^*$, as well as $S \to \lambda$, and two (non-context-free) erasing rules $AB \to \lambda$ and $CD \to \lambda$.

As shown in [19], there might be several other forms of non-context-free rules, e.g., only $ABC \to \lambda$, see [19] for more details.

The rules $S \to uSv,\ u, v \in (N_T \cup T)^*$, and $S \to \lambda$ may be considered as linear rules over the terminal alphabet $N_T \cup T$. Based on the proofs given in [19], various special variants for the linear rules over the terminal alphabet $N_T \cup T$ have been elaborated, for example, see [20]–[22].

We would like to note that the derivation of a terminal string in a grammar being in Geffert normal form is done in three stages. During the first two stages the "linear" rules are applied. In the first stage, only productions having strings $u \in T^*$ are applied and during the second stage only productions having $u \in N_T^*$ are applied. The non-context-free erasing rules are not applicable during these two stages. The third stage begins after the application of the rule $S \to \lambda$ and during this stage only the cooperative erasing rules can be applied anymore.

In the area of insertion-deletion systems, a particular variant of Geffert normal form is used, namely the *special Geffert normal form* (SGNF). In SGNF, the "linear" rules $S \to uSv$ from the Geffert normal form are transformed into a set of "right-" and "left-linear" rules using a standard approach, e.g., see [22], [23] for more details. These "right-" and "left-linear" rules are of the forms $X \to bY$ and $X \to Yb$ with $X, Y \in N$ and $b \in N_T \cup T$, where N is the new alphabet of non-terminal symbols including S. The rule $S \to \lambda$ is replaced by a new rule $S' \to \lambda$, where $S' \in N$. According to this construction, in the first two stages of a derivation in a grammar in SGNF, exactly one non-terminal symbol from N is present, and the third stage starts with applying $S' \to \lambda$, after what only the chosen non-context-free cooperative erasing rules can be applied.

Several variants of SGNF have been proposed having interesting properties for proofs, we refer to Section 2.1 and [24] for more details.

2.2 Insertion-Deletion Systems

Definition 1. Let V be an alphabet, and let $x \in V^+$, as well as $u, v \in V^*$. Then the triple $(u, x, v)_{ins}$ is called an *insertion* rule, and the triple $(u, x, v)_{del}$ is called a *deletion* rule.

The application of the rule $r : (u, x, v)_{ins}$ (labeled by r) to a string w yields the string w' if $w = zuvz'$ and $w' = zuxvz'$, for some $z, z' \in V^*$, and we write $w \Longrightarrow_r w'$. Similarly, the application of the rule $r' : (u, x, v)_{del}$ to a string w yields the string w' if $w = zuxvz'$ and $w' = zuvz'$, for some $z, z' \in V^*$, and we write $w \Longrightarrow_{r'} w'$. Hence, r corresponds to the rewriting rule $uv \to uxv$, and r' corresponds to the rule $uxv \to uv$. However, we would like to note that for an insertion rule, both contexts u and v are allowed to be empty, which is not allowed in traditional Chomsky grammars; however, this behavior can be simulated by using an additional symbol for context marking.

As usual, by $\Longrightarrow_R$ we denote the set $\{\Longrightarrow_r | \ r \in R\}$ and the transitive and reflexive closure of $\Longrightarrow_R$ by $\Longrightarrow_R^*$.

Definition 2. An *insertion-deletion system* is a quadruple (V, T, A, R), where

- V is an alphabet,
- $T \subseteq V$ is the terminal alphabet,
- $A \subseteq V^*$ is a finite set of initial strings (axioms),
- $R \subseteq V^* \times V^+ \times V^* \times \{ins, del\}$ is a finite set of insertion and deletion rules.

The language generated by the system $\Gamma = (V, T, A, R)$ is defined as follows:

$$L(\Gamma) = \{x \in T^* \mid \exists y \in A : y \Longrightarrow_R^* x\}.$$

The size of an insertion-deletion system $\Gamma = (V, T, A, R)$ is defined as the tuple $(n, m, m'; p, q, q')$, where

$$n = \max\{|x| \mid (u, x, v)_{ins} \in R\} \qquad p = \max\{|x| \mid (u, x, v)_{del} \in R\}$$
$$m = \max\{|u| \mid (u, x, v)_{ins} \in R\} \qquad q = \max\{|u| \mid (u, x, v)_{del}, \in R\}$$
$$m' = \max\{|v| \mid (u, x, v)_{ins} \in R\} \qquad q' = \max\{|v| \mid (u, x, v)_{del} \in R\}$$

By $INS_n^{m,m'} DEL_p^{q,q'}$ we denote the family of languages generated by insertion-deletion systems of size $(n, m, m'; p, q, q')$.

We call systems of size $(n, 0, 0; p, 0, 0)$ *context-free* insertion-deletion systems. Such systems are particular because they do not use any context — this means that any (insertion) operation can happen anytime at any place in the string. This variant initially appeared as a generalization of the operations of concatenation and quotient [2], [3], [25] and later was studied in the framework of DNA computing [26]–[28] and strings over an infinite alphabet [29].

2.3 Related Models

Contextual grammars. Contextual grammars [1] use the operation of context adjoining: $x \Longrightarrow y$, if $y = uxv$ and $(u, v) \in C(x)$, where $C(x)$ is a list of possible contexts to adjoin for x. Such grammars were introduced with a linguistic motivation and have been used to model various natural language phenomena. While the operation of context adjoining is different from the insertion, this model served as an inspiration for the introduction of insertion grammars (under the name of semi-contextual grammars) [4]. Insertion grammars are pure grammars and correspond directly to insertion systems (where no deletion rules are considered). According to [7], they directly inspired the model of insertion-deletion systems.

Generalization of the operations of concatenation and quotient. In [2], [3], the operation of concatenation was generalized to the operation of (context-free) insertion, by allowing the concatenation to happen anywhere in the string (and not only at its end). Different language-theoretical properties of this operation were established. The PhD thesis [2] also introduced several variants of the insertion

operation like parallel or scattered insertion. In the same thesis, the operation of quotient was also generalized to the operation of deletion, where a substring can be removed from any place in the string, see also [25].

Leftist grammars. Leftist grammars are formal grammars having rules $a \to ba$ and $cd \to d$. Such grammars were used to model accessibility problems in the field of computer security [5]. It is not difficult to observe that these grammars directly correspond to insertion-deletion systems of size $(1, 1, 0; 1, 1, 0)$. It was shown that the reachability problem for such grammars is decidable [5]. At the same time, it was shown that complex functions can be computed using this model, like the Ackermann function [30]. The PhD thesis [31] examines in details this variant of insertion-deletion systems and provides some interesting insight on its functioning.

Restarting automata. Restarting automata are a machine model that is motivated by the linguistic technique of analysis by reduction. The basic variant introduced in [6] works like a finite automaton with a $k-$symbols lookahead. It has an additional operation allowing to shorten the string under the lookahead window and immediately moving the head to the beginning of the string (thus restarting the computation). This model directly corresponds to the iterated contextual deletion operation, where the right context is always empty. There are numerous variants of restarting automata, some of them corresponding to grammars using different types of insertion-deletion rules.

DNA computing. DNA computing is an interdisciplinary field that leverages the principles of molecular biology, particularly the properties of DNA, to perform computational tasks. By manipulating DNA strands through processes like hybridization, ligation, and polymerase chain reactions, researchers can encode and solve complex problems, including mathematical and combinatorial challenges. As pointed out in [12], the process of mismatched annealing of DNA strands can be seen as an insertion or a deletion of a string in a specified context. This observation led to the intense study of *insertion-deletion* systems

in the framework of DNA computing, and most of the existing results directly or indirectly relate to this motivation.

RNA editing. A similar process happens in the case of RNA editing [13], where the uracil base U is inserted or deleted in some left context. This was a natural motivation for the study of one-sided and leftist insertion-deletion systems [21], [32]. A related model, guided-insertion systems [33] used to model RNA editing, is using a similar principle of insertion or deletion of a string in a specified left context.

CRISPR-Cas9. Recently, another biological process has been discovered that uses insertion and deletion to edit the genome forming, being the base for the CRISPR-Cas9 technology [14], [15]. This technology uses a protein called Cas9 that can be programmed to target specific DNA sequences and cut them. The cell then repairs the cut using the insertion and deletion operations. This process is used to edit the genome of living organisms, including humans, and has a wide range of applications in medicine, agriculture, and biotechnology. No formal model has been proposed to capture the action of CRISPR-Cas9 yet, but it is clear that it can be modeled using insertion-deletion systems.

3 New Proof Methods

This section introduces two significant proof techniques for computational completeness in the area of insertion-deletion systems. They are not described in the previous overview [17] because they originate from 2011 and 2020. The first key idea is the use of the special Geffert normal form as a simulation target. By imposing restrictions on how the terminal string is generated, this approach reduces proof complexity and enhances control over the derivation process. This method has turned out to be so effective that it is now employed in nearly all proofs developed over the past 15 years.

The second idea involves the concept of rule independence. When this condition is met, rules can be applied in any sequence without affecting the outcome. By strategically selecting specific rule orders, it

becomes possible to significantly streamline the proof process. This is a relatively new approach, and we expect it to be further developed in the future, with an increasing number of proofs likely to adopt it.

3.1 Special Geffert Normal Form Target

The first results on the computational completeness of insertion-deletion systems were established by simulating Turing machines. Very quickly, formal grammars were used as a simulation target. In order to decrease the descriptional complexity parameters, the grammars were supposed to be in some normal form, usually the Kuroda normal form. Starting from 2007, the method of *direct simulation* was developed by Yurii Rogozhin and the third author of this paper. This method uses a different insertion-deletion system as a target for the simulation [10]. Hence, instead of simulating a Turing machine or a grammar in Kuroda normal form, one simulates a particular type of insertion or deletion rules. For example, to prove the computational completeness of systems of size $(2,0,0;1,1,1)$ it is sufficient to show how an insertion rule $(a,b,c)_{ins}$ can be simulated using insertion-deletion rules of size $(2,0,0;1,1,1)$. Since the systems of size $(1,1,1;1,1,1)$ are computationally complete, this allows to prove the computational completeness of systems of size $(2,0,0;1,1,1)$. Most of the results from 2007 to 2011 were obtained using this method. We refer to [17], [34] for a detailed description of the method and for an overview of corresponding results.

However, the method of direct simulation has some limitations. When considering systems of small size, e.g., $(1,1,0;1,1,0)$, in the regulated insertion-deletion framework like matrix or graph-controlled systems, the method of direct simulation had difficulties to take into account the control mechanism. In [22], a different approach was proposed by using as a simulation target a grammar in the special Geffert normal form (SGNF). The grammar in SGNF has several important features useful for simulations. First, rules are very simple, corresponding to at most 3 single-symbol insertion-deletion operations: $X \to bY$, $X \to Yb$, and $AB \to \lambda$. In many cases, two of these rule types can be easily simulated by corresponding (regulated) insertion-deletion system. Next, during the first two stages of the derivation, there is only

one non-terminal symbol present in the string, which is very useful for the simulation. Finally, during the third stage, only the cooperative erasing rules can be applied, which is also very useful for the simulation, as one does not have to take into account the interference with the two other rule types. Different proofs make additional restrictions on SGNF rules for technical purposes, e.g., ensuring that X is different from Y, or that there are no sequences of the same symbol (like AAA) generated by the grammar — such variant is called space-separated SGNF (ssSGNF) [24]. The proofs using SGNF allow for addressing a part of the proof difficulty by carefully imposing additional restrictions on SGNF rules that allow to forbid some proof cases. In turn, this allows for getting computational completeness results with smaller descriptional complexity parameters. The versatility of the SGNF target is so significant that it is employed in nearly all existing proofs from the last 15 years.

3.2 Independent Rules

The core concept of the method of independent rules is to identify the conditions under which the sequence of rule applications does not influence the derivation's outcome. Once these conditions are established, any given construction can be evaluated against them. If the conditions are met, the rules can be applied in an order that simplifies the proof. If the conditions are not met, the construction can be adjusted to ensure they are satisfied, thereby facilitating a more straightforward proof process by grouping and applying immediately rules that perform some specific part of the computation.

To simplify the presentation of the method, we consider only insertion rules (hence we omit the subscript ins).

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 r) may happen inside w.

Definition 3. An insertion rule $r_1 = (u_1, x_1, v_1)$ matches an insertion rule $r_2 = (u_2, x_2, u_2)$ if $u_1v_1 = u_2v_2$.

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 specific sense, matching rules compete for being applied on a string that they match.

Definition 4. 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 in 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 $\mathrm{Suff}(u)v \cap \mathrm{Pref}(w) \neq \emptyset$.

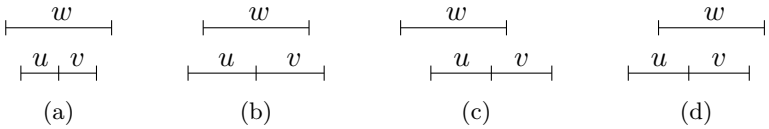

Figure 1: Four possible cases for string overlap: (a) uv is a substring of w, (b) w is a substring of uv, (c) $\mathrm{Suff}(w) \cap u\,\mathrm{Pref}(v) \neq \emptyset$, (d) $\mathrm{Suff}(u)v \cap \mathrm{Pref}(w) \neq \emptyset$.

We now introduce a crucial concept: the notion of an *independent* rule set. The key idea here is to define the conditions under which rules

can be applied in any sequence, without affecting the overall outcome. This independence allows us to select specific orders of rule application, which can significantly simplify the proof process.

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.

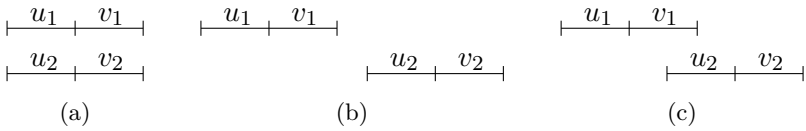

Figure 2: Three possible rule relations for an independent rule set (we recall that these conditions are symmetric with respect to rules): (a) rules perfectly match, (b) rules do not overlap and $\mathrm{Suff}(u_1v_1) \cap \mathrm{Pref}(u_2v_2) = \emptyset$, (c) rules do not overlap and $\mathrm{Pref}(u_2)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

As shown in [23], the independence property allows rules to be applied in any order.

Theorem 1. *Let $G = (V, A, R)$ be an insertion system 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.*

Considering the last theorem, it is worth noting that one can always choose to apply the rule at the leftmost (or rightmost) position in the string, which corresponds to the leftmost (or rightmost) derivation.

A similar concept applies to the independence of deletion rules, where a rule should not overlap the whole context uxv of other rules, as well as to sets containing both insertion and deletion rules. Although these definitions are slightly more complex, the conclusion of Theorem 1 remains valid. Even if the construction only partially satisfies the independence property — for instance, applying only to insertion rules — then all insertion rules can be applied in any order, while deletion rules must be applied in a specific sequence. Therefore, the independence property is a powerful tool in the study of insertion-deletion systems, enabling significant simplification of proof arguments.

3.3 Lazy/Eager Deletion

Another interesting idea was explored in [16]. It is custom in regulated rewriting area to consider particular allowed sequences of rules. Matrix and time-varying insertion-deletion systems are typical examples of such an approach. The cited article introduces the concept of *lazy* and *eager* deletion. In the case of lazy deletion, any derivation of the system is performed as follows: first, insertion rules are applied and, after that, deletion rules are applied. Hence, any rule sequence for a successful derivation is in I^*D^*. The lazy deletion can be interpreted as an iterated insertion followed by an iterated deletion. In the case of eager deletion, a deletion rule is applied immediately after an insertion rule, so any rule sequence for a successful derivation belongs to $(ID)^*$. The paper [16] gives some examples of families of insertion-deletion systems where it is possible to reorder any derivation to make it lazy or eager.

The lazy deletion can be very useful for proofs, as it decouples the derivation into two parts that can be examined independently. The eager deletion somehow models the string rewriting process (which corresponds to one insertion and one deletion), thus helping to simulate grammar-like targets. In a more general way, the eager deletion is a particular variant of the notion of M-related rules [26], which are a

group of related rules that are applied one after another in order to simulate some particular string rewriting. E.g., by using rule $(\lambda, vR, \lambda)_{ins}$ followed by $(\lambda, Ru, \lambda)_{del}$, the rewriting rule $u \to v$ can be simulated. The proofs from the cited paper basically show that any derivation can be reorganized as a sequence of applications of M-related rules. The simplest variant of M-related rules is the eager deletion. In more complex cases, particular sequences of insertions and deletions are needed. All known proofs on insertion-deletion systems use the concept of M-related rules (sometimes not explicitly naming it), so we think that it could be useful to present the proofs from this perspective — for some string transformation, there is a sequence of insertion-deletion rules that should be applied in some order, and the proofs show that any derivation can be reorganized to follow this order.

4 Insertion-deletion Systems

Insertion-deletion systems become computationally complete with rules of rather small size. We below recall the best known parameters for computationally complete insertion-deletion systems. In the overview [17], there are more technical details concerning the proof methods.

4.1 Recall of Results

Table 1 recalls the results on the computational power of symmetrical insertion-deletion systems.

4.2 One-sided Insertion-deletion Systems

One-sided insertion-deletion systems are a particular case of insertion-deletion, where one of the contexts in rules is always empty.

Table 2 recalls the results on the computational power of one-sided insertion-deletion systems.

One-sided systems with small parameters are known to not achieve computational completeness. In Table 3, the list of non-complete insertion-deletion systems is depicted. Systems with these (or smaller) parameters are commonly considered in a regulated framework.

Table 1: Results on symmetrical insertion-deletion systems

Size	Family	Reference	Size	Family	Reference
$(1,2,2;1,1,1)$	RE	[9], [12]	$(3,0,0;2,0,0)$	RE	[26]
$(1,2,2;2,0,0)$	RE	[9], [12]	$(1,1,1;2,0,0)$	RE	[12]
$(2,1,1;2,0,0)$	RE	[9], [12]	$(2,0,0;1,1,1)$	RE	[11]
$(1,1,1;1,2,2)$	RE	[35]	$(1,1,1;1,1,1)$	RE	[35]
$(2,1,1;1,1,1)$	RE	[35]	$(2,0,0;2,0,0)$	$\subsetneq CF$	[27]
$(3,0,0;3,0,0)$	RE	[26]	$(m,0,0;1,0,0)$	$\subsetneq CF$	[27]
$(2,0,0;3,0,0)$	RE	[26]	$(1,0,0;p,0,0)$	$\subsetneq REG$	[27]

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

Size	Reference	Size	Reference
(1,1,2;1,1,0)	[11]	(1,1,0;1,1,2)	[10]
(2,0,2;1,1,0)	[11]	(1,1,0;2,0,2)	[10]
(2,0,1;2,0,0)	[11]	(2,0,0;2,0,1)	[10]
(1,2,0;1,0,2)	[36]		

4.3 Regular Contexts

An interesting variant of insertion-deletion systems was considered in [31], [32], [38], where the contexts of the rules are regular expressions.

Given an alphabet V, an *extended insertion rule* r is the tuple $(E_l, x, E_r)_{ins}$, where $x \in V^*$ and E_l and E_r are regular expressions over V. The rule r can be applied to the string uv to yield uxv, if $u = u_1u_2$ such that $u_2 \in L(E_l)$ and $v = v_1v_2$ such that $v_1 \in L(E_r)$. An extended deletion rule is defined in a similar way. The corresponding families of languages are denoted using the REG string instead of a number. In [39], the following results are shown:

Table 3: Insertion-deletion systems known to be non-complete

Size	Reference
$(1,1,1;1,1,0)$	[10]
$(1,1,0;1,1,1)$	[11]
$(2,0,0;1,1,0)$	[37]
$(1,1,0;2,0,0)$	[37]
$(2,0,0;2,0,0)$	[27]
$(n,0,0;1,0,0)$	[27]
$(1,0,0;p,0,0)$	[27]

Theorem 2. *[39] The following equalities hold:*

$$INS_1^{REG,0}DEL_1^{REG,0} = INS_1^{2,0}DEL_1^{1,0} = INS_1^{1,0}DEL_1^{2,0}.$$

These results open an interesting perspective. Any insertion-deletion system having rules with the size greater than shown above can be considered as having rules with regular contexts. This allows for simplifying the constructions and for having a greater control over the derivation process. However, the above equivalence does not always hold for the regular rewriting variants, like graph or matrix control as shown in [39]. However, in some cases, it is possible to achieve such an equivalence. In the cited paper, the proof of the universality of graph-controlled insertion-deletion systems with two states is done by first showing the universality using extended rules and then showing that the proof argument holds for the regular rules. Also, the same paper contains a series of results on the computational completeness of insertion-deletion systems with one-sided regular contexts and different control mechanisms.

4.4 Pure Insertion Systems

When considering only the (contextual) insertion operation, a special variant of context-sensitive grammars is obtained, called insertion grammars. As shown in [23], such grammars are universal (in Turing sense) with insertion rules of size $(2,2,2)$. Since these are pure

grammars, additional mechanisms are needed to extract the needed result (the intersection with a terminal alphabet has no sense anymore). When an appropriate "squeezing" mechanism is used — such as inverse morphism plus projection, left/right quotient, intersection with $LOC(2)$ and projection — they generate any recursively enumerable language. Matrix control allows to decrease the size of insertion rules to $(2, 1, 1)$ for similar results. We refer to [23] for the technical details, as well as for the historical overview of the results in this area.

The study of pure insertion systems is interesting from the point of view of formal grammars, as this is a particular case of growing context-sensitive languages [40]. Moreover, similar models are very common in the linguistics area. Finally, these results allow to immediately claim the computational completeness of systems of size $(2, 2, 2; 2, 0, 0)$ using the lazy deletion derivation strategy.

The complexity of the constructions for pure insertion systems motivated the introduction of the notion of independent rules that allowed to simplify the main proof. We believe that the study of these systems can be useful for the discovery of new proof ideas for the insertion-deletion area.

4.5 Derivation Graphs

Derivation graphs are an alternative representation of a derivation of an insertion-deletion system with rules of size $(1, 1, 0; 1, 1, 0)$, introduced in [31]. They can be seen as a generalization of derivation trees [37], and they are a useful tool to understand complex derivations like in [30].

Consider the insertion-deletion system $\Gamma = (V, T, A, R)$ with rules of size $(1, 1, 0; 1, 1, 0)$. Since a rule in R has the form $r : (x, y, \lambda)_t$, $t \in \{ins, del\}$, when r is applied, we may say that "x inserted y" if $t = ins$ and that "x deleted y" if $t = del$. To give these statements a formal meaning, it is necessary to distinguish between the symbols of the alphabet V and their occurrences in a string $w \in V^*$. In [31], the term "letter" is used to designate an occurrence of a symbol in a string. A derivation graph is a formalization of the relations "x inserted y" and "x deleted y" between the letters of a derivation.

More formally, consider a derivation $C : w \Rightarrow_\Gamma^* v$ of Γ and let

$\bar{C}$ be the set of letters (different occurrences of symbols) of C. The *derivation graph* of C is the labeled graph $G = (\bar{C}, E, h)$, with the following properties:

- if a letter x inserts the letter y in C, then $(x, y) \in E$ and $h((x, y)) = ins$,
- if a letter x deletes the letter y in C, then $(x, y) \in E$ and $h((x, y)) = del$.

We refer to [31] for a more formal treatment of derivation graphs.

For example, consider the system $\Gamma_{abc} = (V, V, A, R)$ with $V = \{a, b, c\}$, $A = \{a\}$, and the following set of four insertion rules and one deletion rule of size $(1, 1, 0; 1, 1, 0)$:

$$R = \{(a, b, \lambda)_{ins}, (a, a, \lambda)_{ins}, (b, c, \lambda)_{ins}, (a, c, \lambda)_{ins}, (c, b, \lambda)_{del}\}.$$

Take the following derivation of Γ:

$$C_1 : \underline{a} \Rightarrow \underline{a}a \Rightarrow \underline{a}ba \Rightarrow a\underline{a}ba \Rightarrow aac\underline{b}a \Rightarrow aac\underline{b}ca \Rightarrow aa\underline{c}bcca \Rightarrow aaccca.$$

Underlining shows the context of the rule applied in the next derivation step, e.g., in $a\underline{a}ba \Rightarrow aacba$, the underlined a inserted the first occurrence of c, and in $aa\underline{c}bcca \Rightarrow aaccca$ the underlined c deleted the occurrence of b. Figure 3 gives a graphical presentation of the derivation graph corresponding to C_1.

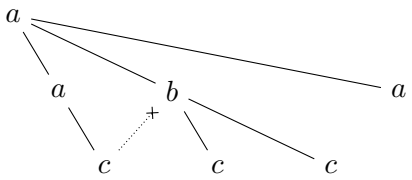

Figure 3: The derivation graph of derivation C_1 of Γ_{abc}. Full lines — show insertions and dotted lines with a cross ⋯× show deletions. As a visual aid, all edges go down and right, because the insertions and deletions occur to the right of respective contexts

Since multiple derivations may result in the same word, multiple derivation graphs can correspond to the same word. Furthermore,

derivation graphs discard some information about the order in which the operations occur: e.g., a derivation C_1' in which b inserts a c *before* the second a inserts a c would correspond to the same derivation graph shown in Figure 3. In [31], further detailed examples are given of using derivation graphs to deal with classes of derivations sharing a certain property. Finally, derivation graphs intuitively highlight the connection between insertion-deletion systems of size $(1, 1, 0; 1, 1, 0)$, leftist grammars, and access control [41], [42].

5 Regulated Insertion-deletion Systems

5.1 Matrix insertion-deletion systems

A *matrix insertion-deletion system* is a construct

$$\Gamma = (V, T, A, M), \text{ where}$$

- V is a finite alphabet,
- $T \subseteq V$ is the terminal alphabet,
- $A \subseteq V^*$ is a finite set of axioms,
- $M = \{r_1, \ldots, r_t\}$, $t \geq 1$, is a finite set of sequences of rules called *matrices* of the form $r_i = [r_{i1}, \ldots, r_{ik_i}]$, with $k_i \geq 1$, $1 \leq i \leq t$, $1 \leq j \leq k_i$, where r_{ij} is an insertion or deletion rule over V.

For two strings $w, z \in V^*$, we write $w \Longrightarrow_\Gamma z$ if there exist a matrix $r_i = [r_{i1}, \ldots, r_{ik_i}] \in M$ and the sequence of strings $(w_i)_{1 \leq i \leq n+1}$, $w_i \in V^*$, such that $w = w_1$, $z = w_{n+1}$, and $w_j \Rightarrow_{r_{ij}} w_{j+1}$. The language generated by Γ is the language of terminal strings that can be obtained from the axioms via the reflexive and transitive closure of this derivation relation:

$$L(\Gamma) = \{v \in T^* \mid \exists u \in A : u \Longrightarrow_\Gamma^* v\}.$$

Note that the semantics of matrix insertion-deletion systems does not feature appearance checking, as is traditional in regulated rewriting. This is because appearance checking can often be implemented

via additional rules in matrices and is not necessary for computational complexity in most of the cases. We also observe that the matrix control can be viewed as a particular case of the graph control, we refer to [16] for more details on this topic.

We summarize in Table 4 the best known computational completeness results on matrix insertion-deletion systems. At the same time, we remark that systems having rules of size $(2, 0, 0; 2, 0, 0)$ cannot be computationally complete with matrices of any size, as shown in [37].

Table 4: Computationally complete matrix insertion-deletion systems

Size	Matrix size	References
$(1, 1, 0; 1, 1, 0)$	3	[22]
$(1, 0, 1; 1, 0, 1)$	3	[22], [43], [44]
$(1, 0, 1; 1, 1, 0)$	3	[43], [44]
$(1, 1, 0; 1, 1, 1)$	2	[44]
$(1, 1, 0; 2, 0, 0)$	2	[22]
$(2, 0, 0; 1, 1, 0)$	2	[22]
$(2, 0, 0; 1, 0, 1)$	2	[43], [44]
$(1, 0, 1; 2, 0, 0)$	2	[43], [44]
$(1, 1, 1; 1, 0, 0)$	2	[45]
$(1, 0, 0; 1, 1, 1)$	3	[44]
$(1, 0, 0; 1, 2, 0)$	3	[45]
$(1, 2, 0; 1, 0, 0)$	2	[45]
$(1, 0, 0; 1, 0, 2)$	3	[45]
$(1, 0, 2; 1, 0, 0)$	2	[45]

5.2 Graph-controlled Insertion-deletion Systems

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

$$\Gamma = (V, T, A, H, i_0, I_f, R), \text{ where}$$

- V is a finite alphabet,
- $T \subseteq V$ is the terminal alphabet,
- $A \subseteq V^*$ is the finite set of axioms,

- H is the set of *states* of Γ,
- $i_0 \in H$ is the *initial state*,
- $I_f \subseteq H$ is the set of *final states*,
- R is a finite set of rules of the form (l, r, E), where r is an insertion or deletion rule over V, $l \in H$ is the source state, and $E \subseteq H$ is a set of target states.

The relation $\{(i, j) \mid (i, r, E) \in R, j \in E\}$ defines the communication graph of the system. Some works assume without losing generality that the correspondence between the source state i and the rule r is a bijective mapping. Other works slightly modify the definition by assigning the rules to the edges between the states rather than the states themselves, which yields an equivalent model.

A configuration of Γ is a pair (i, w), where $i \in H$ is the current state and w is the current string. For two strings $w, z \in V^*$, we write $(w, i) \Longrightarrow_\Gamma (z, j)$ if there exists a rule $(i, r, E) \in R$ such that $w \Longrightarrow_r z$ and $j \in E$. The language generated by Γ is the language of terminal strings that can be obtained from the axioms by the reflexive and transitive closure of this derivation relation:

$$L(\Gamma) = \{v \in T^* \mid \exists u \in A, \exists i_f \in I_f : (u, i_0) \Longrightarrow_\Gamma^* (v, i_f)\}.$$

Some of the best results on computational completeness for such systems are summarized in Table 5. On the other hand, as shown in [37], graph-controlled insertion-deletion systems of size $(2, 0, 0; 2, 0, 0)$ cannot be computationally complete.

By using appearance checking [48], more powerful systems can be constructed. From [49], it can be immediately deduced that Parikh sets of graph-controlled insertion-deletion systems with appearance checking and rules of size $(1, 0, 0; 1, 0, 0)$ are exactly $PsRE$, the Parikh sets of all recursively enumerable languages. A similar result from [50] shows that by using the priority of deletion over insertion (which can also be seen as a special type of appearance check), $PsRE$ can be generated with rules of size $(1, 0, 0; 1, 0, 0)$. By adding more context, RE can be generated as well. In Table 6, we collect the results on the computational completeness of graph-controlled insertion-deletion systems with

Table 5: Computational completeness for graph-controlled insertion-deletion systems

Size	Graph size	Reference
$(2,0,0;1,1,0)$	3	[46]
$(2,0,0;1,0,1)$	3	[46]
$(1,1,0;2,0,0)$	3	[46]
$(1,0,1;2,0,0)$	3	[46]
$(2,0,0;1,1,1)$	2	[43]
$(1,1,0;1,2,0)$	2	[21]
$(1,2,0;1,1,0)$	2	[21]
$(1,1,1;1,0,0)$	5	[47]
$(1,0,0;1,1,1)$	5	[47]
$(1,i,i';1,j,j')$, $i+i'=1, j+j'=1$	3	[47]

deletion priority. We remark that the size of the graph depends on the simulated register machine.

Another possibility is to consider insertions or deletions on the left/right side of the string. In this case, context-free insertion-deletion of one symbol leads to computational completeness in the context of the graph control with the size of the graph equal to 8 [51].

Table 6: Results for graph-controlled insertion-deletion systems with the priority of deletion over insertion

Size	Graph size	Family	Reference
$(1,0,0;1,0,0)$	*	$PsRE$	[50]
$(1,1,0;1,0,0)$	*	RE	[50]
$(1,0,0;1,1,0)$	*	RE	[50]

Several recent works have focused on graph-controlled insertion-deletion systems in which the graph has a particular shape. Table 7 lists the results from [52] for graphs having a star topology, and Table 8 describes the results from [53] on graphs having a path topology.

Another particular case of graph-controlled insertion-deletion systems are time-varying insertion-deletion systems [16]. For such systems, the rule set available cyclically changes in each step. In the

Table 7: Computational completeness for graph-controlled insertion-deletion systems with a *star-shaped* control graph

Size	Graph size	Reference
$(1,1,0;2,0,0)$	6	[52]
$(1,0,1;2,0,0)$	6	[52]
$(2,1,1;1,0,0)$	4	[52]

Table 8: Computational completeness for graph-controlled insertion-deletion systems with a *path-shaped* control graph, where $i', i'', j', j'' \in \{0,1\}$

Size	Constraints	Graph size	Reference
$(1,i',i'';1,j',j'')$	$i'+i''=1,\ j'+j''=2$	3	[53]
$(2,i',i'';1,j',j'')$	$i'+i''=1,\ j'+j''=1$	3	[53]
$(2,i',i'';1,j',j'')$	$i'+i''=0,\ j'+j''=1$	4	[53]
$(1,i',i'';1,j',j'')$	$i'+i''=1,\ j'+j''=1$	4	[53]

cited paper, computational completeness is shown for rules of size $(1,1,0;1,1,1)$ and period 3, as well as for rules of size $(1,0,1;1,0,1)$ and period 5.

Moreover, paper [39] shows the computational completeness of graph-controlled insertion-deletion systems with extended rules having the size $(1, REG, 0; 1, REG, 0)$ and two states.

5.3 Context-dependent Insertion-deletion Systems

Context conditions can be attached to an insertion or a deletion rule by indicating a set of permitting and forbidding strings, i.e., rules are of the form (q, P, F), where q is an insertion or a deletion rule, and P, F are finite subsets of V^*. The application of a rule $r : (q, P, F)$ to a string w is defined as follows: $w \Longrightarrow_r w'$ if and only if $w \Longrightarrow_q w'$ and for all $x \in P$, x is a substring of w, and for all $y \in F$, y is not a substring of w.

Definition 6. A *semi-conditional insertion-deletion system* (SID for short) is a tuple $G_{SID} = (V, T, A, R)$, where V is an alphabet, $T \subseteq$

N is the terminal alphabet, $A \subseteq V^*$ is a finite set of initial strings (axioms), and $R \subseteq (V^*)^3 \times \{ins, del\} \times 2^{V^*} \times 2^{V^*}$ is a finite set of semi-conditional insertion-deletion rules. The derivation relation obtained by the definition of a derivation using a rule $r : (q, P, F)$ in R is denoted by $\Longrightarrow_{G_{SID}}$, its reflexive and transitive closure – by $\Longrightarrow^*_{G_{SID}}$.

The language generated by an SID $G_{SID} = (N, T, A, R)$ is defined by

$$L(G_{SID}) = \{w \in T^* \mid \exists u \in A : u \Longrightarrow^*_{G_{SID}} w\}.$$

Table 9: Results for semi-conditional and random-context insertion-deletion systems

Size	Degree	Power	Reference
$(1, 0, 0; 1, 0, 0)$	(2,2)	$= RE$	[39]
$(2, 0, 0; 1, 1, 0)$	(1,1)	$= RE$	[39]
$(1, 1, 0; 1, 1, 0)$	(3,1)	$= RE$	[54]
$(1, 0, 1; 1, 1, 0)$	(3,1)	$= RE$	[54]
$1 + i, i', i''; 1 + j, j', j'')$ $i + i' + i'' = 1 = j + j' + j''$	(2,1)	$= RE$	[55]
$(1 + i, i', i''; 1 + j, j', j'')$ $i + i' + i'' = 1 = j + j' + j''$	(0,2)	$= RE$	[24]
$(1, 0, 0; 1, 0, 0)$	(1,2)	$= RE$	[56]
$(1, 0, 0; 0, 0, 0)$	(2,2)	$\subseteq MON$	[39]
$(1, 1, 0; 1, p, 0)$	(1,1)	$\neq RE$	[39]
$(1, 0, 0; 1, 0, 0)$	$(n$,1)	$\neq RE$	[56]
$(1, 0, 0; 1, 0, 0)$	(0,2)	$\supseteq REG$	[56]

We define the *degree* of a semi-conditional insertion-deletion system $G_{SID} = (N, T, A, R)$ as the pair (a, b), where

$$a = \max\{|w| \mid w \in P, ((u, x, v)_z, P, Q) \in R, z \in \{ins, del\}\},$$
$$b = \max\{|w| \mid w \in Q, ((u, x, v)_z, P, Q) \in R, z \in \{ins, del\}\}.$$

We denote the family of languages generated by semi-conditional insertion-deletion systems of size $(n, m, m'; p, q, q')$ and degree (a, b) by $SC_{a,b}INS_n^{m,m'}DEL_p^{q,q'}$. Moreover, semi-conditional insertion-deletion systems of degree $(1, 1)$ are called *random context* insertion-deletion systems. We will refer to the families of languages produced by such systems by the notation $RCINS_n^{m,m'}DEL_p^{q,q'}$.

The results on semi-conditional and random-context insertion-deletion systems are summarized in Table 9. What is interesting in this table is that, while generally most of the computational completeness results in the area of insertion-deletion systems hold if the sizes of insertion and deletion are exchanged, this is not the case for semi-conditional and random-context insertion-deletion systems, where the size of the insertion or the length of the forbidding context seem to be more important than the converse counterparts.

We would like to point out the result from [56] about the computational completeness of semi-conditional insertion-deletion systems of size $(1, 0, 0; 1, 0, 0)$ and degree $(1, 2)$. First of all, this result uses only context-free insertions and deletions of single symbols. For most variants of insertion-deletion systems, such parameters do not allow for achieving computational completeness. Moreover, in the construction, all rules are of degree $(0, 2)$ except a single rule of degree $(1, 2)$. This suggests that more optimal results could be obtained, e.g., by using rules of degrees $(1, 0)$ and $(0, 2)$ only.

5.4 Other Models

In this section, we mention some models of regulated insertion-deletion systems, as introduced in [16]. The first model, Cooperating Distributed Insertion-Deletion Systems (CD-IDS), is the adaptation of the idea of CD grammar systems [57] to the use of the insertion-deletion operations instead of rewriting. We recall that the CD model features a set of components containing insertion-deletion rules and that are applied in turns according to different derivation strategies. The above paper introduces the model and shows some relations between derivation modes.

Another model introduced in the above paper is an insertion-deletion system with activation of rules. The main idea of the concept of activation of rules is to activate rules for succeeding steps of a derivation with the application of a specific rule in a derivation step. This concept of activation of rules allows for a dynamic evolution of the rule sets available at a specific time during a derivation, depending on the history of the derivation steps carried out so far, i.e., depending on the rules having been applied in the derivation steps carried out previously. The cited paper shows tight relations between this model and time-varying insertion-deletion systems.

6 Insertion-deletion with Substitutions

An interesting extension of insertion-deletion systems has been studied in [58]–[60]. It enriches the standard model with the operation of substitution of a single symbol by a single symbol, possibly with context. We remark that the context-free substitution was also considered together with context-free insertion-deletion operations restricted to the left/right side of the string [51].

For an alphabet V, a substitution rule (of symbol a by symbol b with left context u and right context v) is written as $(u, a \to b, v)$ with $a, b \in V$ and $u, v \in V^*$. It precisely corresponds to the rewriting rule $uav \to ubv$. Let S be the set of rewriting rules. Then, $r = \max\{|u| : (u, a \to b, v) \in S\}$ and $r' = \max\{|v| : (u, a \to b, v) \in S\}$. For insertion-deletion systems of size $(n, m, m'; p, q, q')$, enriching them with substitution rules with left context of size at most r and right context of size at most r' gives us insertion-deletion-substitution systems of size $(n, m, m'; p, q, q'; r, r')$, and the corresponding family of languages is denoted by $INS_n^{m,m'} DEL_p^{q,q'} SUB^{r,r'}$.

Remark 1. Since exactly one symbol is replaced by exactly one symbol by any substitution rule, there is no subscript below SUB in the notation. Note, however, that, unlike $INS_0^{0,0}$ or $DEL_0^{0,0}$ representing absence of insertion or deletion operations, $SUB^{0,0}$ represents a set of substitution operations without context, i.e., rules $(\lambda, a \to b, \lambda)$, precisely corresponding to renaming rules $a \to b$ with $a, b \in V$.

In [60], one goes even further, considering matrices of insertion-deletion-substitution rules; the definitions are similar to those in Subsection 5.1. The size of such systems with matrices of size at most k is denoted by $(k; n, m, m'; p, q, q'; r, r')$, and the corresponding families of languages are denoted by $MAT_k INS_n^{m,m'} DEL_p^{q,q'} SUB^{r,r'}$. Superscript ac is added to MAT if appearance checking is allowed. An unbounded parameter is denoted by $*$.

Table 10 recalls the computationally complete insertion-deletion-substitution systems.

Table 10: Computationally complete insertion-deletion-substitution systems

Size	Reference
$(1, 1, 0; 1, 1, 0; 0, 1)$	[58]
$(1, 1, 1; 1, 1, 0; 1, 0)$	[58]
$(1, 1, 0; 2, 0, 0; 0, 1)$	[58]
$(2, 0, 0; 2, 0, 0; 1, 0)$	[59]
$(1, 0, 0; 1, 0, 0; 1, 1)$	[59]
$(1, 0, 1; 1, 0, 0; 1, 0)$	[60]

In Table 11, the list of non-complete insertion-deletion-substitution systems is given.

Table 11: Insertion-deletion-substitution systems known to be non-complete

Size	Reference
$(n, m, m'; 0, 0, 0; r, r')$	[58]
$(1, 1, 0; 1, 1, 1; 0, 0)$	[58]

In Table 12, we summarize the best known computational completeness results on matrix insertion-deletion-substitution systems.

Table 13 lists non-completeness results on matrix insertion-deletion-substitution systems.

In [58], one conjectures non-completeness of insertion-deletion-substitution systems of sizes (1,1,0;1,1,0;1,0) and (1,1,0;2,0,0;1,0) and

Table 12: Computationally complete matrix insertion-deletion-substitution systems

Size	Matrix size	Reference
$(1,0,0;1,1,1;0,0)$	2	[60]
$(1,1,1;1,0,0;0,0)$	2	[60]
$(1,0,0;1,0,0;1,0)$	2	[60]
$(1,0,0;2,0,0;0,0)$	*,ac	[60]
$(1,0,0;1,1,0;0,0)$	*,ac	[60]
$(1,1,0;1,1,0;0,0)$	2	[60]
$(1,1,0;1,0,1;0,0)$	2	[60]
$(2,0,0;1,1,0;0,0)$	2	[60]
$(1,1,0;2,0,0;0,0)$	2	[60]

Table 13: Matrix insertion-deletion-substitution systems known to be non-complete

Size	Matrix size	Reference
$(1,1,0;1,0,0;0,0)$	2	[60]
$(1,0,0;1,1,0;0,0)$	*	[60]
$(2,0,0;2,0,0;0,0)$	*	[60]
$(1,0,0;0,0,0;1,0)$	2	[60]

declares the problem for size (1,1,0;1,0,1;1,0) open. In [59], one conjectures non-completeness for systems of size (1,0,0;1,0,0;1,0).

Networks of Evolutionary Processors

Networks of evolutionary processors are a distributed computational model with biological inspiration [61]. The model consists of a set of processors arranged in a graph that can communicate with each other. The processors can perform the operations of context-free insertion, deletion, or substitution of one symbol. Hence, they are closely related to insertion-deletion systems with substitutions. In terms above, they have rules of size $(1,0,0;1,0,0;0,0)$. The computation is a sequence of compute and communication steps. During the compute step, all possible rules are applied to the set of strings from the processor, yielding

a new set of strings. During the communication step, the processors can exchange strings as follows. Each processor has an attached regular input and output filters. All strings that pass the output filter of the processor are removed from it. Next, such strings (that pass the output filter) are added to the connected processors if they pass their input filter. Several variants of the model have been studied, including those with specialized processors (performing only one type of operation) and having special subregular filters (e.g., LOC(2), Star, finite, etc.). We refer to [62] for a recent review of this topic. We would only like to mention that Yurii Rogozhin constructed a small universal network of evolutionary processors having only 4 rules [63].

7 Simulator for Insertion-deletion Systems

Early proofs of computational completeness for graph-controlled and matrix insertion-deletion systems [20],[22] encountered significant challenges due to the numerous possible rule applications in each derivation step. The complexity was much greater than for ordinary insertion-deletion systems, primarily because the shorter context length provided less control over the derivation process. This complexity made it increasingly difficult and error-prone to manually trace the derivation tree. In fact, our peer review activity of insertion-deletion systems-related papers from the last decade revealed that every first version contained at least one error in their computational completeness proofs, typically due to overseeing some possible rule applications. Recognizing the need for better tools, we developed a simulator for graph-controlled semi-conditional grammars, called `gcrsim` and written in the Perl language. Initially distributed privately from 2010, `gcrsim` is now freely available on GitHub [64]. Since 2011, we have systematically used this tool to aid in the design of insertion-deletion systems in all our papers, as well as during paper peer reviews.

The syntax of the input file for `gcrsim` is simple and intuitive. It contains two keywords "Axioms" and "Rules" that delimit corresponding sections. The axioms are listed as a sequence of strings, separated by commas and enclosed in square brackets. The rules are listed one per line and are in the following format: `state1,u->v,state2`, where

`state1` and `state2` are the states of the graph control, and $u \to v$ is the rewriting rule applied to the string, see Fig 4. The simulator generates a trace of the derivation indicating all new strings obtained during the derivation process. It also allows for tracking the application of rules and for visualizing the graph control, see Fig. 5.

```
Axioms
[S, aSb]
Rules
1,S->aS,2
2,S->Sb,1
1,S->,1
```

Figure 4: Example of the simulator input file

```
 .........
============
STEP 2
============
Component 1:
aaSbb
Component 2:
============
STEP 3
============
Component 1:
aabb
Component 2:
aaaSbb
```

Figure 5: The second and the third step of the simulation of the grammar from Fig. 4

More details and usage examples can be found on the GitHub page of the simulator.

8 Conclusion

The field of insertion-deletion systems has undergone continuous development, with significant advancements made over the past decade. In particular, much of the recent research has focused on regulated insertion-deletion systems, exploring their potential and expanding the boundaries of the model.

We recall that the operations of insertion and deletion are simpler than traditional string rewriting, as they decouple the processes of inserting and deleting in a single step. These operations naturally occur in various domains, including linguistics and biological systems, making them more intuitive and applicable to real-world scenarios.

As the field progresses, several promising directions for future research can be identified. It would be beneficial to explore insertion-deletion systems beyond their traditional role in generating recursively enumerable languages. Their application in linguistics — particularly for modeling agreement languages or ambiguity — and in the generation of biomolecular structures, offers new interdisciplinary perspectives. Additionally, investigating the generation of specific types of languages, such as regular languages, can broaden our understanding of the model's computational scope and yield interesting applications.

Finally, one of the most exciting prospects for future work involves modeling biological processes, particularly the CRISPR-Cas9 gene-editing mechanism. Insertion-deletion systems are well-suited for simulating the insertion and deletion operations integral to CRISPR technology. By exploring this connection, researchers can propose experiments that test the theoretical models in a biological setting, potentially leading to valuable insights for both computational and biological sciences.

Acknowledgments. Artiom Alhazov acknowledges Project 011301 "Information systems based on artificial intelligence" by the Moldova State University.

References

[1] S. Marcus, "Contextual grammars," in *Third International Conference on Computational Linguistics, COLING 1969, Stockholm, Sweden, September 1-4, 1969*, 1969.

[2] L. Kari, "On insertion and deletion in formal languages," Ph.D. dissertation, University of Turku, 1991.

[3] D. Haussler, "Insertion languages," *Information Sciences*, vol. 31, no. 1, pp. 77–89, 1983.

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

[5] R. Motwani, R. Panigrahy, V. Saraswat, and S. Ventkatasubramanian, "On the decidability of accessibility problems (extended abstract)," in *Proceedings of the Thirty-second Annual ACM Symposium on Theory of Computing, STOC*. ACM, 2000, pp. 306–315.

[6] P. Jančar, F. Mráz, M. Plátek, and J. Vogel, "Restarting automata," in *Fundamentals of Computation Theory, FCT*, ser. Lecture Notes in Computer Science, H. Reichel, Ed., vol. 965, 1995, pp. 283–292.

[7] L. Kari and G. Thierrin, "Contextual insertions/deletions and computability," *Information and Computation*, vol. 131, no. 1, pp. 47–61, 1996.

[8] A. Takahara and T. Yokomori, "On the computational power of insertion-deletion systems," *Natural Computing*, vol. 2, no. 4, pp. 321–336, 2003.

[9] L. Kari, Gh. 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, pp. 318–333.

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

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

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

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

[14] F. A. Ran, P. D. Hsu, J. Wright, V. Agarwala, D. A. Scott, and F. Zhang, "Genome engineering using the CRISPR-Cas9 system," *Nature Protocols*, vol. 8, no. 11, pp. 2281–2308, 2013.

[15] M. Adli, "The CRISPR tool kit for genome editing and beyond," *Nature Communications*, vol. 9, no. 1, pp. 1–13, 2018.

[16] A. Alhazov, R. Freund, S. Ivanov, and S. Verlan, "Regulated insertion-deletion systems," *Journal of Automata, Languages and Combinatorics*, vol. 27, no. 1-3, pp. 15–45, 2022.

[17] S. Verlan, "Recent developments on insertion-deletion systems," *Computer Science Journal of Moldova*, vol. 18, no. 2, pp. 210–245, 2010.

[18] G. Rozenberg and A. Salomaa, *Handbook of Formal Languages*. Springer, 1997.

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

[20] R. Freund, M. Kogler, Yu. Rogozhin, and S. Verlan, "Graph-controlled insertion-deletion systems," in *Proceedings Twelfth Annual Workshop on Descriptional Complexity of Formal Systems, DCFS*, ser. EPTCS, I. McQuillan and G. Pighizzini, Eds., vol. 31, 2010, pp. 88–98.

[21] S. Ivanov and S. Verlan, "Universality and computational completeness of controlled leftist insertion-deletion systems," *Fundamenta Informaticae*, vol. 155, no. 1–2, pp. 163–185, 2017.

[22] I. Petre and S. Verlan, "Matrix insertion-deletion systems," *Theoretical Computer Science*, vol. 456, pp. 80–88, 2012.

[23] S. Verlan, H. Fernau, and L. Kuppusamy, "Universal insertion grammars of size two," *Theoretical Computer Science*, vol. 843, pp. 153–163, 2020.

[24] H. Fernau, L. Kuppusamy, and I. Raman, "On the power of generalized forbidding insertion-deletion systems," in *Descriptional Complexity of Formal Systems – 22nd International Conference, DCFS 2020, Vienna, Austria, August 24–26, 2020, Proceedings*, ser. Lecture Notes in Computer Science, G. Jirásková and G. Pighizzini, Eds., vol. 12442. Springer, 2020, pp. 52–63.

[25] L. Kari, "Deletion operations: closure properties," *International Journal of Computer Mathematics*, vol. 52, pp. 23–42, 1994.

[26] M. Margenstern, Gh. Păun, Yu. Rogozhin, and S. Verlan, "Context-free insertion-deletion systems," *Theoretical Computer Science*, vol. 330, no. 2, pp. 339–348, 2005.

[27] S. Verlan, "On minimal context-free insertion-deletion systems," *Journal of Automata, Languages and Combinatorics*, vol. 12, no. 1–2, pp. 317–328, 2007.

[28] Gh. Păun, M. J. Pérez-Jiménez, and T. Yokomori, "Representations and characterizations of languages in Chomsky hierarchy by means of insertion-deletion systems," *International Journal of Foundations of Computer Science*, vol. 19, no. 4, pp. 859–871, 2008.

[29] I. Potapov, O. Prianychnykova, and S. Verlan, "Insertion-deletion systems over relational words," in *Reachability Problems - 10th International Workshop, RP 2016, Aalborg, Denmark, September 19-21, 2016, Proceedings*, ser. Lecture Notes in Computer Science, K. G. Larsen, I. Potapov, and J. Srba, Eds., vol. 9899. Springer, 2016, pp. 177–191.

[30] T. Jurdzinski, "Leftist grammars are non-primitive recursive," in *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II - Track B: Logic, Semantics, and Theory of Programming & Track C: Security and Cryptography Foundations*, ser. Lecture Notes in Computer Science, L. Aceto, I. Damgård, L. A. Goldberg, M. M. Halldórsson, A. Ingólfsdóttir, and I. Walukiewicz, Eds., vol. 5126. Springer, 2008, pp. 51–62. [Online]. Available: https://doi.org/10.1007/978-3-540-70583-3_5

[31] S. Ivanov, "On the power and universality of biologically-inspired models of computation," Ph.D. dissertation, University of Paris Est, 2015.

[32] S. Ivanov and S. Verlan, "On the lower bounds for leftist insertion-deletion languages," *Annals of the University of Bucharest (informatics series)*, vol. LXII, no. 2, pp. 77–88, 2015. [Online]. Available: http://www.lacl.fr/verlan/data/articles/insdelreg2.pdf

[33] F. Biegler, M. J. Burrell, and M. Daley, "Regulated RNA rewriting: Modelling RNA editing with guided insertion," *Theoretical Computer Science*, vol. 387, no. 2, pp. 103–112, 2007.

[34] S. Verlan, "Study of language-theoretic computational paradigms inspired by biology," Habilitation thesis, Université Paris Est, 2010.

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

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

[37] A. Krassovitskiy, Yu. Rogozhin, and S. Verlan, "Computational power of insertion-deletion (P) systems with rules of size two," *Natural Computing*, vol. 10, no. 2, pp. 835–852, 2011.

[38] S. Ivanov and S. Verlan, "Universality of graph-controlled leftist insertion-deletion systems with two states," in *Machines, Computations, and Universality – 7th International Conference, MCU*, ser. Lecture Notes in Computer Science, J. Durand-Lose and B. Nagy, Eds., vol. 9288. Springer, 2015, pp. 79–93.

[39] S. Ivanov and S. Verlan, "Random context and semi-conditional insertion-deletion systems," *Fundamenta Informaticae*, vol. 138, no. 1–2, pp. 127–144, 2015.

[40] E. Dahlhaus and M. K. Warmuth, "Membership for growing context-sensitive grammars is polynomial," *Journal of Computer and System Sciences*, vol. 33, no. 3, pp. 456–472, 1986.

[41] P. Chambart and P. Schnoebelen, "Toward a compositional theory of leftist grammars and transformations," in *Foundations of Software Science and Computational Structures, 13th International Conference, FOSSACS 2010, Paphos, Cyprus, March 20-28, 2010.*

Proceedings, ser. Lecture Notes in Computer Science, C. L. Ong, Ed., vol. 6014. Springer, 2010, pp. 237–251. [Online]. Available: https://doi.org/10.1007/978-3-642-12032-9_17

[42] R. Motwani, R. Panigrahy, V. A. Saraswat, and S. Venkatasubramanian, "On the decidability of accessibility problems (extended abstract)," in *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, F. F. Yao and E. M. Luks, Eds. ACM, 2000, pp. 306–315. [Online]. Available: https://doi.org/10.1145/335305.335341

[43] H. Fernau, L. Kuppusamy, and I. Raman, "Generative power of matrix insertion-deletion systems with context-free insertion or deletion," in *Unconventional Computation and Natural Computation - 15th International Conference, UCNC 2016, Manchester, UK, July 11-15, 2016, Proceedings*, ser. Lecture Notes in Computer Science, M. Amos and A. Condon, Eds., vol. 9726. Springer, 2016, pp. 35–48. [Online]. Available: https://doi.org/10.1007/978-3-319-41312-9_4

[44] H. Fernau, L. Kuppusamy, and I. Raman, "Investigations on the power of matrix insertion-deletion systems with small sizes," *Natural Computing*, vol. 17, no. 2, pp. 249–269, 2018. [Online]. Available: https://doi.org/10.1007/s11047-017-9656-8

[45] H. Fernau, L. Kuppusamy, and I. Raman, "On the generative capacity of matrix insertion-deletion systems of small sum-norm," *Natural Computing*, vol. 20, no. 4, pp. 671–689, 2021. [Online]. Available: https://doi.org/10.1007/s11047-021-09866-y

[46] H. Fernau, L. Kuppusamy, and I. Raman, "On the generative power of graph-controlled insertion-deletion systems with small sizes," *Journal of Automata, Languages and Combinatorics*, vol. 22, no. 1-3, pp. 61–92, 2017. [Online]. Available: https://doi.org/10.25596/jalc-2017-061

[47] H. Fernau, L. Kuppusamy, and I. Raman, "On the computational completeness of graph-controlled insertion-deletion systems with

binary sizes," *Theoretical Computer Science*, vol. 682, pp. 100–121, 2017. [Online]. Available: https://doi.org/10.1016/j.tcs.2017.01.019

[48] J. Dassow and Gh. Păun, *Regulated Rewriting in Formal Language Theory*. Springer, 1989. [Online]. Available: https://www.springer.com/de/book/9783642749346

[49] B. Haberstroh, "Elementary programmed graph grammars," Master's thesis, TU Wien, 1990.

[50] A. Alhazov, A. Krassovitskiy, Yu. Rogozhin, and S. Verlan, "P systems with minimal insertion and deletion," *Theoretical Computer Science*, vol. 412, no. 1-2, pp. 136–144, 2011.

[51] R. Freund, Yu. Rogozhin, and S. Verlan, "Generating and accepting P systems with minimal left and right insertion and deletion," *Natural Computing*, vol. 13, no. 2, pp. 257–268, 2014. [Online]. Available: http://dx.doi.org/10.1007/s11047-013-9396-3

[52] H. Fernau, L. Kuppusamy, and I. Raman, "When stars control a grammar's work," in *Proceedings of the 16th International Conference on Automata and Formal Languages, AFL 2023, Eger, Hungary, September 5-7, 2023*, ser. EPTCS, Z. Gazdag, S. Iván, and G. Kovásznai, Eds., vol. 386, 2023, pp. 96–111. [Online]. Available: https://doi.org/10.4204/EPTCS.386.9

[53] H. Fernau, L. Kuppusamy, and I. Raman, "On path-controlled insertion-deletion systems," *Acta Informatica*, vol. 56, no. 1, pp. 35–59, 2019. [Online]. Available: https://doi.org/10.1007/s00236-018-0312-2

[54] H. Fernau, L. Kuppusamy, and I. Raman, "Computational completeness of simple semi-conditional insertion-deletion systems," in *Unconventional Computation and Natural Computation – 17th International Conference, UCNC 2018, Fontainebleau, France, June 25–29, 2018, Proceedings*, ser. Lecture Notes in Computer Science, S. Stepney and S. Verlan, Eds., vol. 10867. Springer, 2018, pp. 86–100.

[55] H. Fernau, L. Kuppusamy, and I. Raman, "Computational completeness of simple semi-conditional insertion-deletion systems of degree (2, 1)," *Natural Computing*, vol. 18, no. 3, pp. 563–577, 2019.

[56] S. Ivanov and S. Verlan, "Single semi-contextual insertion-deletion systems," *Natural Computing*, vol. 20, no. 4, pp. 703–712, 2021.

[57] E. Csuhaj-Varjú and J. Dassow, "On cooperating/distributed grammar systems," *Journal of Information Processing and Cybernetics EIK*, vol. 26, no. 1/2, pp. 49–63, 1990.

[58] M. Vu and H. Fernau, "Insertion-deletion with substitutions II: about the role of one-sided context," *Journal of Automata, Languages and Combinatorics*, vol. 28, no. 1-3, pp. 221–244, 2023. [Online]. Available: https://doi.org/10.25596/jalc-2023-221

[59] M. Vu and H. Fernau, "Insertion-deletion systems with substitutions I," *Computability*, vol. 11, no. 1, pp. 57–83, 2022.

[60] M. Vu and H. Fernau, "Adding matrix control: Insertion-deletion systems with substitutions III," *Algorithms*, vol. 14, no. 5, p. 131, 2021. [Online]. Available: https://doi.org/10.3390/a14050131

[61] J. Castellanos, C. Martín-Vide, V. Mitrana, and J. M. Sempere, "Networks of evolutionary processors," *Acta Informatica*, vol. 39, no. 6-7, pp. 517–529, 2003.

[62] B. Truthe, "A survey on computationally complete accepting and generating networks of evolutionary processors," in *Machines, Computations, and Universality - 9th International Conference, MCU 2022, Debrecen, Hungary, August 31 - September 2, 2022, Proceedings*, ser. Lecture Notes in Computer Science, J. Durand-Lose and Gy. Vaszil, Eds., vol. 13419. Springer, 2022, pp. 12–26.

[63] S. Ivanov, Yu. Rogozhin, and S. Verlan, "Small universal networks of evolutionary processors," *Journal of Automata, Languages and Combinatorics*, vol. 19, no. 1-4, pp. 133–144, 2014.

[64] "GSRsim simulator." [Online]. Available: https://github.com/sverlan/gcrsim

Artiom Alhazov, Sergiu Ivanov, Sergey Verlan,

Received September 14, 2024
Revised October 21, 2024
Accepted October 23, 2024

Artiom Alhazov
ORCID: https://orcid.org/0000-0002-6184-3971
State University of Moldova,
Vladimir Andrunachievici Institute of Mathematics and Computer Science
Academiei 5, Chișinău, MD-2028, Moldova
E–mail: artiom@math.md

Sergiu Ivanov
ORCID: https://orcid.org/0000-0002-1537-6508
Université Paris-Saclay, Université Évry,
IBISC, 91020, Évry-Courcouronnes, France
E–mail: sergiu.ivanov@ibisc.univ-evry.fr

Sergey Verlan
ORCID: https://orcid.org/0000-0001-7800-1618
University of Paris Est Creteil
Univ Paris Est Creteil, LACL, F-94010, Creteil, France
E–mail: verlan@u-pec.fr