Universality of Graph-controlled Leftist Insertion-deletion Systems with Two States

Sergiu Ivanov[1] and Sergey Verlan[1,2](✉)

[1] Laboratoire d'Algorithmique, Complexité et Logique, Université Paris Est – Créteil Val de Marne, 61, av. gén. de Gaulle, 94010 Créteil, France
sergiu.ivanov@u-pec.fr

[2] Institute of Mathematics and Computer Science, Academy of Sciences of Moldova, Academiei 5, Chisinau MD-2028, Moldova
verlan@u-pec.fr

Abstract. In this article, we consider leftist insertion-deletion systems, in which all rules have contexts on the same side, and may only insert or delete one symbol at a time. We start by introducing extended rules, in which the contexts may be specified as regular expressions, instead of fixed words. We then prove that leftist systems with such extended rules and two-state graph control can simulate any arbitrary 2-tag system. Finally, we show how our construction can be simulated in its turn by graph-controlled leftist insertion-deletion systems with conventional rules of sizes $(1, 1, 0; 1, 2, 0)$ and $(1, 2, 0; 1, 1, 0)$ (where the first three numbers represent the maximal size of the inserted string and the maximal size of the left and right contexts respectively, while the last three numbers provide the same information about deletion rules), which implies that the latter systems are universal.

1 Introduction

Abstract insertion and deletion operations are simple, yet powerful, special cases of string rewriting rules. Intuitively, insertion is adding a substring at a site having a specified left, right, or both contexts, while deletion is removing a substring from a site having a specified left, right, or both contexts. The precursor of insertion was context adjoining, first introduced by S. Marcus in the seminal paper [19] with a linguistic motivation, and then further developed in [24,25]. The modern definition of insertion was introduced in [6] in the form of semi-contextual grammars.

The works [7,8] defined insertion differently, by generalising Kleene's operations of concatenation and closure [15]. Indeed, insertion can be seen as concatenation which is allowed to happen anywhere in the string. Following a similar approach, the work [12] introduced the dual operation of deletion as a generalised quotient operation which does not necessarily happen at the ends of the string. The paper [14] first considers systems containing finite sets of insertion

J. Durand-Lose and B. Nagy (Eds.): MCU 2015, LNCS 9288, pp. 79–93, 2015.
DOI: 10.1007/978-3-319-23111-2_6

and deletion rules working together: insertion-deletion systems. Such a system works in generative mode: it sequentially applies insertion or deletion rules to one of its finitely many axioms; the generated language includes all the terminal words obtained in the process.

Exciting sources of motivation for studying insertion and deletion operations were found in biology [4,13,26,28]. A well known one is the theoretically conceived process of mismatched annealing of DNA, which effectively results in insertions or deletions of certain segments of the strands [26]. In the formal framework of insertion and deletion operations, such modifications to DNA strands are modelled by context-free rules, i.e. insertion and deletion rules which can be applied anywhere in the string. The expressive power of such context-free operations was studied in the article [20], for example, which shows that context-free insertion-deletion operations of sizes $(3, 0, 0; 2, 0, 0)$ and $(2, 0, 0; 3, 0, 0)$ can simulate arbitrary string rewriting rules and are thus computationally complete. For a detailed overview of the computational power of context-free insertion and deletion operations, the reader is referred to [29]. For surveys of results on insertion-deletion systems in general, we refer to [16,30].

Another biological phenomenon which can be seen as a sequence of insertions and deletions is RNA editing, which was discovered in some species of protozoa [1,2]. RNA editing consists in inserting or deleting fragments of messenger RNA, and is guided by an anchor segment always located on one side of the edited locus. This directly motivates the study of one-sided insertion-deletion systems, i.e. systems in which all rules must have the context on one and the same side. The works [17,18,21] investigate the power of such systems and give several computational completeness results as a function of the size of the rules, as well as describe some families of one-sided systems which are not computationally complete. For these families, additional control mechanisms can be considered which often increase the expressive power, for example, matrix control [23], or semi-conditional and random context control [10].

One of the most frequently discussed variant of controlled insertion and deletion are *graph-controlled* insertion-deletion systems (sometimes also called insertion-deletion P systems). The work [18] shows that five-state graph control increases the power of small one-sided insertion-deletion rules to computational completeness. In [5], this result is improved upon and four-state graph control is shown to suffice for generating all recursively enumerable languages. The article [11] considers insertion-deletion systems of sizes $(1, 2, 0; 1, 1, 0)$ and $(1, 1, 0; 1, 2, 0)$, and proves that adding three-state graph-control to them results in computationally complete devices.

In this paper, we focus on a special variant of one-sided insertion-deletion systems, introduced in [9]: *leftist* systems. In such systems, the rules can only insert or delete one symbol at a time, and must use contexts on the same side. In [9], it is shown that all leftist insertion-deletion systems can be simulated by systems of sizes $(1, 2, 0; 1, 1, 0)$ and $(1, 1, 0; 1, 2, 0)$, and that these systems can generate all regular languages. Moreover, [9, Theorem 3.3] shows that leftist systems can generate non-context-free languages.

We continue the exploration of the expressive power of leftist systems and show that that adding two-state graph control to insertion-deletion systems of sizes $(1, 2, 0; 1, 1, 0)$ and $(1, 1, 0; 1, 2, 0)$ renders them capable of simulating any 2-tag system (Theorems 4 and 5). That 2-tag systems are universal [3,22] implies therefore the universality of two-state graph-controlled leftist insertion-deletion systems. Our proofs are based on an extension to insertion and deletion rules we introduce in Definition 1, which allows the specification of contexts as regular expressions instead of single fixed words. We show that graph-controlled leftist systems with such extended contexts can simulate any 2-tag system (Theorem 1). Interestingly, in the non-controlled case, allowing extended context in leftist systems does not augment computational power, as normal leftist systems can simulate regular contexts (Theorems 2 and 3). While the same argument does not generally work in the graph-controlled case, it does apply to the construction from the proof of Theorem 1, which leads to the universality result mentioned above.

2 Preliminaries

We do not present here definitions concerning standard concepts of the theory of formal languages and we refer to [27] for more details. We denote by $|w|$ the length of a word w, by $card(A)$ the cardinality of the alphabet A and by REG, CF, CS, and RE the families of regular, context-free, context-sensitive, and recursively enumerable languages, respectively.

An *m-tag system* is the tuple $TS = (m, \mathcal{A}, P)$, where m is a positive integer, $\mathcal{A} = \{a_1, \dots, a_{n+1}\}$ is a finite alphabet, and P contains rules of the form $a_i \to \alpha_i$, where $\alpha_i \in A^*$, for $1 \leq i \leq n$. The letter a_{n+1} is called the halting symbol.

A configuration of the tag system TS is a word $w \in \mathcal{A}^*$. The system passes from the configuration $w = a_{i_1} \dots a_{i_m} w'$, $1 \leq i_j \leq n+1$, $1 \leq j \leq m$ to the next configuration z by applying one of the productions $a_i \to \alpha_i$: the first m symbols of w are erased and α_i is added to the end of the word: $w \Longrightarrow z$, if $z = w'\alpha_i$.

A computation of TS over the word $x \in V^*$ is a sequence of configurations $x \Longrightarrow \dots \Longrightarrow y$, where either $y = a_{n+1} a_{i_1} \dots a_{i_{m-1}} y'$, or $|y| < m$. In this case we say that TS halts on x and that y is the result of the computation of TS over x, which is denoted by $y = TS(x)$.

Minsky proved that 2-tag systems are universal [3,22]. Moreover, according to his proof, it is sufficient to consider only tag systems that halt only on the halting symbol and do not have empty productions.

An *insertion-deletion system* is a construct $ID = (V, T, A, I, D)$, where:

- V is an alphabet;
- $T \subseteq V$ is the *terminal* alphabet (the symbols from $V \setminus T$ are called *non-terminals*);
- $A \subseteq V^*$ is the set of *axioms*;
- I, D are finite sets of triples of the form (u, α, v), where u, α $(\alpha \neq \lambda)$, and v are strings over V.

The triples in I are *insertion rules*, and those in D are *deletion rules*. An insertion rule $(u, \alpha, v)_{ins} \in I$ indicates that the string α can be inserted between u and v (which corresponds to the rewriting rule $uv \to u\alpha v$), while a deletion rule $(u, \alpha, v)_{del} \in D$ indicates that α can be removed from between the contexts u and v (which corresponds to the rewriting rule $u\alpha v \to uv$). By $\Longrightarrow$ we denote the relation defined by the insertion or deletion rules and by $\Longrightarrow^*$ the reflexive and transitive closure of $\Longrightarrow$.

The language generated by $ID = (V, T, A, I, D)$ is defined by

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

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

$$\begin{aligned} n &= \max\{|\alpha| \mid (u, \alpha, v)_{ins} \in I\}, \ \ p = \max\{|\alpha| \mid (u, \alpha, v)_{del} \in D\}, \\ m &= \max\{|u| \mid (u, \alpha, v)_{ins} \in I\}, \ \ q = \max\{|u| \mid (u, \alpha, v)_{del} \in D\}, \\ m' &= \max\{|v| \mid (u, \alpha, v)_{ins} \in I\}, \ \ q' = \max\{|v| \mid (u, \alpha, v)_{del} \in D\}. \end{aligned}$$

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

If one of the parameters n, m, m', p, q, or q' is not bounded, then we write instead the symbol $*$. If one of the numbers from the pairs m, m' and q, q' is equal to zero (while the other is not), then we say that the corresponding families have a one-sided context. If the $m' = q' = 0$, and $n = p = 1$, the insertion-deletion systems are called leftist.

We also recall that the family of insertion-deletion languages of size $(1, 1, 0; 1, 1, 0)$ is incomparable with REG: $(CF \setminus REG) \cap INS_1^{1,0}DEL_0^{0,0} \neq \emptyset$ and $(ba)^+ \notin INS_1^{1,0}DEL_1^{1,0}$ [18].

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

The relation $\{(i, j) \mid (i, r, E) \in R \text{ and } j \in E\}$ defines a graph, called the communication graph of the system. We remark that in the literature the term "graph control" often implies that there is a one-to-one correspondence between the label i and the rule $(i, r, E) \in R$. This corresponds to the point of view that the rules are located on the edges of the communication graph. Another point of view is to place the rules in the nodes of the communication graph. In this paper we do not consider such restrictions, as all corresponding models are equivalent in the computational power and have mostly identical descriptional complexity parameters.

As is common for graph controlled systems, a configuration of Π is represented by a pair (w, i), where $i \in H$ is the current state and w is the current string. A transition $(w, i) \Longrightarrow (w', j)$ is performed if there is a rule $(i, (u, \alpha, v)_t, E)$ 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 state from an axiom and the initial label, i.e.,

$$L(\Pi) = \{w \in T^* \mid (w_0, i_0) \Longrightarrow^* (w, i_f),\ \text{for some } w_0 \in A,\ i_f \in I_f\}.$$

The family of languages generated by graph-controlled insertion-deletion systems having k states and insertion/deletion rules of size $(n, m, m'; p, q, q')$ is denoted as $GC_kINS_n^{m,m'}DEL_p^{q,q'}$. As we deal with universality we define the notion of the *computation* of a system Π on an input word w with respect to a recursive coding ϕ, denoted as $\Pi(\phi(w))$. This can be obtained by replacing the set of axioms A from the definition of Π by $\{\phi(w)\}$ and then by evolving Π as usual, $L(\Pi)$ being considered as the result of the computation.

3 Universality Results

In this section we extend insertion and deletion rules to contexts given by regular expressions rather than by simple strings.

Definition 1. *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.*

A (graph-controlled) insertion-deletion system with regular contexts is a (graph-controlled) insertion-deletion system in which extended insertion and deletion rules are allowed.

We will use the same notation for families of languages generated by insertion-deletion systems with regular contexts as for those generated by conventional ones: $INS_n^{m,m'}DEL_p^{q,q'}$, where m, m', q, and q' will be replaced by REG if the corresponding contexts of insertion or deletion rules are allowed to contain regular expressions.

We will now show that any 2-tag system can be simulated by a graph-controlled insertion-deletion system with regular contexts. Next, we show that regular contexts can be reduced to contexts of size $(1, 1, 0; 1, 2, 0)$ or $(1, 2, 0; 1, 1, 0)$.

Theorem 1. *For every tag system TS, there exists a two-state graph-controlled insertion-deletion system Π of size* $(1, REG, 0; 1, REG, 0)$ *and a recursive coding ϕ such that the following conditions hold:*

– *$\Pi(\phi(w)) = \{TS(w)\}$, if TS halts on w, and*
– *$\Pi(\phi(w)) = \emptyset$, if TS does not halt on w.*

Proof. Consider an arbitrary tag system $TS = (2, \mathcal{A}, P)$. We will now construct the extended graph-controlled insertion-deletion system $\Pi(\beta) = (V, \mathcal{A}, \emptyset, \{1,2\}, 1, \{1\}, R)$ simulating the computation of TS on the word $\beta \in \mathcal{A}^*$. The alphabet V is defined as follows:

$$V = \{\bar{B}, B, S, F, E, E', Z, Z'\} \cup \{R_i, R_i', R_i'', R_i''', P_i \mid a_i \to \alpha_i \in P\} \cup \mathcal{A}.$$

The coding ϕ is defined as $\phi(\beta) = \bar{B}B\,\beta\,SEE'\,F$.

The simulation of TS happens in several phases; accordingly, we split the rules in R into the following logical groups:

1. generation of the control string of alternating Z and Z', for each $\mathbf{a} \in \mathcal{A}$:

$$\begin{array}{l}
r_{11} : \big(1,\ (S,\ Z,\ \lambda)_{ins},\ \{1\}\big),\\
r_{12} : \big(1,\ (S,\ Z',\ \lambda)_{ins},\ \{1\}\big),\\
r_{13} : \big(1,\ (\mathbf{a},\ S,\ \lambda)_{del},\ \{1\}\big),\\
r_{14} : \big(1,\ (\mathbf{a}(ZZ')^*,\ E,\ \lambda)_{del},\ \{1\}\big),\\
r_{15} : \big(1,\ (Z',\ E',\ \lambda)_{del},\ \{2\}\big);
\end{array}$$

2. deletion of two symbols at the left end of the string and generation of the signal R_i, for each $\mathbf{a} \in \mathcal{A}$:

$$\begin{array}{l}
r_{21} : \big(2,\ (B,\ R_i',\ \lambda)_{ins},\ \{2\}\big),\\
r_{22} : \big(2,\ (R_i'a_i,\ R_i'',\ \lambda)_{ins},\ \{2\}\big),\\
r_{23} : \big(2,\ (R_i''\mathbf{a},\ R_i''',\ \lambda)_{ins},\ \{2\}\big),\\
r_{24} : \big(2,\ (R_i'',\ \mathbf{a},\ \lambda)_{del},\ \{1\}\big),\\
\\
r_{25} : \big(1,\ (R_i',\ a_i,\ \lambda)_{del},\ \{1\}\big),\\
r_{26} : \big(1,\ (R_i',\ R_i'',\ \lambda)_{del},\ \{1\}\big),\\
r_{27} : \big(1,\ (B,\ R_i',\ \lambda)_{del},\ \{1\}\big),\\
r_{28} : \big(1,\ (BR_i''',\ R_i,\ \lambda)_{ins},\ \{2\}\big),\\
\\
r_{29} : \big(2,\ (B,\ R_i''',\ \lambda)_{del},\ \{1\}\big);
\end{array}$$

3. insertion of the right-hand side of a production of TS, for each $\mathbf{a}, \mathbf{b} \in \mathcal{A}$, and $a_i \to \alpha_i \in P$:

$$\begin{array}{l}
r_{31} : \big(1,\ (R_i\mathbf{a},\ R_i,\ \lambda)_{ins},\ \{1\}\big),\\
r_{32} : \big(1,\ (\mathbf{a},\ R_i,\ \lambda)_{del},\ \{1\}\big),\\
r_{33} : \big(1,\ (B,\ R_i,\ \lambda)_{del},\ \{1\}\big),\\
r_{34} : \big(1,\ (B\mathcal{A}^*R_iZ,\ P_i,\ \lambda)_{ins},\ \{1\}\big),\\
r_{35} : \big(1,\ (\mathbf{a}ZP_i,\ b,\ \lambda)_{ins},\ \{1\}\big),\\
r_{36} : \big(1,\ (\mathbf{a},\ Z,\ \lambda)_{del},\ \{1\}\big),\\
r_{37} : \big(1,\ (\mathbf{a}P_i\alpha_i,\ Z',\ \lambda)_{del},\ \{1\}\big),\\
r_{38} : \big(1,\ (\mathbf{a},\ P_i,\ \lambda)_{del},\ \{2\}\big),
\end{array}$$

4. checking of the halting condition and cleanup, where a_{n+1} is the halting symbol of TS:

$$\begin{aligned} r_{41} &: \big(2,\ (\bar{B}, && B,\ \lambda)_{del},\ \{1\}\big),\\ r_{42} &: \big(1,\ (\bar{B}a_{n+1}\mathcal{A}^*, && F,\ \lambda)_{del},\ \{1\}\big),\\ r_{43} &: \big(1,\ (\emptyset, && \bar{B},\ \emptyset)_{del},\ \{1\}\big). \end{aligned}$$

The simulation of the tag system TS by Π is done in 3 stages. During the first stage a repeating sequence of words ZZ' is inserted, their number being equal to the number of steps TS needs to reach the final configuration. This corresponds to the following computation in Π: $(\bar{B}B\,w\,SEE'\,F, 1) \Longrightarrow^* (\bar{B}Bw(ZZ')^kF, 2)$. The second stage repeatedly simulates the application of a production $a_i \to \alpha_i$ by erasing the two starting symbols and by adding the corresponding appendant to the end: $(\bar{B}Ba_ia_jw'(ZZ')^kF, 2) \Longrightarrow^* (\bar{B}Bw'\alpha_i(ZZ')^{k-1}F, 2)$. During the last stage the markers $\bar{B}$, B, and F are removed after checking that the first letter of the word is a_{n+1}: $(\bar{B}Ba_{n+1}w'F, 1) \Longrightarrow^* (a_{n+1}w', 1)$.

Now we will discuss each stage in more details. The simulation of the tag system TS starts with the rules of group (1). The symbol S inserts a sequence of the form $(Z|Z')^*$ by rules r_{11} and r_{12}, and is deleted by r_{13}. Then the non-terminal symbol E is deleted permitting to verify that S has inserted an alternating sequence of the form $(ZZ')^*$. Finally, E' is erased by rule r_{15}, moving the system in state 2 and starting the second stage. This sequence of actions corresponds to the following derivation:

$$\begin{aligned} &(\bar{B}B\,wa\,\underline{S}EE'\,F, 1) \overset{r_{11}}{\underset{}{\overset{r_{12}}{\Longrightarrow}}}{}^*(\bar{B}B\,w\underline{a}\,S(ZZ')^kEE'\,F, 1)\\ \overset{r_{13}}{\Longrightarrow}\ &(\bar{B}B\,w\underline{a\,(ZZ')^k}EE'\,F, 1) \overset{r_{14}}{\Longrightarrow} (\bar{B}B\,wa\,(ZZ')^{k-1}Z\underline{Z'}E'\,F, 1)\\ \overset{r_{15}}{\Longrightarrow}\ &(\bar{B}B\,wa\,S(ZZ')^k\,F, 2), \end{aligned}$$

where $k \in \mathbb{N}$ and underlining indicates the left context of the rule application effecting the transition into the next configuration.

Remark that for the rules from state 1 and from groups (2) and (3) to be applicable, some symbols must be present which may only be inserted in state 2, so if S does not insert the correct alternating sequence, E and E' cannot be erased, and Π blocks on a string with non-terminals. The only exception is r_{36} which can be applied at any time after S is erased, but, as we will see later, if this deletion happens at the incorrect moment, the system will block as well. Rules r_{41} and r_{43}, on the other hand, are also applicable at any moment, but if they are applied too early (while B is still needed), the string will never reach the form required for F to be deleted by r_{42}.

Now we consider the second stage of simulation. During this stage the role of symbols Z and Z' is to ensure that every deletion of the substring $a_i\mathbf{a}$ at the beginning of the string is followed by an insertion of the corresponding α_i at the right end of the string, for $a_i \to \alpha_i \in P$. The string should thus contain as many pairs ZZ' as there are steps in a halting computation of TS starting with w. Remark that, after r_{15} is applied, the rules of group (1) can never become applicable again as there are no more necessary symbols.

Whenever $\mathit{\Pi}$ is in state 2 with a string of the form $\bar{B}Ba_i\mathbf{a}w\,(ZZ')^kF$, rules r_{21} through r_{24} can only be applied, and necessarily in the following order (we only show the evolution of a prefix of $\bar{B}Ba_i\mathbf{a}\,(ZZ')^kF$, because, in state 2, $\mathit{\Pi}$ cannot change anything outside it):

$$\bar{B}\underline{B}a_i\mathbf{a} \xRightarrow{r_{21}} \bar{B}B\underline{R_i'a_i}\mathbf{a} \xRightarrow{r_{22}} \bar{B}BR_i'a_i\underline{R_i''\mathbf{a}} \xRightarrow{r_{23}} \bar{B}BR_i'a_i\underline{R_i''}\mathbf{a}R_i''' \xRightarrow{r_{24}} \bar{B}BR_i'a_iR_i''R_i'''.$$

The application of r_{24} moves the system back into state 1. The rules of group (3) are not applicable at this moment, because the string contains no instances of R_i yet. The two rules of the second group which can be applied are r_{25} and r_{27}; remark though that applying r_{27} removes R_i', so r_{26} cannot be applied to erase R_i'' anymore. Given that a_i must be erased in order to enable the deletion of R_i'' and the insertion of R_i, the following evolution is the only possible one in a terminal derivation (again, we only show the evolution of the prefix):

$$\bar{B}B\underline{R_i'}a_iR_i''R_i''' \xRightarrow{r_{25}} \bar{B}B\underline{R_i'}R_i''R_i''' \xRightarrow{r_{26}} \bar{B}\underline{B}R_i'R_i''' \xRightarrow{r_{27}} \bar{B}\underline{BR_i'''} \xRightarrow{r_{28}} \bar{B}BR_i'''R_i,$$

where the application of r_{28} moves the system back into state 2. This time, however, B is separated from the rest of the string by an instance of R_i''', so if rule r_{21} is applied instead of r_{29}, neither r_{24} nor r_{29} will ever become applicable, and $\mathit{\Pi}$ will block in state 2 on a string with non-terminals. Thus the system has to apply r_{29} immediately after the application of r_{28} to arrive in state 1 with the string $\bar{B}BR_iw(ZZ')^kF$, thereby successfully completing the deletion of $a_i\mathbf{a}$ and introducing the corresponding signal symbol R_i.

Rules r_{31}, r_{32}, and r_{33} move the signal R_i to the right end of the string. Remark that if r_{32} and r_{33} erase all the instances of R_i before r_{34} is applied, $\mathit{\Pi}$ just blocks on a string with non-terminals. On the other hand, the context of r_{34} requires that, for P_i to be inserted, there should be no extra signal symbols in the string; this assures that exactly one insertion happens at the right end of the string per deletion at the left end.

The correct sequence of actions triggered by a signal symbol R_i at the right end of the string is as follows (we only show the evolution of the suffix, because all rules modifying the left end of the string in state 1 require primed R_i symbols):

$$\begin{aligned} &\underline{\ldots\mathbf{a}R_iZ}Z'\,(ZZ')^{k-1}F \xRightarrow{r_{34}} \underline{\mathbf{a}}R_iZP_iZ'\,(ZZ')^{k-1}F \xRightarrow{r_{32}} \underline{\mathbf{a}ZP_i}Z'\,(ZZ')^{k-1}F \\ \xRightarrow{r_{35}}{}^{*}\ &\underline{\mathbf{a}}ZP_i\alpha_iZ'\,(ZZ')^{k-1}F \xRightarrow{r_{36}} \underline{\mathbf{a}P_i\alpha_i}Z'\,(ZZ')^{k-1}F \xRightarrow{r_{37}} \underline{\mathbf{a}}P_i\alpha_i\,(ZZ')^{k-1}F \\ \xRightarrow{r_{38}}\ &\mathbf{a}\alpha_i\,(ZZ')^{k-1}F, \end{aligned}$$

where the last derivation step moves the system into state 2 and initiates the next deletion at the left end of the string. Remark that r_{35} is only applicable after R_i has been erased. Furthermore, even though r_{36} may delete Z almost at any moment when $\mathit{\Pi}$ is in state 1, if this does occur, then both r_{34} and r_{35} are rendered inapplicable, and $\mathit{\Pi}$ will end up blocking in state 1 on a string with non-terminals. Rule r_{37} can only erase Z' when applications of r_{35} insert the exact substring α_i from the production $a_i \to \alpha_i$. If Z' is not erased, the signal symbol R_j of the following simulation step will not be able to use r_{34} to initiate the insertion of the right-hand side α_j, and Π will block. Finally, the application of r_{38} moves the system into state 2, enabling the next deletion at the left end of the string.

The last stage of the computation is assured by the rules of group (4). Rule r_{41} is applied non-deterministically in order to disable any further deletions and insertions. Then, the end marker F is erased only if the string contains no more service symbols, no more Z or Z', and if the first symbol after $\bar{B}$ is the halting symbol of Π. If these conditions are not met, F will never be erased and Π will block. If F is successfully erased, however, the rule r_{43} is applied removing the last non-terminal symbol and finalizing the simulation of TS.

We now show that, in the case of one-sided systems, regular contexts do not bring additional computational power.

Theorem 2. $INS_1^{REG,0}DEL_1^{REG,0} \subseteq INS_1^{2,0}DEL_1^{1,0}$.

Proof. We give here only the sketch of the proof of the statement which is based on the proof of the result $REG \subsetneq INS_1^{2,0}DEL_1^{1,0}$ from [9].

Any rule $r : (E, x, \lambda)_t$, $t \in \{ins, del\}$, can be simulated as follows. Let $FA = (Q, T, q_0, F, \delta)$ be the finite automaton such that $L(FA) = L(E)$. Consider the following sets of rules:

$$\begin{aligned} I =& \{(a,\, Q_0,\, \lambda)_{ins} \mid a \in T\} \cup \{(Q_i a,\, Q_j,\, \lambda)_{ins} \mid q_j \in \delta(q_i, a)\} \\ & \cup \{(Q_f,\, x,\, \lambda)_{ins} \mid f \in F,\ \text{if } t = ins\}, \\ D =& \{(a,\, Q_i,\, \lambda)_{del} \mid a \in T\} \cup \{(Q_f,\, x,\, \lambda)_{del} \mid f \in F,\ \text{if } t = del\}. \end{aligned}$$

We claim that these rules faithfully simulate the action of the extended rule r. The simulation starts by inserting the symbol Q_0 that marks the guess for the leftmost position for the recognition of context E. Then the string is decorated by symbols Q_i according to the transitions of FA. This allows to check if the string to the right of Q_0 belongs to E. In this case a symbol Q_f, $f \in F$ is ultimately inserted into the string. Now this symbol can insert or delete x according to the type t of the rule. Finally, symbols Q_i are cleaned up.

The validity of the simulation is based on the observation that if the full sequence of insertions (checking the contexts) is not performed, then the rule is not applied. Moreover, if the clean-up phase is not completed, then the string will contain non-terminals that will block the corresponding portion of the string from any further evolution.

A similar theorem holds in the case of systems of size $(1, 1, 0; 1, 2, 0)$. It could be immediately deduced from the previous theorem and [9, Lemma 3.3]; we would like to present a simpler construction, however.

Theorem 3. $INS_1^{REG,0}DEL_1^{REG,0} \subseteq INS_1^{1,0}DEL_1^{2,0}$.

Proof. Like for the previous theorem, we shall only give the sketch of the proof.

Any rule $r : (E, x, \lambda)_t, t \in \{ins, del\}$, can be simulated as follows. Let $FA = (Q, T, q_0, F, \delta)$ be the finite automaton such that $L(FA) = L(E)$. Consider the following sets of rules:

$$\begin{aligned}
I =&\{(a,\, Q_i,\, \lambda)_{ins} \mid a \in T\} \cup \{(Q_f,\, x,\, \lambda)_{ins} \mid f \in F,\ \text{if } t = ins\},\\
D =&\{(Q_i a,\, Q_j,\, \lambda)_{del} \mid q_j \in \delta(q_i, a)\} \cup \{(a,\, Q_0,\, \lambda)_{del} \mid a \in T\}\\
&\cup\{(Q_f,\, x,\, \lambda)_{del} \mid f \in F,\ \text{if } t = del\}.
\end{aligned}$$

We claim that these rules simulate the action of r faithfully. The simulation strategy is a bit different from the proof of Theorem 2. First, a guess about the context is made and the string is decorated by the sequence of symbols Q_i. When the symbol Q_f, $f \in F$, corresponding to final state of E is inserted, an insertion or deletion of x can be performed. Finally, the validity of the context is checked by the deletion rules that require a valid accepting path of FA to be present to the left of Q_f. The difference from Theorem 2 is that at first the symbols Q_i are randomly inserted into the string, and only after that the deletion rules check that these symbols were inserted in the correct order. In particular, this means that the insertion or deletion of x can happen even if the left context does not satisfy E. However, in this case it will be impossible to erase the remaining non-terminals Q_i.

While insertion-deletion systems of sizes $(1, 2, 0; 1, 1, 0)$ and $(1, 1, 0; 1, 2, 0)$ can simulate any extended insertion-deletion system of size $(1, REG, 0; 1, REG, 0)$, the same statement cannot be directly extended to the graph-controlled case. Indeed, the simulation of extended contexts is based on inserting additional symbols; repeating the same approach for graph-controlled systems would make it possible to switch states right in the middle of the verification of a context of an extended rule allowing some incorrect behavior. For example consider a system containing the following three rules:

$$1 : \big(1, (ab,\, x,\, \lambda)_{ins}, \{2\}\big), \quad 2 : \big(2, (d,\, a,\, \lambda)_{ins}, \{1\}\big), \quad 3 : \big(1, (a,\, a,\, \lambda)_{ins}, \{2\}\big),$$

and suppose that we use the approach from the proof of Theorem 3 to simulate rule 1 with rules of size $(1, 1, 0; 1, 2, 0)$. We will use the following rules to insert state symbols:

$$\begin{aligned}
1a &: \big(1,\ (b,\, Q_2,\, \lambda)_{ins},\ \{1\}\big), &\quad 1b &: \big(1,\ (a,\ \ Q_1,\, \lambda)_{ins},\ \{1\}\big),\\
1c &: \big(1,\ (\mathbf{a},\, Q_0,\, \lambda)_{ins},\ \{1\}\big), &\quad 1d &: \big(1,\ (Q_2,\, x,\ \ \lambda)_{ins},\ \{2\}\big),
\end{aligned}$$

for all symbols $\mathbf{a}$, and the following deletion rules to attempt to verify the context of rule 1:

$$\begin{aligned}
1e &: \big(2,\ (Q_1 b,\, Q_2,\, \lambda)_{del},\ \{2\}\big), &\quad 1f &: \big(2,\ (Q_0 a,\, Q_1,\, \lambda)_{del},\ \{2\}\big),\\
1g &: \big(2,\ (\mathbf{a},\quad Q_0,\, \lambda)_{del},\ \{2\}\big),
\end{aligned}$$

for all symbols $\mathbf{a}$. In the configuration $(db, 1)$ of the original system, no rule is applicable. However, in the new system with rules of size $(1, 1, 0; 1, 2, 0)$, the following sequence of rule applications is possible:

$$\begin{aligned}
&(d\underline{b}, 1) \xRightarrow{1a} (db\underline{Q_2}, 1) \xRightarrow{1d} (\underline{d}bQ_2x, 2) \xRightarrow{2} (d\underline{a}bQ_2x, 1) \xRightarrow{3} (\underline{d}aabQ_2x, 2)\\
&\xRightarrow{2} (daa\underline{a}bQ_2x, 1) \xRightarrow{1b} (da\underline{a}aQ_1bQ_2x, 1) \xRightarrow{1c} (daaQ_0a\underline{Q_1b}Q_2x, 1)\\
&\xRightarrow{1e} (daa\underline{Q_0a}Q_1bx, 1) \xRightarrow{1f} (da\underline{a}Q_0abx, 1) \xRightarrow{1g} (daaabx, 1).
\end{aligned}$$

Thus, even though the initial string only partially matches the context ab, by switching to state 2 and after that back to state 1, the missing a is inserted and the context is successfully validated. Remark that the state switch should be done on the insertion of x by rule $1d$, because otherwise several occurrences of x can be introduced into the string. In a more general manner, because states are also in play, it may not be possible to reorder the derivation in such a way that the string fully corresponds to the contexts of the simulated rule r at one given moment, which means that a "simulation" of r may be successfully completed even in the situations in which r itself could never be applied.

The above issue does not occur when simulating regular contexts with systems of size $(1, 2, 0; 1, 1, 0)$, because the insertion or deletion of x is done after all of the state symbols checking the context have been inserted, from left to right (cf. proof of the Theorem 2). However, such systems have another problem – symbol Q_f may not necessarily be deleted immediately; then the insertion or deletion of x can happen twice, even if the left context was changed to not match the rule anymore.

Yet, simulation of regular contexts by rules of size $(1, 2, 0; 1, 1, 0)$ is still possible for the construction from Theorem 1, because the situation we have just described cannot happen. Indeed, when the symbol E is erased by a simulation of r_{14}, for example, the system has already assured the correct form of the string to the left of it. Since moving into state 2 is only possible by r_{15} at this time, we are also sure that the string does change after the symbol-by-symbol checking of the context of r_{14} verifies that S is no longer present.

A slightly more complex analysis is needed for the rules of group (3). When the verification of the context of r_{34} is finished, we know that the string contained $B\mathcal{A}^*R_iZ$ some steps ago, but r_{32} and r_{36} might have erased R_i and Z in the meantime, so r_{38} could be applied thereby allowing one more deletion of two symbols at the end of the string. Note, however, that Z' would not be erased, so the next signal symbol R_j would not be able to trigger an insertion of P_j and the system would block. A similar argument is valid for r_{35}: the Z to the left of P_i should stay in the string in order for all of the symbols of α_i to be inserted, or else Z' will not be deleted. In the case of r_{37}, again, if P_i is deleted before Z', the system blocks.

Finally, when the context of rule r_{42} is completely matched, the only modification that may happen to the string before F is erased is the deletion of O by r_{43}, but this behaviour does not break the simulation of the tag system. Hence, we obtain the following statement.

Theorem 4. *For every tag system TS there exists a two-state graph-controlled insertion-deletion system Π of size* $(1, 2, 0; 1, 1, 0)$*, and a recursive coding ϕ such that the following conditions hold:*

– *$\Pi(\phi(w)) = \{TS(w)\}$, if TS halts on w, and*
– *$\Pi(\phi(w)) = \emptyset$, if TS does not halt on w.*

A symmetric statement for the case of graph-controlled insertion-deletion systems of size $(1, 1, 0; 1, 2, 0)$ is also true, but the simulation of the construction from Theorem 1 is less straightforward than for systems of size $(1, 2, 0; 1, 1, 0)$, because, in the case of deletion rules with two-symbol contexts, the simulation of regular rules starts by an insertion of a state symbol at the rightmost end of the substring to be matched. It is therefore possible that the action of a rule is produced before its context is verified, as we have seen above. We will now analyze those rules of the construction from Theorem 1 which are not of the size $(1, 1, 0; 1, 2, 0)$ one by one, and describe how they can be correctly simulated.

To deal with r_{14}, we will simulate such a finite automaton corresponding to the expression $\mathbf{a}(ZZ')^*$ in which the first state is only visited once, in the initial configuration of the automaton. We will then introduce the symbol $Q_0^{(14)}$, representing this state, into the axiom, before S, giving $\bar{B}B\beta Q_0^{(14)}SEE'F$. The symbol $Q_0^{(14)}$ will be erased by the rule $\big(1, (\mathbf{a}, Q_0^{(14)}, \lambda)_{del}, 1\big)$, for all $\mathbf{a} \in \mathcal{A}$. If $Q_0^{(14)}$ is deleted before all state symbols simulating r_{14} are, some of these symbols will stay stuck in the string, because $Q_0^{(14)}$ cannot be inserted. Therefore, the only way to proceed is to erase $Q_0^{(14)}$ after the simulation of r_{14} is successfully finished.

In the case of rules r_{22} and r_{23}, the additional symbols introduced by the simulation will have to be erased before the system moves into state 1, because otherwise they will not be deleted and will block r_{28}, which requires R_i''' to be immediately to the right of B. Rule r_{28} itself will be replaced by the following four rules:

$$\begin{array}{ll}\big(1,\ (R_i''',\ \ X_i^{(28)}, \lambda)_{ins},\ \{1\}\big), & \big(2,\ (B, R_i''',\ \ \lambda)_{del},\ \{2\}\big),\\ \big(1,\ (X_i^{(28)}, R_i,\ \ \lambda)_{ins},\ \{2\}\big), & \big(2,\ (B, X_i^{(28)}, \lambda)_{del},\ \{1\}\big),\end{array}$$

where $X_i^{(28)}$ is a new symbol.

For the rest of the rules, usual simulation of regular contexts works correctly. Indeed, a simulation of the rule r_{31} happens in the middle portion of the string, which cannot be altered by rules other than r_{32} or another simulation of r_{31}. In the case of r_{34}, P_i may be inserted even though the string does not have the correct form, moving the system into state 2, and initiating another deletion at the left end. In this situation, however, Z must have been deleted for r_{38} to become applicable, so the string has the form $\bar{B}B\mathcal{A}^*Z'(ZZ')^*F$. Since the next signal symbol R_j cannot interact with Z', this means that the sequence $Z'(ZZ')^*$ will never be deleted. A similar argument is true for r_{35}: if the system switches into state 2 before Z' can be erased, it eventually blocks. As to the simulation of r_{37}, if the system switches away from state 1 before the context is fully verified, the next signal symbol will not be able to insert another P_j, because the state symbols verifying the context of r_{37} will block it on its way to the right end of the string.

Finally, suppose that the simulation of r_{42} erases F at an early stage. Remark that, for this simulation to start at all, F has to be preceded by a symbol from $\mathcal{A}$, which means that the system cannot switch into state 2 while the context of r_{42} is being verified. Therefore, if the string still contains other non-terminals than those simulating r_{24} or $\bar{B}$, the system blocks. Our observations imply the truth of the following statement.

Theorem 5. *For every tag system TS, there exists a two-state graph-controlled insertion-deletion system Π of size $(1, 1, 0; 1, 2, 0)$ and a recursive coding ϕ such that the following conditions hold:*

- *$\Pi(\phi(w)) = \{TS(w)\}$, if TS halts on w, and*
- *$\Pi(\phi(w)) = \emptyset$, if TS does not halt on w.*

4 Conclusions

In this paper, we continued the study of leftist insertion-deletion systems introduced in [9], and showed that systems of sizes $(1, 2, 0; 1, 1, 0)$ and $(1, 1, 0; 1, 2, 0)$ equipped with a two-state graph control mechanism can simulate any 2-tag system, and are therefore universal. This contributes to the study of the computational power of leftist systems started in [9].

The proofs shown in the present work are based on an extension to the conventional insertion and deletion rules, whereby specifying the contexts is done by regular expressions instead of fixed words. We proved that two-state graph-controlled leftist insertion-deletion systems with regular contexts can simulate any 2-tag system.

It turned out that, in the case of leftist insertion-deletion systems without control, considering regular contexts does not increase the expressive power: rules of sizes $(1, 2, 0; 1, 1, 0)$ or $(1, 1, 0; 1, 2, 0)$ can simulate the language of any system of size $(1, REG, 0; 1, REG, 0)$. Even though this statement is not generally transposable to the graph-controlled case, the specific construction from Theorem 1 can be simulated by conventional leftist rules, which yielded the main result of this paper: two-state graph-controlled insertion-deletion systems of sizes $(1, 2, 0; 1, 1, 0)$ and $(1, 1, 0; 1, 2, 0)$ can simulate any 2-tag system.

An important question left open is whether insertion-deletion systems of sizes $(1, 2, 0; 1, 1, 0)$ or $(1, 1, 0; 1, 2, 0)$ are universal or even computationally complete. We conjecture that this is not the case, because one-symbol one-sided rules can only assure transmission of information in one direction in the string.

The second important open question, which may serve as an intermediate step to solving the previous one, is whether two-state graph-controlled leftist insertion-deletion systems are computationally complete, i.e. whether they can generate all recursively enumerable languages directly, without any coding. Again, our conjecture is negative, because two states only provide a very limited kind of control, which does not seem sufficient for simulating an arbitrary grammar or a Turing machine.

Acknowledgments. The authors would like to acknowledge the support of ANR project SynBioTIC.

References

1. Benne, R.: RNA Editing: The Alteration of Protein Coding Sequences of RNA. Ellis Horwood, Chichester, West Sussex (1993)
2. Biegler, F., Burrell, M.J., Daley, M.: Regulated RNA rewriting: modelling RNA editing with guided insertion. Theoret. Comput. Sci. **387**(2), 103–112 (2007)
3. Cocke, J., Minsky, M.: Universality of tag systems with P = 2. J. ACM **11**(1), 15–20 (1964)
4. Daley, M., Kari, L., Gloor, G., Siromoney, R.: Circular contextual insertions/ deletions with applications to biomolecular computation. In: SPIRE/CRIWG, pp. 47–54 (1999)
5. Freund, R., Kogler, M., Rogozhin, Y., Verlan, S.: Graph-controlled insertion-deletion systems. In: McQuillan, I., Pighizzini, G. (eds.) Proceedings of the Twelfth Annual Workshop on Descriptional Complexity of Formal Systems, vol. 31 of EPTCS, pp. 88–98 (2010)
6. Galiukschov, B.: Semicontextual grammars. Matematicheskaya Logica i Matematicheskaya Lingvistika, pp. 38–50. Tallin University, Russian (1981)
7. Haussler, D.: Insertion and Iterated Insertion as Operations on Formal Languages. PhD thesis, University of Colorado at Boulder (1982)
8. Haussler, D.: Insertion languages. Inf. Sci. **31**(1), 77–89 (1983)
9. Ivanov, S., Verlan, S.: On the lower bounds for leftist insertion-deletion languages. Submitted
10. Ivanov, S., Verlan, S.: Random context and semi-conditional insertion-deletion systems. CoRR, abs/1112.5947 (2011)
11. Ivanov, S., Verlan, S.: About one-sided one-symbol insertion-deletion P systems. In: Alhazov, A., Cojocaru, S., Gheorghe, M., Rogozhin, Y., Rozenberg, G., Salomaa, A. (eds.) CMC 2013. LNCS, vol. 8340, pp. 225–237. Springer, Heidelberg (2014)
12. Kari, L.: On insertion and deletion in formal languages. PhD thesis, University of Turku (1991)
13. Kari, L., Păun, G., Thierrin, G., Yu, S.: At the crossroads of DNA computing and formal languages: characterizing RE using insertion-deletion systems. In: Proceedings of 3rd DIMACS Workshop on DNA Based Computing, pp. 318–333. Philadelphia (1997)
14. Kari, L., Thierrin, G.: Contextual insertions/deletions and computability. Inf. Comput. **131**(1), 47–61 (1996)
15. Kleene, S.C.: Representation of events in nerve nets and finite automata. In: Shannon, C., McCarthy, J. (eds.) Automata Studies, pp. 3–41. Princeton University Press, Princeton, NJ (1956)
16. Krassovitskiy, A.: Complexity and Modeling Power of Insertion-Deletion Systems. PhD thesis, Departament de Filologies Romániques, Universitat Rovira and Virgili (2011)
17. Krassovitskiy, A., Rogozhin, Y., Verlan, S.: Further results on insertion-deletion systems with one-sided contexts. In: Martín-Vide, C., Otto, F., Fernau, H. (eds.) LATA 2008. LNCS, vol. 5196, pp. 333–344. Springer, Heidelberg (2008)
18. Krassovitskiy, A., Rogozhin, Y., Verlan, S.: Computational power of insertion deletion (P) systems with rules of size two. Nat. Comput. **10**(2), 835–852 (2011)

19. Marcus, S.: Contextual grammars. Revue Roumaine de Mathématiques Pures et Appliquées **14**, 1525–1534 (1969)
20. Margenstern, M., Păun, G., Rogozhin, Y., Verlan, S.: Context-free insertion-deletion systems. Theoret. Comput. Sci. **330**(2), 339–348 (2005)
21. Matveevici, A., Rogozhin, Y., Verlan, S.: Insertion-deletion systems with one-sided contexts. In: Durand-Lose, J., Margenstern, M. (eds.) MCU 2007. LNCS, vol. 4664, pp. 205–217. Springer, Heidelberg (2007)
22. Minsky, M.: Computations: Finite and Infinite Machines. Prentice Hall, Englewood Cliffts, NJ (1967)
23. Petre, I., Verlan, S.: Matrix insertion-deletion systems. Theoret. Comput. Sci. **456**, 80–88 (2012)
24. Păun, G.: Marcus Contextual Grammars. Kluwer Academic Publishers, Norwell, MA, USA (1997)
25. Păun, G., My, N.X.: On the inner contextual grammars. Revue Roumaine de Mathématiques Pures et Appliquées **25**, 641–651 (1980)
26. Păun, G., Rozenberg, G., Salomaa, A.: DNA Computing: New Computing Paradigms. Springer, Heidelberg (1998)
27. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages. Springer-Verlag, Berlin (1997)
28. Takahara, A., Yokomori, T.: On the computational power of insertion-deletion systems. In: Hagiya, M., Ohuchi, A. (eds.) DNA8 Sapporo. LNCS, vol. 2568, pp. 269–280. Springer, Heidelberg (2002)
29. Verlan, S.: On minimal context-free insertion-deletion systems. J. Automata, Languages Comb. **12**(1–2), 317–328 (2007)
30. Verlan, S.: Study of language-theoretic computational paradigms inspired by biology. Habilitation thesis, Université Paris Est (2010)