Regulated insertion-deletion systems

Artiom Alhazov, Rudolf Freund, Sergiu Ivanov, Sergey Verlan

For submission to the Journal of Automata, Languages and Combinatorics
Created on August 21, 2021

REGULATED INSERTION-DELETION SYSTEMS

Artiom Alhazov[(A)] Rudolf Freund[(B,E)] Sergiu Ivanov[(C)]
Sergey Verlan[(D)]

[(A)] *Vladimir Andrunachievici Institute of Mathematics and Computer Science*
Acdemiei 5, Chişinău, MD-2028, Moldova
artiom@math.md

[(B)] *Faculty of Informatics, TU Wien*
Favoritenstraße 9–11, 1040 Wien, Austria
rudi@emcc.at

[(C)] *IBISC, Université Évry, Paris-Saclay*
23, boulevard de France 91034 Évry, France
sergiu.ivanov@ibisc.univ-evry.fr

[(D)] *Univ. Paris Est Creteil, LACL, 94010, Creteil, France*
verlan@u-pec.fr

ABSTRACT

The systems using insertion and deletion systems are used in several areas of theoretical computer science, ranging from linguistics to DNA computing. In this paper we investigate insertion and deletion systems within the regulated rewriting framework. We consider various regulation mechanisms already studied in the area of insertion-deletion systems and recall several results, but also elaborate some new results and also consider new control mechanisms.

Keywords: insertion-deletion, regulated rewriting, formal languages

1. Introduction

2. Definitions

In this section we recall some basic notions and definitions used in formal language theory. For further notions and results in formal language theory we refer to textbooks like [5] and [11].

The set of non-negative integers by $\mathbb{N}$. An *alphabet* V is a finite non-empty set of abstract *symbols*. Given V, the free monoid generated by V under the operation of concatenation is denoted by V^*; the elements of V^* are called strings, and the *empty string* is denoted by λ; $V^* \setminus \{\lambda\}$ is denoted by V^+. The cardinality of a set M is denoted by $|M|$.

[(E)] Corresponding author

2.1. Grammars and Normal Forms

Definition 1. A *string grammar* is a construct $G = (N, T, P, S)$ where N is the alphabet of nonterminal symbols, T is the alphabet of terminal symbols, P is a set of productions (or *rules*) of the form $\alpha \to \beta$ with $\alpha, \beta \in (N \cup T)^*$, and $S \in N$ is the start symbol.

If all rules in P are of the form $A \to w$ with with $A \in N$ and $w \in (N \cup T)^*$, then G is called *context-free*.

If all rules in P are of the forms $A \to uBv$ with $A, B \in N$ and $u, v \in T^*$ or $A \to u$ with $A \in N$ and $u \in T^*$, then G is called *linear*.

If all rules in P are of the forms $A \to uB$ with $A, B \in N$ and $u \in T^*$ or $A \to \lambda$ with $A \in N$, then G is called *regular*.

A string v is derivable from a string u, $u, v \in (N \cup T)^*$, if and only if $u = x\alpha y$ and $v = x\beta y$ for some $x, y \in (N \cup T)^*$ and there exists a production $\alpha \to \beta$ in P; we write $u \Longrightarrow_G v$. The reflexive and transitive closure of the derivation relation $\Longrightarrow_G$ is denoted by $\Longrightarrow_G^*$. The string language generated by G is denoted by $L(G)$ and defined as the set of terminal strings derivable from the start symbol, i.e., $L(G) = \{w \mid w \in T^* \text{ and } S \Longrightarrow_G^* w\}$.

The family of regular, linear, context-free, and recursively enumerable string languages is denoted by REG, LIN, CF, and RE, respectively. Two languages of strings L and L' are considered to be equal if and only if $L \setminus \{\lambda\} = L' \setminus \{\lambda\}$.

2.1.1. Penttonen normal form

For every recursively enumerable language $L \subseteq T^*$ there exists a string grammar in Penttonen normal form (N, T, P, S) where the productions in P are of the following forms:

$$XY \to XZ, X, Y, Z \in N,$$
$$X \to YZ, X, Y, Z \in N,$$
$$X \to a, X \in N, a \in T \cup \{\lambda\}.$$

2.1.2. Geffert normal form

As elaborated in [7], for every recursively enumerable language $L \subseteq T^*$ there exists a grammar in Geffert normal form $(\{S\} \cup N_T, T, P, S)$ where the productions in P are of the following forms:

$$S \to uSv, u, v \in (N_T \cup T)^*, |uv| > 0,$$
$$S \to \lambda$$

together with one of the following variants of cooperative erasing rule:

(I) $AA \to \lambda$ and $BBB \to \lambda$, in which case $N_T = \{A, B\}$,
(II) $ABBBA \to \lambda$, in which case $N_T = \{A, B\}$,
(III) $ABC \to \lambda$, in which case $N_T = \{A, B, C\}$,

(IV) $AB \to \lambda$ and $CC \to \lambda$, in which case $N_T = \{A, B, C\}$, or
(V) $AB \to \lambda$ and $CD \to \lambda$, in which case $N_T = \{A, B, C, D\}$.

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 $N_T \cup T$:

Corollary 2. *For each recursively enumerable language $L \subseteq T^*$ there exists a grammar $G' = (\{S\}, \{A, B, C\} \cup T, P', S)$ such that $L(G') \cap T^* = L$ and P' contains only linear rules and the special rule $ABC \to \lambda$.*

Proof. (sketch) We only have to take the grammar in Geffert normal form $G = (\{S, A, B, C\}, T, P, S)$ generating L and define $P' = P$ with the only difference between P and P' that in P the symbols A, B, C are non-terminal symbols, whereas in P' these symbols are terminal, which makes the rules containing S linear ones in P'. □

Similar results can easily be established for the other non-cooperative rules listed above for the Geffert normal form.

Based on the proofs given in [7], various special variants for the linear rules over the "terminal" alphabet $N_T \cup T$ have been elaborated, for example, see [6, 8, 10].

Throughout this paper, we will use a *special Geffert normal form* where these "linear" rules $S \to uSv$ from the Geffert normal form are transformed into a set of "left-" and "right-linear" rules using a standard approach, e.g., see [10, 12] for more details. These "left-" and "right-linear" rules are of the forms $X \to bY$ and $X \to Yb$ with $X, Y \in N$ and $b \in N_T$, where N is is the new alphabet of non-terminal symbols including S.

2.2. Insertion-Deletion Systems

Definition 3. Let V be an alphabet. An insertion / deletion rule is a triple $(u, x, v)_{ins}$ / $(u, x, v)_{del}$, where $x \in V^+$ and $u, v \in V^*$.

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 $w \Longrightarrow_r w'$. In a similar way, the application of the rule $r' : (u, x, v)_{ins}$ 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. We also remark that it is possible to simulate such rules with Chomsky grammars by using an additional non-terminal symbol systematically inserted before and after all other symbols in order to mark a possible insertion site, and at the end all remaining copies of this symbols have to be erased.

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 4. 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:

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

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

$$\begin{aligned} n &= \max\{|x| \mid (u, x, v)_{ins} \in R\} & p &= \max\{|x| \mid (u, x, v)_{ins} \in R\} \\ m &= \max\{|u| \mid (u, x, v)_{ins} \in R\} & q &= \max\{|u| \mid (u, x, v)_{ins}, \in R\} \\ m' &= \max\{|v| \mid (u, x, v)_{ins} \in R\} & q' &= \max\{|v| \mid (u, x, v)_{ins} \in R\} \end{aligned}$$

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

In the following, we will also use the notation $I(u, x, v)$ for the insertion rule $(u, x, v)_{ins}$ and the notation $D(u, x, v)$ for the deletion rule $(u, x, v)_{del}$.

3. Insertion-Deletion Systems with Regular Control

In this section we first consider the

3.1. Insertion-Deletion Systems with Regular Control

We now define the general model of insertion-deletion systems with regular control:

Definition 5. An insertion-deletion system with regular control is a construct $\Gamma = (V, T, A, R, L)$ where $\Gamma = (V, T, A, R)$ is an insertion-deletion system and L is a regular language over the set of labels for the labeled insertion-deletion rules in R. A valid derivation in Γ then is a derivation for which the sequence of labels for the applied rules is in L.

3.2. Insertion-Deletion Systems with Contextless Rules

We recall that it is known, [cfinsdelTCS], that insdels 200300 and 300200 can simulate specific computationally complete normal forms of type-0 grammars, where each production is simulated by one or a few insertions followed by one or a few deletions. These insdel operations are called M-related.

We claim that **each** derivation (or a fragment of a derivation) of **any** insdel system without context can be reordered in such a way that all insertions precede all deletions, yielding the same string. Together with a known technique, [cfinsdelTCS], that shows how to generate precisely the language of the simulated grammar, it follows that the same language can be simulated in $(ID)^*$ mode, where I is the set of all insertion operations and D is the set of all deletion operations.

We do it by proving an even more general statement: given a derivation in a contextless insdel, the same string can be derived by delaying a deletion operation for arbitrarily long, provided that the order of deletion operation is preserved.

In the second part of this subsection, we show computational completeness of insertion-deletions systems of size-200300 and size-300200 with associated Szilard languages of the form $(ID)^*$.

In this paper we show that contextless insertion-deletion systems, including the optimal computationally complete ones, are in some sense tolerant with respect to the order of deletions with respect to the order of insertions. We consider two somewhat extreme cases, calling them **eager deletion** (where each insertion is followed by a deletion and each deletion is preceding by insertion) and **lazy deletion** (where all deletions are performed only after all insertions have been done).

3.2.1. Lazy deletion mode

Lemma 6. *Let δ be a (fragment of a) derivation in a contextless insdel system $\gamma = (V, T, A, I, D)$, $\delta = w_1 \Rightarrow_{del} w_2 \Rightarrow_{ins} w_3$. There exists an equivalent (fragment of a) derivation $\delta' = w_1 \Rightarrow_{ins} w_2' \Rightarrow_{del} w_3$ in γ.*

Proof. Then one of two cases must hold: either insertion happens to the left of the deletion, or it happens to the right of the deletion. For our purpose, if it happens in the same position, it can be classified as either. The two cases are symmetric, let us consider the first one. Let the deleted string be $u \in D$ and the inserted string be $v \in I$. Then there exists strings $x_1, x_2, x_3 \in V^*$ such that $w_1 = x_1x_2ux_3$, $w_2 = x_1x_2x_3$ and $w_3 = x_1vx_2x_3$. It is easy to see that there exists an equivalent derivation where the deletion is delayed until after the insertion: $\delta' = x_1x_2ux_3 \Rightarrow_{ins} x_1vx_2ux_3 \Rightarrow_{del} x_1vx_2x_3$.

The second case, where the insertion is performed to the right of the deletion, is similar: $\delta = x_1ux_2x_3 \Rightarrow_{del} x_1x_2x_3 \Rightarrow_{ins} x_1x_2vx_3$ and $\delta' = x_1ux_2x_3 \Rightarrow_{ins} x_1ux_2vx_3 \Rightarrow_{del} x_1x_2vx_3$. As mentioned before, the same-position case can be seen as either, with $x_2 = \lambda$. □

A deletion can be delayed until after multiple insertions.

Corollary 7. *In a derivation of a contextless insdel, a deletion operation can be postponed until an arbitrary number of insertion operations, provided that the order of deletion operations is unchanged.*

Proof. It suffices to consider any derivation where some deletion precedes insertion, and replace such a fragment δ by an equivalent fragment δ' using previous corollary. If the following operations are also insertions, the same process can be repeated. □

The same (delaying a deletion) can be done for multiple deletion operations.

Corollary 8. *Let δ be a derivation in a contextless insdel, and let $c \in (I \cup D)^*$ be its associated control sequence. Let $c_I = (i_1, \cdots, i_m)$ be the sequence of insertion operations from c and $c_D = (d_1, \cdots, d_n)$ be the sequence of deletion operations from*

c. Then for any $c' \in c_I \sqcup\!\sqcup c_D$ there exists an equivalent derivation δ' with control sequence c' if the following condition holds: i_j precedes d_k in c implies i_j precedes d_k in c'.

Proof. δ can be reordered into an equivalent δ' one deletion operation at a time using previous corollary, from right instances of deletion to left. □

Deletions can be delayed until the end of the derivation.

Corollary 9. *Let δ be a derivation in a contextless insdel. There is an equivalent derivation δ' where its control sequence is in I^*D^*.*

Proof. Let $c \in (I \cup D)^*$ be the control sequence of δ. Let $c_I = (i_1, \cdots, i_m)$ be the sequence of insertion operations from c and $c_D = (d_1, \cdots, d_n)$ be the sequence of deletion operations from c. Then, by previous corollary, a derivation δ' exists with the control sequence $c_I c_D$, because $c_I c_D \in c_I \sqcup\!\sqcup c_D$ and all insertions precede all deletions in δ'. □

Therefore, contextless insdels are universal also in in lazy deletion mode.

Theorem 10. $L(INS_2^{0,0}DEL_3^{0,0}, I^*D^*) = (INS_3^{0,0}DEL_2^{0,0}, I^*D^*) = RE$.

Proof. For any $L \in RE$ there exist insdels γ of size 200300 generating L and γ' of size 300200 generating L, [cfinsdelTCS]. Since lazy deletion mode yields a subset of derivations, it follows that in lazy deletion mode, the languages generated by γ and γ' are subsets of L. Conversely, by the previous corollary, each string $w \in L$ can also be generated by the same insel systems also in lazy deletion mode, so $L(\gamma, I^*D^*) = L(\gamma', I^*D^*) = L$. This proves the claim. □

We would like to note how a subset of a variant of a copy language may originate as a control sequence of such operations in case of size 300300.

Remark 11. Consider insdels of size 300300. In [cfinsdelTCS] it is shown that they generate RE in unrestricted mode, while the proof makes it obvious that $(ID)^*$ mode is enough, since each production of the underlying type-0 grammar is simulated by an insertion followed by a deletion. This pair of operations are called M-related, and this relation is a bijection since the label of the underlying grammar production is used in both, and only in them. In this context, for any insertion operation o, let us denote its m-related deletion operation by o'. The control sequence in $(ID)*$ mode is a concatenation of oo' pairs. Let w be the sequence of insertion operations in a derivation, and let w' be the sequence of deletion operations in the same derivation. We note that $|w| = |w'|$ and n-th symbol of w is an operation which is M-related to the operation which is $n-th$ symbol of w'. By the previous corollary, these operations can be reordered, and the controlled sequence of such an equivalent reordering is ww'.

3.2.2. Eager deletion mode

The computational completeness of insels of size 300300 in $(ID)^*$ mode follows directly from the proof of their computational completeness in the unrestricted mode. As we

show below, for sizes 200300 and 300200 this result also holds, but needs more arguing. The idea is that we take an arbitrary derivation in an unrestricted mode and shuffle in some dummy-operations, which either do not affect the derivation or are harmless (not leading to the terminal string).

Lemma 12. *Let $\delta = A \Rightarrow^* w$ be a derivation in a contextless insdel $\gamma = (V, T, A, I, D)$, and let $X \notin V$ be a new non-terminal system. Consider a new insdel $\gamma'(n) = (V', T', A', I', D')$ where $V \cup \{X\}$, $T' = T \cup \{X\}$, $A' = X^n A$, $I' = I \cup \{ins(X)\}$, $D' = D \cup \{del(X)\}$. Then there exists $n \in \mathbb{N}$ and a derivation $\delta' = X^n A \Rightarrow^* w'$ in eager deletion mode where $w' \in X^* w$.*

Proof. Notice that δ is also a derivation in $\gamma'(0)$. Find all places where deletion is preceded by another deletion (or is the first step), and perform $ins(X)$ in the beginning of the string every time it happens (so every del is preceded by an ins). Now let us assume we start with a sufficient number of symbols X, e.g., it suffices to take n to be the number of times an insertion is followed by another insertion in δ (or is the last step). Then, perform $del(X)$ every time it happens (so every ins is followed by a del; this preserves the property that every del is preceded by an ins). Denote the resulting derivation by δ'. Clearly, δ' yields a string in $X^* w$, because it only differs from δ by additional symbols X, and it is a derivation in eager deletion mode. □

We claim that with this technique generation of every terminal string can be done in eager deletion mode, and it is also impossible to over-generate.

Corollary 13. *In the previous lemma, for the constructed insdel $\gamma' = \gamma'(n)$, the projection of $L(\gamma')$ on V equals $L(\gamma)$.*

Proof. One inclusion follows by the lemma, since a derivation has been constructed yielding a string in $X^* w$. It remains to argue that out of $(T \cup \{X\})^*$, only strings in $L(\gamma) \sqcup\!\sqcup X^*$ can be derived.

Indeed, from any derivation in γ' producing a string in $(T \cup \{X\})^*$ we can extract a derivation in γ by removing operations $ins(X)$ and $del(X)$ and initial symbols X, and clearly γ is a valid derivation yielding some string $w \in T^*$. Therefore, the string produced in γ' is in $L(\gamma) \sqcup\!\sqcup X^*$. This proves the corollary. □

Corollary 14. *For any contextless insdel of size at least 200200, another insdel of the same size can be constructed generating the same language in $(ID)^*$ mode.*

Proof. Let $\gamma = (V, T, A, I, D)$ be such an insdel system. Consider $\gamma'(n) = (V', T', A', I', D')$ from the previous lemma. It remains to transform it into an insdel that generates the same language modulo symbols X, but does not start with any symbols X and does not yield any symbols X. Let γ'' be a new insdel system (V', T, A, I'', D'') where $I'' = I' \cup \{ins(XX)\}$ and $D'' = D' \cup \{del(XX)\}$. Notice that X is now a non-terminal and the axiom is the same as in γ. We claim that $L(\gamma'') = L(\gamma'', (ID)^*) = L(\gamma)$.

First, we argue that each string from $L(\gamma)$ is generated by γ'' in eager deletion mode. Notice that any derivation δ' in γ' is also a (fragment of) a derivation in γ''. The necessary number n of starting symbols X can be produced by $(ins(XX), del(X))^n$ applied in the beginning of the string, and any symbols X remaining in the start of the string yielded by γ' can be removed by some operation sequence in $(ins(X), del(XX))^*$. Notice that such derivation is in eager deletion mode.

It remains to argue that no other string can be produced except those in $L(\gamma)$. Notice that all symbols $X \notin T$ must be deleted in order to reach a terminal string, and the process of inserting and removing them cannot alter the projection of the sentential form onto T, concluding the converse part of the proof, so the same string can always be produced (the "garbage" symbols X can always be cleaned up by suitable insertions or deletions of X and XX).

Therefore, $L(\gamma'', (ID)^*) = L(\gamma)$, which proves the corollary. □

We now conclude that insdels of size 200300 and 300200 are computationally complete also in eager deletion mode.

Theorem 15. $L(INS_2^{0,0}DEL_3^{0,0}, (ID)^*) = (INS_3^{0,0}DEL_2^{0,0}, (ID)^*) = RE.$

3.3. Time-Varying Insertion-Deletion Systems

Time-varying insertion-deletion systems are a special case of insertion-deletion systems with regular control where the regular control language has the special form $(R_1 \dots R_n)^*$ with the R_i, $1 \leq i \leq n$, being finite sets of rules:

Definition 16. A time-varying insertion-deletion system is insertion-deletion system with regular control $\Gamma = (V, T, A, R, L)$ where $L = (R_1 \dots R_n)^*$ and $R_i \subseteq R$, $1 \leq i \leq n$; n is called the *period*.

In a time-varying insertion-deletion system $\Gamma = (V, T, A, R, L)$ the set of avaiable rules changes periodically with period n, i.e., in step $mn+k$, $m \geq 0$, of any derivation exactly the rules in R_k may be used.

We now show that any grammar in special Geffert normal form can be simulated by a time-varying insertion-deletion system with period 5.

Theorem 17. *For any RE language L we can construct a time-varying insertion-deletion system Γ of size $(1, 1, 0; 1, 1, 0)$ and period 5 such that $L(\Gamma) = L$.*

Proof.

Consider an arbitrary grammar $G = (N, T, S, P)$ in special Geffert normal form generating L.

We then construct a time-varying insertion-deletion system

$$\Gamma = (V, T, A, R = (R_1 \cup R_2 \cup R_3 \cup R_4 \cup R_5, L = (R_1R_2R_3R_4R_5)^*)$$

with period 5 as follows:

For any rule $r : X \to bY \in P$ we add following rules to Γ:

$R1 : r.1 : (\lambda, r, \lambda)_{ins}$
$R2 : r.2 : (r, X, \lambda)_{del}$
$R3 : r.3 : (r, Y, \lambda)_{ins}$
$R4 : r.4 : (r, b, \lambda)_{ins}$
$R5 : r.5 : (\lambda, r, \lambda)_{del}$

For any rule $r : X \to bY \in P$ we add following rules to Γ:

$R1 : r.1 : (\lambda, r, \lambda)_{ins}$
$R2 : r.2 : (r, X, \lambda)_{del}$
$R3 : r.3 : (r, b, \lambda)_{ins}$
$R4 : r.4 : (r, Y, \lambda)_{ins}$
$R5 : r.5 : (\lambda, r, \lambda)_{del}$

For rule $r : S \to \lambda \in P$ we add following rules to Γ:

$R1 : r.1 : (\lambda, r, \lambda)_{ins}$
$R2 : r.2 : (r, S, \lambda)_{del}$
$R3 : r.3 : (r, \#, \lambda)_{ins}$
$R4 : r.4 : (r, \#, \lambda)_{del}$
$R5 : r.5 : (\lambda, r, \lambda)_{del}$

For the rule $r : ABC \to \lambda$ we add following rules to Γ:

$R1 : r.1 : (\lambda, r, \lambda)_{ins}$
$R2 : r.2 : (r, A, \lambda)_{del}$
$R3 : r.3 : (r, B, \lambda)_{ins}$
$R4 : r.4 : (r, C, \lambda)_{del}$
$R5 : r.5 : (\lambda, r, \lambda)_{del}$

Note that this cycle ensures A, B, and C to be deleted in this particular order. □

Corollary 18. *For any RE language L we can construct a time-varying insertion-deletion system Γ of size $(1, 0, 1; 1, 0, 1)$ and period 5 such that $L(\Gamma) = L$.*

Proof. Obvious. □

4. Matrix, Programmed and Graph-Controlled Insertion-Deletion Systems

4.1. $(1,1,0;1,1,0)$ + *programmed control = RE*

The construction for the time-varying insertion-deletion system elaborated in the proof of Theorem 17 has to be modified in order to serve as the basis for the construction of the corresponding programmed insertion-deletion system, as in contrast to the graph-controlled version all nodes in the control graph can be initial and final nodes as well.

Let

$$\Gamma = (V, T, A, R = (R_1 \cup R_2 \cup R_3 \cup R_4 \cup R_5, L = (R_1 R_2 R_3 R_4 R_5)^*)$$

be the time-varying system from Theorem 17. Then the system modified for the programmed control has the form $\Gamma' = (V', T, A, Rr_I S_I S_E S'_E, R'_2, R'_3, R'_4, R'_5, R'_1)$.

The new alphabet contains some additional symbols which will force Γ' to always start and end the simulation in the correct phases $V' = V \cup \{S_I, S_E, S'_E, r_I, r_E\}$.

The sets of rules are shifted circularly by 1 to the left with respect to Γ in order to reduce the number of modifications to get the programmed control.

The new symbol r_I corresponds to the deletion of the symbol S_I from the axiom and is simulated by the following sequence of rules:

$$R'_2 : r_I.2 : (r_I, S_I, \lambda)_{del}$$
$$R'_3 : r_I.3 : (r_I, \#, \lambda)_{ins}$$
$$R'_4 : r_I.4 : (r_I, \#, \lambda)_{del}$$
$$R'_5 : r_I.5 : (\lambda, r_I, \lambda)_{del}$$
$$R'_1 : r_I.1 : (\lambda, r, \lambda)_{del}$$

where r corresponds to any other rule of the simulated grammar, and $r \neq r_I$. Thus, Γ' must start by carrying out the sequence of rules $r_I.i$ and thus must start with $r_I.2$ in R'_2, since no other simulating rule in any component is applicable to the axiom.

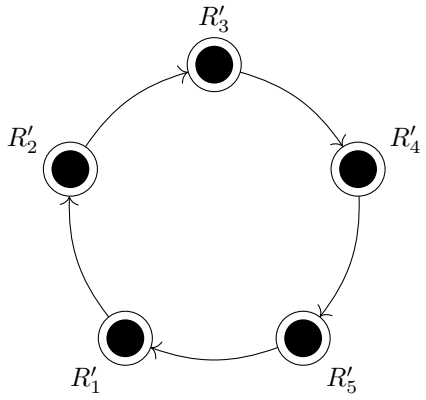
To ensure that Γ' halts correctly when no more simulations are possible, after having completed a whole number of simulations, its rules include a clean-up phase which erases S_E and S'_E. This phase is started by the insertion of r_E instead of a normal rule marker. The action associated with r_E are the following:

$$R'_2 : r_E.2 : (r_E, S_E, \lambda)_{del}$$
$$R'_3 : r_E.3 : (r_E, \#, \lambda)_{ins}$$
$$R'_4 : r_E.4 : (r_E, \#, \lambda)_{del}$$
$$R'_5 : r_E.5 : (\lambda, r_E, \lambda)_{del}$$
$$R'_1 : r_E.1 : (r_E, S'_E, \lambda)_{del}$$

Note that this sequence of actions does not insert any other rule symbol, rendering the rules in the corresponding components inapplicable and thus stopping the derivation.

Compared to Γ, Γ' includes several new symbols and the associated rules meant to ensure that starting the simulation in any other component than R'_2 will never lead to a terminal string. In particular, if the simulation starts in R'_1 by inserting a new rule marker r, then there will always be an extra rule marker (either r_I or an r) in the string which will never be deleted.

The particularity of this construction is that Γ' uses a programmed graph control with a circular structure:



5. Insertion-Deletion Systems with Activation of Rules

Activation at least two steps forward $A(2)$ is sufficient to simulate time variance. Activation $A(k)$ subsumes time-variance with period k: $A(k) \supseteq TV(k)$. Note that $A(1)$ is strictly less powerful than $A(2)$, so $A(1)$ will probably not be enough to simulate $TV(k)$.

6. CD Grammar Systems

Definition 19. A CD grammar system is

The next example shows that using CD control the computational power of insertion-deletion systems is strictly increased (we recall that in [9] it was shown that insertion-deletion systems of size $(1, 1, 1; 1, 1, 0)$ cannot generate the language $a^n b^n$).

Example 20. Consider the following CD insertion-deletion system Γ of size $(1, 1, 1; 1, 0, 0)$ with 3 components working in t-mode.

$V = \{a, b, X, Y\}$, $T = \{a, b\}$, $A = \{ab\}$.

$R_1 = \{(a, X, b)_{ins}, (X, Y, b)_{ins}\}$.

$R_2 = \{(X, a, Y)_{ins}, (a, b, Y)_{ins}\}$.

$R_3 = \{(\lambda, X, \lambda)_{del}, (\lambda, Y, \lambda)_{del}\}$.

For the initial string ab no rule in components 2 and 3 is applicable. If component 1 is applied, then string $aXYb$ is obtained. Next, only rules of components 2 or 3 are

applicable. By using rules from component 3, the string ab is obtained again. If a derivation using component 2 is applied, then string $aXabYb$ is obtained. Now, only rules of components 1 and 3 are applicable. By using those from component 3 the string $aabb$ is obtained. If using those from component 1, the string $aXaXYbYb$ is obtained.

By repeating the procedure above strings a^3b^3 and $aXaXabYbYb$ are obtained. It is easy to observe that strings of the last type can derive only strings of form a^nb^n, $n > 2$, because the contexts Xa, aX, Yb and bY are never used in rules.

Hence, $L(\Gamma) = \{a^nb^n \mid n > 0\}$.

In a similar manner the non-context-free language $\{ww \mid w \in \{a, b\}^*\}$ can be generated.

Example 21. Consider the following CD insertion-deletion system Γ of size $(1, 1, 1; 1, 0, 0)$ with 3 components working in t-mode.

$V = \{a, b, X, Y, B, E, M\}$, $T = \{a, b\}$, $A = \{BME\}$.

$R_1 = \{(B, X, u)_{ins}, (M, Y, v)_{ins} \mid u \in V \setminus \{X\}, v \in V \setminus \{Y\}\}$.

$R_2 = \{(B, a, X)_{ins}, (M, a, Y)_{ins}\}$.

$R_3 = \{(B, b, X)_{ins}, (M, b, Y)_{ins}\}$.

$R_4 = \{(\lambda, X, \lambda)_{del}, (\lambda, Y, \lambda)_{del}\}$.

$R_5 = \{(\lambda, B, \lambda)_{del}, (\lambda, E, \lambda)_{del}, (\lambda, M, \lambda)_{del}\}$.

Like in the previous example, component 1 allows an insertion of a single copy of X after the start marker B and of a single copy of Y after the middle marker M. Then rules in components 2 or 3 are applied extending the string by the same single letter after B and M. This works because there is a single context BX and MY. Component 4 allows to erase symbols X and Y and the process can be repeated again. At some moment, rules from component 5 can be applied, removing the markers B, E and M in the string, and then stopping the derivation.

Hence, $L(\Gamma) = \{ww \mid w \in \{a, b\}^*\}$.

CD insertion-deletion systems of size $(1, 1, 1; 1, 1, 0)$ can generate all linear languages.

Theorem 22. *For any linear language L there exists $n > 0$ and a CD insertion-deletion system $\Gamma = (V, T, A, R_1, \ldots, R_{n+2})$ such that $L = L(\Gamma)$.*

Proof. Consider an arbitrary linear grammar $G = (V', T', S, P)$ such that $L(G) = L$. We may suppose that rules in P are left- and right-linear (of form $X \to bY$, $X \to Yb$ and $X \to \lambda$). We also suppose that rules from G are labelled by natural numbers from 1 to $|P|$.

The components of Γ are defined as follows.

$T = T'$.

$V = T \cup V' \cup \{[i], [i'] \mid 1 \leq i \leq n\}$

$R_i = \{([i], Y, X)_{ins}, ([i], b, Y)_{ins}, (Y, X, \lambda)_{del} \mid i : X \to bY\}$.

$R_i = \{([i], Y, [i'])_{ins}, (Y, b, [i'])_{ins}, ([i'], X, \lambda)_{del} \mid i : X \to Yb\}$.

$R_i = \{(\lambda, X, \lambda)_{del} \mid i : X \to \lambda\}$.

$R_{n+1} = \{(v, [i], X)_{ins} \mid v \in V \setminus \{[j] \cup [j']|1 \leq j \leq n\}, i : X \to u, u \in V^2\} \cup \{([i], [i'], X)_{ins} \mid i : X \to Yb\}$.

$R_{n+2} = \{(\lambda, [i], \lambda)_{del}, (\lambda, [i'], \lambda)_{del} \mid 1 \leq i \leq n\}$.

The functioning of this system is similar to the examples above. Component R_{n+1} introduces a symbol $[i]$ before the nonterminal in the string. Next, only rules from component i (and $n+2$) are applicable. In component i the sequence bY (or Yb) is obtained only in the right order (because of the insertion contexts). Also, in the same component corresponding symbol X is deleted.

Finally, component R_{n+2} cleans up symbols $[i]and[i']$ yielding a terminal string. □

6.1. CD grammar systems with setmax and t-mode

6.1.1. Generating the copy language ww

To generate the copy language $L_{ww} = \{ww \mid w \in T^*\}$ with CD grammar systems with the set condition and in t-mode, we construct a CD system containing components of the following form:

(I) one component $[I(L, a, \lambda), I(R, a, \lambda)]$ for every $a \in T$,

(II) one clean-up component $[D(\lambda, L, \lambda), D(\lambda, R, \lambda)]$.

All components work in the mode setmax (set condition + t-mode), or set condition + = 2, or set condition + ≥ 2. The axiom / starting string of this CD grammar system is LR.

This system works by non-deterministically choosing a terminal symbol and inserting it both to the right of L and to the right of R. At most one insertion may happen to the right of L, and at least one to the right of R, because of the set condition. Both insertions must happen because of the t-mode / mode ≥ 2 / mode = 2. This process goes on until the CD system chooses component (2) which erases the makers L and R and thereby ends the derivation at a terminal string in L_{ww}.

7. Conclusions

We have shown that contextless insertion-deletion systems, including those of optimal size, i.e., 200300 and 300200, are computationally complete not only in the case of not restricting the Szilard language associated with the derivations, but also in extreme variants for the form of the associated Szilard language , such that alternating insertions and deletions, or all insertions preceding all deletions.

References

[1] A. Alhazov, R. Freund, S. Ivanov, M. Oswald, Relations between control mechanisms for sequential grammars. *Fundamenta Informaticae* **181** (2021) 2–3, 239—271.

[2] A. Alhazov, A. Krassovitskiy, Yu. Rogozhin, S. Verlan, P systems with minimal insertion and deletion. *Theoretical Computer Science* **412** (2011) 1-2, 136–144.

[3] F. Biegler, M. J. Burrell, M. Daley, Regulated RNA rewriting: Modelling RNA editing with guided insertion. In: H. Leung, G. Pighizzini (eds.), *8th International Workshop on Descriptional Complexity of Formal Systems – DCFS 2006, Las Cruces, New Mexico, USA, June 21–23, 2006. Proceedings*. New Mexico State University, Las Cruces, New Mexico, USA, 2006, 70–81.

[4] F. Biegler, M. J. Burrell, M. Daley, Regulated RNA rewriting: Modelling RNA editing with guided insertion. *Theor. Comput. Sci.* **387** (2007) 2, 103–112.

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

[6] R. Freund, M. Kogler, Yu. Rogozhin, S. Verlan, Graph-controlled insertion-deletion systems. In: I. McQuillan, G. Pighizzini (eds.), *Proceedings Twelfth Annual Workshop on Descriptional Complexity of Formal Systems, DCFS*. EPTCS 31, 2010, 88–98.

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

[8] S. Ivanov, S. Verlan, Universality of graph-controlled leftist insertion-deletion systems with two states. In: J. Durand-Lose, B. Nagy (eds.), *Machines, Computations, and Universality – 7th International Conference, MCU*. LNCS 9288, Springer, 2015, 79–93.

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

[10] I. Petre, S. Verlan, Matrix insertion-deletion systems. *Theoretical Computer Science* **456** (2012), 80–88.

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

[12] S. Verlan, H. Fernau, L. Kuppusamy, Universal insertion grammars of size two. *Theor. Comput. Sci.* **843** (2020), 153–163.

8. Ideas for the paper

Recall results on existing regulated insdel variants and propose new ones. Give some small proofs for several new cases (not all). Give definitions in particular (not generalized form!)

Proposed contents:

- Discussion on insdel and non-complete insdels.
- General discussion on regulated rewriting and also regulated RNA editing.
- Recall of results for matrix control. Discuss the ac case as well.
- Recall of results for graph control (also discuss 2 variants).
- Note for programmed grammars. Also discuss the ac case.
- Define prescribed sequences variant. Some proofs?
- Discuss Time-varying variant. Proofs.
- Recall semi-contextual and random-context variants.
- Discuss permitting, forbidding and ordered variants (also recal ordered graph-control variant from our paper). Proofs.
- Regular and non-regular context conditions. Examples.
- Indian parallel variant?
- E(T)0L variant?
- Recall results from NEPs.
- Grammar systems CD-style variant. Proofs.
- Other variants? (scattered context, indexed, pure, etc)
- Different new rule execution strategies (e.g. each insertion step followed by a deletion step, or first perform only insertions and then only deletions)

Cite [5].

9. Definitions

We suppose that the reader is familiar with standard notions and notations from formal language theory. We refer to [11] for missing details. An alphabet is a non-empty finite set of symbols. The set of all strings over an alphabet V is denoted by V^*. A string x is said to be a substring of a string w if $w = uxv$.

9.1. Special Geffert Normal Form

This is a verbatim copy from a published article. To reduce

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

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

Throughout this paper, we will use the *special Geffert normal form* (SGNF) [6, 8, 10]. This normal form is obtained by transforming "linear" rules from Geffert normal form to a set of "left- and right-linear" rules using a standard approach, see [10] for more details. Additionally, for commodity reasons, the terminals are generated at the left side of the string, which means by the remark above that "linear" rules from Geffert normal form that have to be transformed are of the form $S \to uSv$ where $v = \lambda$ for some rules, while u is never the empty string (†). Hence, for a grammar in SGNF, it is possible to split the set of non-terminals into disjoint sets as follows: $N = N_S \cup N_T \cup N_L \cup N_R$, where $N_T = N_{AC} \cup N_{BD}$, with $N_{AC} = \{A, C\}$, and $N_{BD} = \{B, D\}$, $N_S = \{S, S'\}$, N_L (resp. N_R) is the set of non-terminals appearing in the left-hand-side of "left-linear" (resp. "right-linear") rules except for S. We cite below some of the interesting properties of a grammar G in SGNF:

(I) G has only two (non-context-free) erasing rules $AB \to \lambda$ and $CD \to \lambda$, and several "right-linear" and "left-linear" rules of one of the following forms; notice that due to Condition (†) above, we start with generating the non-empty part with "right-linear" rules and then might even skip the "left-linear" rules.

- $X \to bY$, where $X \in N_R \cup \{S\}, b \in T \cup N_{AC}, Y \in N_R \cup N_L \cup N_S, X \neq Y$,
- $X \to Yb$, where $X \in N_L, b \in N_{BD}, Y \in N_L \cup N_S, X \neq Y$,
- $S' \to \lambda$.

(II) We can assume that rules rewriting a symbol different from S are deterministic: for each $X \neq S$, there is a single rule $X \to ab \in P$. This property can be achieved, as we can choose different nonterminals in $N_R \cup N_L$ for each "linear" rule $S \to \alpha$ that is going to be simulated by "right-linear" and "left-linear" rules.

(III) The computation in G is done in three stages described below.

- Generating terminals at the left end of the string. The sentential form at this stage can be described by the regular expression $T^*(N \setminus N_T)(N_{BD})^*$. We remark that at this stage only one non-terminal from $N \setminus N_T$ is present in the string.

- Generation of symbols from N_{AC}. The sentential form at this stage can be described by the regular expression $T^*(N_{AC})^*(N \setminus N_T)(N_{BD})^*$. As above, at this stage only one non-terminal from $N \setminus N_T$ is present in the string.
- Erasing stage. The sentential form at this stage can be described by the regular expression $T^*(N_{AC})^*(AB|CD)(N_{BD})^*$. We remark that there is only one matching pair AB or CD present in the string.

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

9.2. Insertion-deletion

Add definitions for matrix, graph-control and programmed variants

9.3. Non-complete insertion-deletion

Insertion-deletion of following sizes is shown to be not complete:

$(1, 1, 0; 1, 1, 1)$

$(1, 1, 1; 1, 1, 0)$

$(1, 1, 0; 2, 0, 0)$

$(2, 0, 0; 1, 1, 0)$

$(2, 0, 0; 2, 0, 0)$

$(1, 0, 0; p, 0, 0)$

$(n, 0, 0; 1, 0, 0)$

Hence, we should take rules from this list for the regulation.

10. Results

Similar proofs as for (110110) may be done for 110200 and 200100 and even 100110/110100 (however in this case the proof would be longer).

10.1. Ordered insertion-deletion systems

Theorem 23. *The language $L = a^*b^*$ cannot be generated by any ordered insertion-deletion system of size* $(1, 1, 0; 1, 1, 0)$.

Proof. The proof needs yet to be formalized. Here is the idea.

Consider a long enough string $a^n b^m$ and a symbol a that was inserted by an insertion rule (say $(x, a, \lambda)_{ins}$):

$z \in A, z \Longrightarrow^* uxv \Longrightarrow uxav \Longrightarrow^* w \in T$.

If after insertion of a rule $(x, a, \lambda)_{ins}$ has priority, then the computation never stops and the result is empty. Otherwise, only rule of type $(a, y, \lambda)_{...}$ has the priority.

Repeat the reasoning. At some moment, either we will consume all non-terminals (recall that we cannot delete terminals) and stop, so (x, a) will have the priority again, or we will insert terminals indefinitively, hence never stopping. □

While I'm quite confident about same result for 110200 and 200110, it might be different for 111110...

It also seems that only very simple languages can be generated by ordered variants – as far as a terminal symbol is inserted, it is inserted perpetually...

Also we recall that combining priorities/ordering and graph-control yields the CC with 110100/100110 (see [2]).

11. CD insertion-deletion systems

For CD insertion-deletion systems working in $= k$ and $\geq k$ mode, the proof is the same like in the time-varying case (put the corresponding 5 rules per component and have as many components as rules).

In t-mode the situation is different, as it is possible to show that there is no computational completeness for standard cases (110110, 110200, 200110) using the same method as in the original insdel. The main idea: when generating a^*b^*, if a letter a is inserted, then any number can be inserted (because of the t-mode) and we cannot distinguish them.

The case 111110 is a bit different. More power is obtained with CD control:

One can generate a^nb^n as follows (even with CF deletion only):

axiom: ab

C1: $(a, X, b)_{ins}$, $(X, Y, b)_{ins}$

C2: $(X, a, Y)_{ins}$, $(a, b, Y)_{ins}$

C3: $(\lambda, X, \lambda)_{del}$, $(\lambda, Y, \lambda)_{del}$

Similarly, $a^nb^nc^n$ or xx can be obtained.

One can generate all CF languages:

For a rule $r : X \to bY$:

$C : (\neg r, r, X)_{ins}$

$R_r : (r, Y, X)_{ins}$, $(r, b, Y)_{ins}$, $(Y, X, \lambda)_{del}$

$D : (\lambda, r, \lambda)_{del}$

For a rule $r : X \to Yb$:

$C : (\neg r, r, X)_{ins}$, (r, r', X)

$R_r : (r, Y, r')_{ins}$, $(Y, b, r')_{ins}$, $(r', X, \lambda)_{del}$

$D : (\lambda, r, \lambda)_{del}$, $(\lambda, r', \lambda)_{del}$

Works because there is only one non-terminal in the string.

Unfortunately, I could not find a way to simulate $AB \to \lambda$...