P Systems with Minimal Left and Right Insertion and Deletion

Rudolf Freund[1], Yurii Rogozhin[2], and Sergey Verlan[3]

[1] Faculty of Informatics, Vienna University of Technology
Favoritenstr. 9, 1040 Vienna, Austria
Email: rudi@emcc.at

[2] Institute of Mathematics and Computer Science
Academy of Sciences of Moldova
Str. Academiei 5, Chişinău, MD-2028, Moldova
Email: rogozhin@math.md

[3] LACL, Département Informatique, Université Paris Est
61, av. Général de Gaulle, 94010 Créteil, France
Email: verlan@univ-paris12.fr

Summary. In this article we investigate the operations of insertion and deletion performed at the ends of a string. We show that using these operations in a P systems framework (which corresponds to using specific variants of graph control), computational completeness can even be achieved with the operations of left and right insertion and deletion of only one symbol.

1 Introduction

The operations of left and right insertion and deletion that we consider in this article correspond to the operations of left and right concatenation and quotient with a finite language. While these operations are known for a long time, their joint investigation in a distributed framework originates from the area of natural computing, where they were used in the context of networks of evolutionary processors (NEP) [7]. Such networks are a special type of networks of language processors [6] that feature a set of (rewriting) nodes rewriting languages and after that redistributing some regular subsets between the nodes. In networks of evolutionary processors, the rewriting operations are replaced by three types of operations having a biological motivation: insertion, deletion, and mutation (substitution). The corresponding systems are quite powerful and we refer to [8] for more details. The redistribution of the node contents based on a regular condition is a very powerful operation. Accepting hybrid networks of evolutionary processors (AHNEP) replace this condition by random context conditions, however, the set

of operations is changed and now includes the insertion and deletion operations at the extremities of the strings; we refer to [21, 9] for more details on AHNEP.

The operations of insertion and deletion on the extremities of a string can also be seen as a particular case of a more general variant, where insertion and deletion can be performed anywhere in the string. The insertion operation defined in such a way was first considered in [14, 15] and after that related insertion and deletion operations were investigated in [17, 18]. Another generalization of the insertion and deletion operations that involves the checking of contexts for the insertion and deletion was considered with a linguistic motivation in [13, 20] and with a biological motivation in [4, 5, 18, 26]. Generally, if the length of the contexts and/or of the inserted and deleted strings are big enough, then the insertion-deletion closure of a finite language leads to computational completeness. There are numerous results establishing the descriptional complexity parameters sufficient to achieve this goal, we refer to [31, 30] for an overview of this area.

Some descriptional complexity parameters lead to variants that are not computationally complete. An investigation of insertion and deletion operations combined with regulating mechanisms was done for these cases, more precisely, with the graph-controlled, the matrix, and the random-context controls [11, 28, 16]. As it was shown in these articles, in most of the cases the additional control leads to computational completeness. The graph-controlled regulation is of particular interest, as it can be related to the notion of P systems. Such systems formalize the functioning of a living cell that topologically delimits processing units by membranes, thus leading to a tree (or graph) structure of processing nodes. The elements processed in some node (membrane) then are distributed among the neighbors in the structure. We refer to [24, 25] and to the web page [29] for more details on P systems. In the case of the operations of insertion and deletion acting on strings this directly corresponds to a graph control where the control nodes correspond to the membranes.

The research on context-free insertion and deletion (i.e., without contextual dependency) shows that if the lengths of the inserted and deleted strings are 2 and 3 (or 3 and 2), respectively, then the insertion-deletion closure of finite languages is computationally complete [22]. When one of these parameters is decreased, this result is not true anymore [32]; moreover, even the graph-controlled variant cannot achieve computational completeness [19]. This changes when a graph control with appearance checking is used [1] or in the case of a random context control [16]. In both variants, minimal operations (involving only one symbol) were considered, leading to RE (the family of recursivey enumerable languages) in the case of set-controlled random context conditions and to $PsRE$ (the family of Parikh sets of RE) in the case of graph control with appearance checking.

We note that the operations of left and right insertion and deletion are incomparable with normal insertion and deletion: because of the positional information, the regular language a^+b^+ can be obtained even with left and right insertions of only one symbol, yet not when insertions are possible at arbitrary positions in the string. On the other hand, the Dyck language cannot be obtained when insertion

is only possible at the ends of the strings, while with normal insertion this can be done easily. In [1, 3], left and right insertion and deletion operations (under the name of exo-insertion and -deletion) were considered in the P systems framework (i.e., with a graph control) and it was shown that systems with insertion of strings of length 2 (respectively 1) and deletion of strings of length 1 (respectively 2) lead to computational completeness. In the case of minimal insertion and deletion (i.e., of only one symbol), a priority of deletion over insertion (corresponding to an appearance check) was used to show computational completeness.

In this article we continue these investigations and we consider P systems with minimal left and right insertion and deletion and prove that computational completeness can be achieved even in this case, with the structure of the P system we need being matrix-like. We also directly show that matrix grammars using minimal left insertion and minimal right deletion rules are computationally complete (with matrices of length at most 3). Moreover, we also prove that using an additional minimal mutation operation (substitution of one symbol by another one) allows for reducing the height of the tree structure of the P system to the minimum size 1.

2 Preliminaries

After some preliminaries from formal language theory, we define the string rewriting rules to be used in this paper. As string rewriting systems, we will consider Post systems, matrix grammars, and sequential P systems. Moreover, we will give some examples and preliminary results to illustrate our definitions.

The set of non-negative integers is denoted 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^+. Let $\{a_1, ..., a_n\}$ be an arbitrary alphabet; the number of occurrences of a symbol a_i in x is denoted by $|x|_{a_i}$; the number of occurrences of all symbols from V in x is denoted by $|x|$. The family of recursively enumerable string languages is denoted by RE. For more details of formal language theory the reader is referred to the monographs and handbooks in this area as [10, 27].

We here consider string rewriting rules only working at the ends of a string:

Post rewriting rule $P\,[x/y]$ with $x, y \in V^*$: $P\,[x/y]\,(wx) = yw$ for $w \in V^*$.
Left substitution $S_L\,[x/y]$ with $x, y \in V^*$: $S_L\,[x/y]\,(xw) = yw$ for $w \in V^*$.
Right substitution $S_R\,[x/y]$ with $x, y \in V^*$: $S_R\,[x/y]\,(wx) = wy$ for $w \in V^*$.

If in a (left or right) substitution $S_L\,[x/y]$ or $S_R\,[x/y]$ x is empty, then we call it an *insertion* and write $I_L\,[y]$ and $I_R\,[y]$, respectively; if in a (left or right) substitution $S_L\,[x/y]$ or $S_R\,[x/y]$ y is empty, then we call it a *deletion* and write $D_L\,[x]$ and $D_R\,[x]$, respectively. If we only insert one symbol a, then we will also write $+a$, $a+$, $-a$, and $a-$ for $I_L\,[a]$, $I_R\,[a]$, $D_L\,[a]$, and $D_R\,[a]$, respectively.

In general, a *(string rewriting) grammar* G of type X is a construct (V, T, A, P) where V is a set of *symbols*, $T \subseteq V$ is a set of *terminal symbols*, $A \in V^+$ is the *axiom*, and P is a finite set of *rules* of type X. Each rule $p \in P$ induces a relation $\Longrightarrow_p \subseteq V^* \times V^*$; p is called *applicable* to a string $x \in V^*$ if and only if there exists at least one string $y \in V^*$ such that $(x, y) \in \Longrightarrow_p$; we also write $x \Longrightarrow_p y$. The derivation relation $\Longrightarrow_G$ is the union of all $\Longrightarrow_p$, i.e., $\Longrightarrow_G = \cup_{p \in P} \Longrightarrow_p$. The reflexive and transitive closure of $\Longrightarrow_G$ is denoted by $\overset{*}{\Longrightarrow}_G$.

The *language generated by* G is the set of all terminal strings derivable from the axiom, i.e., $L(G) = \left\{ v \in T^* \mid A \overset{*}{\Longrightarrow}_G v \right\}$. The family of languages generated by grammars of type X is denoted by $\mathcal{L}(X)$.

In general, we write $S_R^{k,m}$ for a type of grammars using only substitution rules $S_R[x/y]$ with $|x| \leq k$ and $|y| \leq m$. In the same way, we define the type $S_L^{k,m}$ for a type of grammars using only substitution rules $S_L[x/y]$ with $|x| \leq k$ and $|y| \leq m$, as well as the types I_L^m, I_R^m, D_L^k, and D_R^k, respectively. The type $D^k I^m$ allows for the deletion of strings with length $\leq k$ and for the insertion of strings with length $\leq m$. If, in addition, we also allow substitutions $S_R[x/y]$ with $|x| \leq k'$ and $|y| \leq m'$, we get the type $D^k I^m S^{k'm'}$; we observe that the type $D^k I^m S^{k'm'}$ is subsumed by the type $S^{k'm'}$ if $k \leq k'$ and $m \leq m'$. If we allow the parameters k and/or m to be arbitrarily large, we just omit them, e.g., DI is the type allowing to use deletions and insertions of strings of arbitrary lengths.

Example 1. Let $G = (V, T, A, P)$ be a regular grammar, i.e., the rules in P are of the form $A \to bC$ and $A \to \lambda$ with $A, C \in V \setminus T$ and $b \in T$. Then the grammar $G' = (V, T, A, \{S_R[A/y] \mid A \to y \in P\})$ with substitution rules generates the same language as G, i.e., $L(G') = L(G)$. Hence, with REG denoting the family of regular languages, we obviously have got $REG \subseteq \mathcal{L}\left(S_R^{1,2}\right)$. □

It is not difficult to check that grammars of type $D^1 I^1 S^1$ have a rather limited computational power. Indeed, we can show the following representation of languages generated by grammars of type $D^1 I^1 S^1$:

Theorem 1. *Every language $L \subseteq T^*$ in $\mathcal{L}\left(D^1 I^1 S^1\right)$ can be written in the form $T_l^* S T_r^*$ where $T_l, T_r \subseteq T$ and S is a finite subset of T^*.*

Proof. Let $G = (V, T, A, P)$ be a grammar of type $D^1 I^1 S^1$ and let $N := V \setminus T$. We first construct the start set S as follows: Consider all possible derivations in G from A with only using substitutions and deletions, but without loops, i.e., no string is allowed to appear more than once in such a derivation, which means that all these derivations are of bounded length (bounded by the number of strings in V of length at most $|V|$).Then S consists of all terminal strings obtained in this way (finding these strings is a finitely bounded process, as to each of the possible strings in V of length at most $|V|$, at most $|P|$ rules can be applied). A symbol from N remaining inside a string blocks that string from ever becoming terminal by applying rules from P, and deletion of a symbol can be avoided by

just not introducing the symbol which by a sequence of minimal substitutions would lead to the symbol to be deleted. Hence, for constructing the sets T_l (T_r, respectively) we can restrict ourselves to the terminal symbols b either directly inserted by minimal insertion rules $I_l[b]$ ($I_r[b]$, respectively) or obtained by a sequence of one minimal insertion together with a bounded (by $|V|$) number of minimal substitutions $S_l[a/b]$ ($S_r[a/b]$, respectively).

Therefore, in sum $L(G)$ can be written as the finite union of languages generated by grammars of type I^1, i.e., $L(G) = \cup_{w \in S} L(G_w)$ where

$$G_w = (T, T, w, \{I_l[b] \mid b \in T_l\} \cup \{I_r[b] \mid b \in T_r\}).$$

In fact, this representation of languages in $\mathcal{L}\left(D^1I^1S^1\right)$ means that for the type $D^1I^1S^1$ we could forget minimal deletions and substitutions and instead consider finite subsets of axioms instead of a single axiom. Putting an A in front of the types for this variant of grammars, we just have proved that

$$\mathcal{L}\left(A\text{-}D^1I^1S^1\right) = \mathcal{L}\left(A\text{-}I^1\right). \qquad \square$$

2.1 Post Systems

A *Post system* is a grammar using only Post rewriting rules (a grammar of *type* PS). A Post system (V, T, A, P) is said to be in *normal form* (a grammar of *type* $PSNF$) if and only if the Post rewriting rules $P[x/y]$ in P are only of the forms $P[ab/c]$, $P[a/bc]$, $P[a/b]$, and $P[a/\lambda]$, with $a, b, c \in V$. A Post system (V, T, A, P) is said to be in *Z-normal form* (a grammar of *type* $PSZNF$) if and only if it is in normal form and, moreover, there exists a special symbol $Z \in V \setminus T$ such that

- Z appears only once in the string x of a Post rewriting rule $P[x/y]$, and this rule is $P[Z/\lambda]$;
- if the rule $P[Z/\lambda]$ is applied, the derivation in the Post system stops yielding a terminal string;
- a terminal string can only be obtained by applying the rule $P[Z/\lambda]$.

Although basic results concerning Post systems are folklore since many years, e.g., see [23], we need the special Z-normal form for the proof of our main theorem; the following result is an immediate consequence of the proof given in [12] for Lemma 1 there:

Theorem 2. *For every recursively enumerable language $L \subseteq T^*$ there exists a Post rewriting system G, $G = (V, T, A, P)$, in Z-normal form such that $L(G) = L$, i.e., $\mathcal{L}(PS) = \mathcal{L}(PSNF) = \mathcal{L}(PSZNF) = RE$.*

2.2 Matrix Grammars

A *matrix grammar* of type X is a construct $G_M = (G, M)$ where $G = (V, T, A, P)$ is a grammar of type X, M is a finite set of sequences of the form $(p_1, \ldots, p_n)$, $n \geq 1$, of rules in P. For $w, z \in V^*$ we write $w \Longrightarrow_{G_M} z$ if there are a matrix

$(p_1, \ldots, p_n)$ in M and objects $w_i \in V^*$, $1 \leq i \leq n+1$, such that $w = w_1$, $z = w_{n+1}$, and, for all $1 \leq i \leq n$, $w_i \Longrightarrow_G w_{i+1}$. The maximal length n of a matrix $(p_1, \ldots, p_n) \in M$ is called the degree of G_M.

$L(G_M) = \left\{v \in T^* \mid A \Longrightarrow^*_{G_M} v\right\}$ is the language generated by G_M. The family of languages generated by matrix grammars of type X (of degree at most n) is denoted by $\mathcal{L}(X\text{-}MAT)$ ($\mathcal{L}(X\text{-}MAT_n)$).

Theorem 3. $\mathcal{L}\left(D^2I^2\text{-}MAT_2\right) = \mathcal{L}\left(D^1I^1\text{-}MAT_3\right) = \mathcal{L}(PSNF) = RE.$

Proof. From Theorem 2 we know that $\mathcal{L}(PSNF) = RE$, hence, we will only show that for every Post system $G = (V, T, A, P)$ in normal form we are able to construct equivalent matrix grammars $G_1 = (G, M_1)$ and $G_2 = (G, M_2)$ of type D^2I^2 and of type D^1I^1, respectively:

$$\begin{aligned} M_1 &= \{(D_R[x], I_L[y]) \mid P[x/y] \in P\}, \\ M_2 &= \{(D_R[b], D_R[a], I_L[c]) \mid P[ab/c] \in P\} \\ &\quad \cup \{(D_R[a], I_L[c], I_L[b]) \mid P[a/bc] \in P\} \\ &\quad \cup \{(D_R[a], I_L[b]) \mid P[a/b] \in P\} \\ &\quad \cup \{(D_R[a]) \mid P[a/\lambda] \in P\}. \end{aligned}$$

As each rule in G is directly simulated by a matrix in M_1 and in M_2, respectively, we immediately infer $L(G) = L(G_1) = L(G_2)$. □

Whereas the matrices in M_1 are only of length 2, the degree of M_2 is 3; it remains as an open question whether also with rules of type D^1I^1 we could decrease the degree to 2 or not; we conjecture that the answer is *no*. As we have shown in Theorem 1, with grammars using rules of type $D^1I^1S^1$ we are not able to obtain RE, we even remain below the regular language class; hence, we need such regulating mechanisms as matrices to reach computational compleness.

2.3 P Systems

We now introduce another variant to guide the derivations in a grammar using rules of those types introduced above, i.e., specific variants of left and right substitution rules.

A *(sequential) P system of type X with tree height n* is a construct $\Pi = (G, \mu, R, i_0)$ where $G = (V, T, A, P)$ is a grammar with rules of type X and

- μ is the membrane (tree) structure of the system with the height of the tree being n (μ usually is represented by a string containing correctly nested marked parentheses); we assume the membranes, i.e., the nodes of the tree representing μ, being uniquely labelled by labels from a set Lab;
- R is a set of rules of the form (h, r, tar) where $h \in Lab$, $r \in P$, and tar, called the *target indicator*, is taken from the set $\{here, in, out\} \cup \{in_j \mid 1 \leq j \leq n\}$; the rules assigned to membrane h form the set $R_h = \{(r, tar) \mid (h, r, tar) \in R\}$, i.e., R can also be represented by the vector $(R_h)_{h \in Lab}$;

- i_0 is the initial membrane where the axiom A is put at the beginning of a computation.

As we only have to follow the trace of a single string during a computation of the P system, a configuration of Π can be described by a pair (w, h) where w is the current string and h is the label of the membrane currently containing the string w. For two configurations (w_1, h_1) and (w_2, h_2) of Π we write $(w_1, h_1) \Longrightarrow_\Pi (w_2, h_2)$ if we can pass from (w_1, h_1) to (w_2, h_2) by applying a rule $(h_1, r, tar) \in R$, i.e., $w_1 \Longrightarrow_r w_2$ and w_2 is sent from membrane h_1 to membrane h_2 according to the target indicator tar. More specifically, if $tar = here$, then $h_2 = h_1$; if $tar = out$, then the string w_2 is sent to the region h_2 immediately outside membrane h_1; if $tar = in_{h_2}$, then the string is moved from region h_1 to the region h_2 immediately inside region h_1; if $tar = in$, then the string w_2 is sent to one of the regions immediately inside region h_1.

A sequence of transitions between configurations of Π, starting from the initial configuration (A, i_0), is called a *computation* of Π. A *halting computation* is a computation ending with a configuration (w, h) such that no rule from R_h can be applied to w anymore; (w, h) is called the result of this halting computation if $w \in T^*$. $L(\Pi)$, the language generated by Π, consists of all strings over T which are results of a halting computation in Π.

By $\mathcal{L}(X\text{-}LP)$ $(\mathcal{L}(X\text{-}LP^{\langle n \rangle}))$ we denote the family of languages generated by P systems (of tree height at most n) using rules of type X. If only the targets $here, in, out$ are used, then the P system is called *simple*, and the corresponding families of languages are denoted by $\mathcal{L}(X\text{-}LsP)$ $(\mathcal{L}(X\text{-}LsP^{\langle n \rangle}))$.

Example 2. Let $\Pi = (G, [_1 [_2]_2 [_3]_3 [_4]_4]_1, R, 1)$ be a P system of type $D_R^1 I_L^2$ with

$$\begin{aligned} G &= (\{a, B\}, \{a\}, \{D_R[a], D_R[B], I_L[aa], I_L[B]\}, aB), \\ R &= \{(1, D_R[a], in_2), (1, D_R[B], in_3), (1, D_R[B], in_4)\} \\ &\quad \cup \{(2, I_L[aa], out), (3, I_L[B], out)\} \end{aligned}$$

The computations in Π start with aB in membrane (region) 1. In general, starting with a string $a^{2^n}B$, $n \geq 0$, in membrane 1, we may either delete B by the rule $(1, D_R[B], in_4)$, getting a^{2^n} as the terminal result in the elementary membrane 4 (a membrane is called *elementary* if and only if it contains no inner membrane) or delete B by the rule $(1, D_R[B], in_3)$. With the string a^{2^n} arriving in membrane 3, we get Ba^{2^n} in membrane 1 by the rule $(3, I_L[B], out)$. Now we double the number of symbols a by applying the sequence of rules $(1, D_R[a], in_2)$ and $(3, I_L[aa], out)$ 2^n times, finally obtaining $a^{2^{n+1}}B$. Hence, in sum we get $L(\Pi) = \{a^{2^n} \mid n \geq 0\}$ for the language generated by this P system μ of type $D_R^1 I_L^2$. □

3 Computational Completeness of P Systems with Minmal Substitution Rules

In this section we consider several variants of P systems with substitution rules of minimal size, the main result showing computational completeness for simple P systems with rules of type D^1I^1. Yet first we show that for any recursively enumerable language we can construct a P system, with the height of the tree structure being only 1 (which is the minimum possible according to Theorem 1), of type $D_R^1 I_L^1 S_R^1$, i.e., using minimal right insertions and minimal right deletions and mutations (substitutions).

Theorem 4. $\mathcal{L}\left(D_R^1 I_L^1 S_R^1\text{-}LP^{\langle 1\rangle}\right) = RE.$

Proof. From Theorem 2 we know that $\mathcal{L}(PSZNF) = RE$, hence, we will only show that for every Post system $G = (V, T, A, P)$ in Z-normal form we are able to construct equivalent P system Π of type $D_R^1 I_L^1 S_R^1$. We assume that the rules in P are labelled in a unique way by labels from a finite set Lab with $1 \notin Lab$ and $z \in Lab$. We now construct a P system Π, $\Pi = (G', \mu, R, 1)$, with a flat tree structure μ of height 1, i.e., with the outermost membrane (the so-called skin membrane) being labelled by 1, and all the other membranes being elementary membranes inside the skin membrane being labelled by labels from

$$Lab' = \{1, \#\} \cup \{l \mid l : p \in Lab\} \cup \left\{\bar{h} \mid h : P\left[a_h b_h / c_h\right] \in P\right\} \cup \left\{\bar{h} \mid h : P\left[a_h b_h / c_h\right] \in P\right\}.$$

$G' = (V', T, A, P')$, $V' = \left\{x, \bar{x}^l \mid x \in V, l \in Lab\right\} \cup \{\#\}$, and P' contains the minimal left insertion, right deletion, and right substitution rules contained in the rules of R as listed in the following:

$$\begin{array}{ll}
h : P\left[a_h b_h / c_h\right]: & \left(1, D_R\left[b_h\right], in_h\right),\ \left(h, S_R\left[a_h / \bar{a}_h^h\right], out\right),\ \left(h, I_L\left[\#\right], out\right), \\
& \left(1, D_R\left[\bar{a}_h^h\right], in_{\bar{h}}\right),\ \left(\bar{h}, I_L\left[c_h\right], out\right); \\
h : P\left[a_h / b_h c_h\right]: & \left(1, S_R\left[a_h / \bar{a}_h^h\right], in_h\right),\ \left(h, I_L\left[c_h\right], out\right), \\
& \left(1, D_R\left[\bar{a}_h^h\right], in_{\bar{h}}\right),\ \left(\bar{h}, I_L\left[b_h\right], out\right); \\
h : P\left[a_h / b_h\right]: & \left(1, D_R\left[a_h\right], in_h\right),\ \left(h, I_L\left[b_h\right], out\right); \\
h : P\left[a_h / \lambda\right]: & \left(1, S_R\left[a_h / a_h\right], in_h\right),\ \left(l, D_R\left[a_h\right], out\right), \text{ for } a_h \neq Z; \\
z : P\left[Z / \lambda\right]: & \left(D_R\left[Z\right], in_z\right);
\end{array}$$

the additional membrane # is used to trap all computations not leading to a terminal string in an infinite loop by the rules $\left(1, I_L\left[\#\right], in_\#\right)$ and $\left(\#, I_L\left[\#\right], out\right)$; for this purpose, the rule $\left(h, I_L\left[\#\right], out\right)$ is used in case of $h : P\left[a_h b_h / c_h\right]$, too. Due to the features of the underlying Post system in Z-normal form, all terminal strings from $L(G)$ can be obtained as final results of a halting computation in the elementary membrane z, whereas all other possible computations in Π never halt, finally being trapped in an infinite loop guaranteed by the rules leading into and out from membrane #. Hence, in sum we get $L(\Pi) = L(G)$. □

Summarizing the results of Theorems 1 and 4, we get:

Corollary 1. $\mathcal{L}\left(D^1I^1S^1\right) = \mathcal{L}\left(D^1I^1S^1\text{-}LP^{\langle 0\rangle}\right) \subset REG \subset \mathcal{L}\left(D^1_RI^1_LS^1_R\text{-}LP^{\langle n\rangle}\right) = RE$ *for all* $n \geq 1$.

If we want to restrict ourselves to the simple targets $here, in, out$, then we have to use a more difficult proof technique than in the proof of Theorem 4.

Theorem 5. $\mathcal{L}\left(D^1I^1\text{-}LsP^{\langle 8\rangle}\right) = RE$.

Proof. In order to show the inclusion $RE \subseteq \mathcal{L}\left(D^1I^1\text{-}LsPLsP^{\langle 8\rangle}\right)$, as in the proof of Theorem 4 we start from a Post system $G = (V, T, A, P)$ in Z-normal form with assuming the rules in P to be labelled in a unique way by labels from a finite set Lab with $1 \notin Lab$ and $z \in Lab$ and construct an equivalent simple P system Π, $\Pi = (G', \mu, R, 1)$, of type D^1I^1, with $G' = (V', T, A, P')$ and

$V' = V \cup V_R \cup \{S\}$, $V_R = \{D, E, F, H, J, K, M\}$,
$P' = \{+X, -X \mid X \in V \cup \{S\}\} \cup \{X+, X- \mid X \in V \cup V_R\}$,

as follows: The membrane structure μ consists of the skin membrane 1 as well as of linear structures needed for the simulation of the rules in G: For every rule $h : P\,[a_hb_h/c_h]$ and every rule $h : P\,[a_h/b_hc_h]$ in P we need a linear structure of 8 membranes $\left[_{(h,1)}\,\left[_{(h,2)}\cdots\left[_{(h,8)}\;\right]_{(h,8)}\cdots\right]_{(h,2)}\right]_{(h,1)}$ and for every rule $h : P\,[a_h/b_h]$ and every rule $h : P\,[a_h/\lambda]$ in P we need a linear structure of 6 membranes $\left[_{(h,1)}\,\left[_{(h,2)}\cdots\left[_{(h,6)}\;\right]_{(h,6)}\cdots\right]_{(h,2)}\right]_{(h,1)}$; moreover, for getting the terminal results, we need the linear structure of 3 membranes $\left[_{(z,1)}\,\left[_{(z,2)}\,\left[_{(z,3)}\;\right]_{(z,3)}\right]_{(z,2)}\right]_{(z,1)}$.

The simulations of the other rules from P are accomplished by the procedures as shown in the tables below, where the columns have to be interpreted as follows: in the first column, the membrane (label) h is listed, in the second one only the rule $p \in P$ is given, which in total describes the rule $(h, p, in) \in R$, whereas the rule p in the fifth column has to be interpreted as the rule $(h, p, out) \in R$.; the strings in the third and the fourth column list the strings obtained when going up in the linear membrane structure with the rules (h, p, in) from column 2 and going down with the rules (h, p, out) from column 5, respectively. The symbol F cannot be erased anymore, hence, whenever F has been introduced, at some moment, the computation will land in an infinite loop with only introducing more and more symbols F. The main idea of the proof is that we choose the membrane to go into by the rule $(1, K+, in)$ in a non-deterministic way. The goal is to reach the terminal membrane $(z, 3)$ starting with a string wZ, $w \in T^*$, in the skin membrane:

$(z,3)$		w		
$(z,2)$	$Z-$	wZ		$F+$
$(z,1)$	$K-$	wZK	wF	$F+$
1	$K+$	wZ	wFF	

Getting the terminal string $w \in T^*$

The tables below are to be interpreted in the same way as above; yet now we only list the results of correct simulations in column 4 and omit the results of adding the trap symbol F. Moreover, the rule $D-$ in the skin membrane is the only one in the whole system which uses the target *here*, i.e., it has to be interpreted as $(1, D-, here)$.

$(h,8)$		$ScwDH$		$H-$, $F+$
$(h,7)$	$+S$	$cwDH$	$ScwD$	$E+$, $F+$
$(h,6)$	$+c$	wDH	$ScwDE$	$M+$, $F+$
$(h,5)$	$H+$	wD	$ScwDEM$	$-S$, $F+$
$(h,4)$	$D+$	w	$cwDEM$	$M-$, $F+$
$(h,3)$	$a-$	wa	$cwDE$	$J+$, $F+$
$(h,2)$	$b-$	wab	$cwDEJ$	$J-$, $F+$
$(h,1)$	$K-$	$wabK$	$cwDE$	$E-$, $F+$
1	$K+$	wab	cwD	$D-$
			cw	

Simulation of $h : P\,[ab/c]$

$(h,8)$		$SbcwDH$		$H-$, $F+$
$(h,7)$	$+S$	$bcwDH$	$SbcwD$	$E+$, $F+$
$(h,6)$	$+b$	$cwDH$	$SbcwDE$	$M+$, $F+$
$(h,5)$	$+c$	wDH	$SbcwDEM$	$-S$, $F+$
$(h,4)$	$H+$	wD	$bcwDEM$	$M-$, $F+$
$(h,3)$	$D+$	w	$bcwDE$	$J+$, $F+$
$(h,2)$	$a-$	wa	$bcwDEJ$	$J-$, $F+$
$(h,1)$	$K-$	waK	$bcwDE$	$E-$, $F+$
1	$K+$	wa	$bcwD$	$D-$
			bcw	

Simulation of $h : P\,[ab/c]$

$(r,6)$		$SwDH$		$H-$, $F+$
$(r,5)$	$+S$	wDH	SwD	$E+$, $F+$
$(r,4)$	$H+$	wD	$SwDE$	$S-$, $F+$
$(r,3)$	$D+$	w	wDE	$J+$, $F+$
$(r,2)$	$a-$	wa	$wDEJ$	$J-$, $F+$
$(r,1)$	$K-$	waK	wDE	$E-$, $F+$
1	$K+$	wa	wD	$D-$
			w	

Simulation of $h : P\,[a/\lambda]$, $a \neq Z$

$(h,6)$		$SbwD$		$D-$, $F+$
$(h,5)$	$+S$	wD	Sbw	$E+$, $F+$
$(h,4)$	$+b$	wD	$SbwE$	$-S$, $F+$
$(h,3)$	$D+$	w	bwE	$J+$, $F+$
$(h,2)$	$a-$	wa	$bwEJ$	$J-$, $F+$
$(h,1)$	$K-$	waK	bwE	$E-$, $F+$
1	$K+$	wa	bw	

Simulation of $h : P\,[a/b]$

From the descriptions given in the tables above, it is easy to see how a successful simulation of a rule $h : P\,[x_h/y_h] \in P$ works. If we enter a membrane $(h,1)$ with a string v not being of the form ux_h, then at some moment the only

chance will be to use $F+$, introducing the trap symbol F which cannot be erased anymore and definitely leads to a non-halting computation. The additional symbols D, E, H, J, M intermediately introduced on the right-hand side of the string guarantee that loops inside the linear membrane structure for the simulation of a rule $h : P\,[x_h/y_h] \in P$ cannot lead to successful computations as well. In sum, we conclude $L\,(\Pi) = L\,(G)$. □

Due to the matrix-like membrane structure of the simple P systems constructed in the preceding proof, we could obtain the computational completeness of matrix grammars of type D^1I^1 as an obvious consequence of Theorem 5, yet the direct transformation of the construction given in the proof of this theorem would yield a lot of matrices with lengths more than 3, whereas the direct proof given in Theorem 3 only needed matrices of length at most 3.

4 Conclusion

In this paper we have considered string rewriting systems using the operations of minimal left and right insertion and deletion. Using even only the operations of minimal left insertion and minimal right deletion, matrix grammars reach computational completeness with matrices of length at most 3; our conjecture is that this required length cannot be reduced to 2. As our main result, we have shown that sequential P systems using the operations of minimal left and right insertion and deletion are computationally complete, thus solving an open problem from [2]. The simple P system constructed in the proof of Theorem 5 had rather large tree height; it remains an open question to reduce this complexity parameter. On the other hand, in Theorem 4 we have shown that using minimal left insertion, minimal right deletion, and, in addition, minimal right mutation (substitution of one symbol by another one on the right-hand side of a string) we can reduce the height of the tree structure of the P system to the minimum 1 and even avoid the use of the target *here*. Moreover, we would also like to avoid the target *here* in the case of simple P systems using minimal left and right insertion and deletion, as with avoiding the target *here*, the applications of the rules could be interpreted as being carried out when passing a membrane, in the sense of a molecule passing a specific mebrane channel from one region to another one. We shall return to this qestion and related ones in an extended version of this paper.

References

1. A. Alhazov, A. Krassovitskiy, Y. Rogozhin, S. Verlan: P Systems with minimal insertion and deletion. *Theor. Comp. Sci.* **412** (1-2), 136–144 (2011).
2. A. Alhazov, A. Krassovitskiy, Y. Rogozhin, S. Verlan: P Systems with insertion and deletion exo-operations. *Fundamenta Informaticae* **110** (1-4), 13–28 (2011).

3. A. Alhazov, A. Krassovitskiy, Y. Rogozhin: Circular Post machines and P systems with exo-insertion and deletion. In: M. Gheorghe et al.(eds.): *Membrane Computing - 12th International Conference, CMC 2011, Fontainebleau, France, August 23-26, 2011, Revised Selected Papers*, LNCS **7184**, Springer, 73–86 (2012).
4. R. Benne: RNA Editing: The Alteration of Protein Coding Sequences of RNA. Ellis Horwood, Chichester, West Sussex, 1993.
5. F. Biegler, M. J. Burrell, and M. Daley: Regulated RNA rewriting: Modelling RNA editing with guided insertion. *Theor. Comput. Sci.* **387** (2), 103–112 (2007).
6. E. Csuhaj-Varjú, A. Salomaa: Networks of parallel language processors. In: Gh. Păun, A. Salomaa (eds.): *New Trends in Formal Languages*. LNCS **1218**, 299–318, Springer, Heidelberg (1997).
7. J. Castellanos, C. Martín-Vide, V. Mitrana, J.M. Sempere: Solving NP-complete problems with networks of evolutionary processors. In: J. Mira, A.G. Prieto (eds.): *IWANN 2001*. LNCS **2084**, 621–628, Springer, Heidelberg (2001).
8. J. Dassow, F. Manea, B. Truthe: On normal forms for networks of evolutionary processors. *Proc. UC 2011*, 89–100 (2011).
9. J. Dassow, F. Manea: Accepting hybrid networks of evolutionary processors with special topologies and small communication. *Proc. DCFS 2010*, 68–77 (2010).
10. J. Dassow, Gh. Păun: *Regulated Rewriting in Formal Language Theory*. Springer-Verlag, 1989.
11. R. Freund, M. Kogler, Y. Rogozhin, S. Verlan: Graph-controlled insertion-deletion systems. In: I. McQuillan, G. Pighizzini (eds.): *Proc. of 12^{th} Workshop on Descriptional Complexity of Formal Systems, vol. **31** of EPTCS*, 88–98 (2010).
12. R. Freund, M. Oswald, A. Păun: Gemmating P systems are computationally complete with four membranes. In: L. Ilie, D. Wotschke: *Pre-proceedings DCFS 2004*. The University of Western Ontario, Rep. No. 619, 191–203 (2004).
13. B. Galiukschov: Semicontextual grammars. *Logica i Matem. Lingvistika*, 38–50. Tallin University (in Russian) (1981).
14. D. Haussler: *Insertion and Iterated Insertion as Operations on Formal Languages*. PhD thesis, Univ. of Colorado at Boulder, 1982.
15. D. Haussler: Insertion languages. *Information Sciences* **31** (1), 77–89 (1983).
16. S. Ivanov, S. Verlan: Random context and semi-conditional insertion-deletion systems. *arXiv*, CoRR abs/1112.5947 (2011).
17. L. Kari: *On Insertion and Deletion in Formal Languages*. PhD thesis, University of Turku, 1991.
18. L. Kari, G. Păun, G. Thierrin, 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,* 318–333, Philadelphia (1997).
19. A. Krassovitskiy, Y. Rogozhin, S. Verlan: Computational power of insertion-deletion (P) systems with rules of size two. *Natural Computing* **10** (2), 835–852 (2011).
20. S. Marcus: Contextual Grammars. *Rev. Roum. Math. Pures Appl.* **14**, 1525–1534 (1969).
21. M. Margenstern, V. Mitrana, M. Pérez-Jiménez: Accepting hybrid networks of evolutionary systems. *Proc. DNA10*, LNCS **3384**, 235–246 (2005).
22. M. Margenstern, G. Păun, Y. Rogozhin, S. Verlan: Context-free insertion-deletion systems. *Theor. Comput. Sci.* **330** (2), 339–348 (2005).
23. M.L. Minsky: *Computation: Finite and Infinite Machines.* Prentice Hall, Englewood Cliffs, New Jersey, USA, 1967.
24. G. Păun: *Membrane Computing. An Introduction.* Springer-Verlag, 2002.

25. Gh. Păun, G. Rozenberg, A. Salomaa: *The Oxford Handbook of Membrane Computing*. Oxford University Press, 2010.
26. G. Păun, G. Rozenberg, A. Salomaa: DNA Computing: *New Computing Paradigms*. Springer, 1998.
27. G. Rozenberg, A. Salomaa: *Handbook of Formal Languages,* 3 volumes. Springer-Verlag, Berlin, Heidelberg, New York, 1997.
28. I. Petre, S. Verlan: Matrix insertion-deletion systems. *arXiv*, CoRR abs/1012.5248, 2010.
29. The P systems Web page. `http://ppage.psystems.eu/`
30. S. Verlan: Recent developments on insertion-deletion systems. *Comp. Sci. J. of Moldova* **18** (2), 210–245 (2010).
31. S. Verlan: *Study of language-theoretic computational paradigms inspired by biology.* Habilitation thesis, University of Paris Est, 2010.
32. S. Verlan: On minimal context-free insertion-deletion systems. *J. of Automata, Languages and Combinatorics* **12** (1-2), 317–328 (2007).