P Systems with Minimal Insertion and Deletion

Artiom Alhazov[1,3], Alexander Krassovitskiy[2],
Yurii Rogozhin[2,3], Sergey Verlan[4]

[1] IEC, Department of Information Engineering, Graduate School of Engineering
Hiroshima University, Higashi-Hiroshima 739-8527 Japan
[2] Research Group on Mathematical Linguistics, Rovira i Virgili University
Av. Catalunya, 35, Tarragona 43002 Spain
alexander.krassovitskiy@estudiants.urv.cat
[3] Institute of Mathematics and Computer Science, Academy of Sciences of Moldova
Academiei 5, Chişinău MD-2028 Moldova
{artiom,rogozhin}@math.md
[4] LACL, Département Informatique, Université Paris 12
61 av. Général de Gaulle, 94010 Créteil, France
verlan@univ-paris12.fr

Summary. In this paper we consider insertion-deletion P systems with priority of deletion over the insertion. We show that such systems with one symbol context-free insertion and deletion rules are able to generate $PsRE$. If one-symbol one-sided context is added to insertion or deletion rules but no priority is considered, then all recursively enumerable languages can be generated. The same result holds if a deletion of two symbols is permitted. We also show that the priority relation is very important and in its absence the corresponding class of P systems is strictly included in MAT.

1 Introduction

The operations of insertion and deletion are fundamental in formal language theory, and generative mechanisms based on them were considered (with linguistic motivation) for some time, see [14] and [6]. Related formal language investigations can be found in several places; we mention only [8], [10], [16], [19]. In the last years, the study of these operations has received a new motivation from molecular computing, see [3], [9], [21], [23], [15].

In general form, an insertion operation means adding a substring to a given string in a specified (left and right) context, while a deletion operation means removing a substring of a given string from a specified (left and right) context. A finite set of insertion-deletion rules, together with a set of axioms provide a language generating device: starting from the set of initial strings and iterating insertion-deletion operations as defined by the given rules we get a language. The number of axioms, the length of the inserted or deleted strings, as well as the

length of the contexts where these operations take place are natural descriptional complexity measures in this framework. As expected, insertion and deletion operations with context dependence are very powerful, leading to characterizations of recursively enumerable languages. Most of the papers mentioned above contain such results, in many cases improving the complexity of insertion-deletion systems previously available in the literature.

Some combinations of parameters lead to systems which are not computationally complete [17], [11] or even decidable [24]. However, if these systems are combined with the distributed computing framework of P systems [20], then their computational power may strictly increase, see [12], [13].

In this paper we study P systems with insertion and deletion rules of one symbol without context. We show that this family is strictly included in MAT, however some non-context-free languages may be generated. If Parikh vectors are considered, then the corresponding family equals to $PsMAT$. When a priority of deletion over insertion is introduced, $PsRE$ can be characterized, but in terms of language generation such systems cannot generate a lot of languages because there is no control on the position of an inserted symbol. If one-sided contextual insertion or deletion rules are used, then this can be controlled and all recursively enumerable languages can be generated. The same result holds if a context-free deletion of two symbols is allowed.

2 Definitions

All formal language notions and notations we use here are elementary and standard. The reader can consult any of the many monographs in this area – for instance, [22] – for the unexplained details.

We denote by $|w|$ the length of a word w and by $|w|_a$ the number of occurrences of symbol a in w. For a word $w \in V^*$ we denote by $\Delta(w)$ all words w' having the same number of letters as w, $\Delta(w) = \{w' \mid, |w'|_a = |w|_a \text{ for all } a \in V\}$ and we denote by $\sqcup\!\sqcup$ the binary shuffle operation. By $card(V)$ we denote the cardinality of the set V.

An *InsDel system* is a construct $ID = (V, T, A, I, D)$, where V is an alphabet, $T \subseteq V$, A is a finite language over V, and I, D are finite sets of triples of the form (u, α, v), $\alpha \neq \lambda$, where u and v are strings over V and λ denotes the empty string. The elements of T are *terminal* symbols (in contrast, those of $V - T$ are called nonterminals), those of A are *axioms*, the triples in I are *insertion rules*, and those from D are *deletion rules*. An insertion rule $(u, \alpha, v) \in I$ indicates that the string α can be inserted in between u and v, while a deletion rule $(u, \alpha, v) \in D$ indicates that α can be removed from the context (u, v). As stated otherwise, $(u, \alpha, v) \in I$ corresponds to the rewriting rule $uv \to u\alpha v$, and $(u, \alpha, v) \in D$ corresponds to the rewriting rule $u\alpha v \to uv$. We refer by $\Longrightarrow$ to the relation defined by an insertion or deletion rule.

The language $L(ID)$ generated by ID is defined as $\{w \in T^* \mid A \ni x \Longrightarrow^* w\}$.

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

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

We also denote by $INS_n^{m,m'}DEL_p^{q,q'}$ corresponding families of languages. Traditionally, in the literature, instead of pairs m/m' and q/q' the maximum of both numbers is used. However, such a complexity measure is not accurate and it cannot distinguish between universality and non-universality cases, see [24] and [11]. If some of the parameters n, m, m', p, q, q' is not specified, then we write symbol $*$ instead. For example, $INS_*^{0,0}DEL_*^{0,0}$ denotes the family of languages generated by *context-free InsDel systems*. InsDel systems of a "sufficiently large" size characterize RE, the family of recursively enumerable languages.

Now we present a definition of insertion-deletion P systems. The *insertion-deletion tissue P systems* are defined in an analogous manner.

An *insertion-deletion P system* is a construct

$\Pi = (O, T, \mu, M_1, \cdots, M_n, R_1, \cdots, R_n)$, where

- O is a finite alphabet,
- $T \subseteq O$ is the terminal alphabet,
- μ is the membrane (tree) structure of the system which has n membranes (nodes) and it can be represented by a word over the alphabet of correctly nested marked parentheses,
- M_i, for each $1 \leq i \leq n$, is a finite language associated to the membrane i,
- R_i, for each $1 \leq i \leq n$, is a set of insertion and deletion rules with target indicators associated to region i, of the following forms: $(u, x, v; tar)_a$, where (u, x, v) is an insertion rule, and $(u, x, v; tar)_e$, where (u, x, v) is a deletion rule, and the *target indicator tar* is from the set $\{here, in_j, out \mid 1 \leq j \leq n\}$.

An n-tuple $(N_1, \cdots, N_n)$ of finite languages over O is called a configuration of Π. The transition between the configurations consists in applying the insertion and deletion rules in parallel to all possible strings, non-deterministically, and following the target indications associated with the rules.

A sequence of transitions between configurations of a given insertion-deletion P system Π starting from the initial configuration is called a computation with respect to Π. We say that Π generates $L(\Pi)$, the result of its computations. It consists of all strings over T ever sent out of the system during its computations.

We denote by $ELSP_k(ins_p^{m,m'}, del_p^{q,q'})$ the family of languages $L(\Pi)$ generated by insertion-deletion P systems with at most $k \geq 1$ membranes and insertion and deletion rules of size at most $(n, m, m'; p, q, q')$. We omit the letter E if $T = O$. In this paper we also consider insertion-deletion P systems where deletion rules have a priority over insertion rules; the corresponding class is denoted as

$(E)LSP_k(ins_p^{m,m'} < del_p^{q,q'})$. Letter "t" is inserted before P to denote classes for the tissue case, e.g., $ELStP_k(ins_p^{m,m'}, del_p^{q,q'})$.

A *register machine* (introduced in [18], see also [4]) is a construct

$M = (d, Q, q_0, h, P)$, where

- d is the number of registers,
- Q is a finite set of bijective labels of instructions of P,
- $q_0 \in Q$ is the initial label,
- $h \in Q$ is the halting label, and
- P is the set of instructions of the following forms:

1. $p : (\mathrm{ADD}(k), q, s)$, with $p, q, s \in Q$, $1 \leq k \leq d$ ("increment" -instruction). Add 1 to register k and go to one of the instructions with labels q, s.
2. $p : (\mathrm{SUB}(k), q, s)$, with $p, q, s \in Q$, $1 \leq k \leq d$ ("decrement" -instruction). Subtract 1 from the positive value of register k and go to the instruction with label q, otherwise (if it is zero) go to the instruction with label s.
3. $h : HALT$ (the halt instruction). Stop the computation of the machine.

For generating languages over T, we use the model of a *register machine with output tape* (introduced in [18], see also [1]), which also uses a tape operation:

4. $p : (\mathrm{WRITE}(A), q)$, with $p, q \in Q$, $A \in T$.

The configuration of a register machine is $(q, n_1, \cdots, n_d)$, where $q \in Q$, $n_i \geq 0$, $1 \leq i \leq d$. A register machine generates an m-dimensional vector as follows: let the first m registers be output registers, and the computation starts from $(q_0, 0, \cdots, 0)$; if the configuration $(h, n_1, \cdots, n_d)$ is reached, then the resulting vector is $(n_1, \cdots, n_m)$. Without restricting generality we assume $(n_{m+1}, \cdots, n_d) = (0, \cdots, 0)$. The set of all vectors generated in this way by M is denoted by $Ps(M)$. It is known (e.g., see [18], [25]) that register machines generate $PsRE$. If the WRITE instruction is used, then RE can be generated.

In the case when a register machine cannot check whether a register is empty we say that it is partially blind; the second type of instructions is then written as $p : (\mathrm{SUB}(k), q)$ and the transition is undefined if register k is zero.

The word "partially" stands for an implicit test for zero at the end of a (successful) computation: counters $m + 1, \cdots, d$ should be empty. It is known, [4], that partially blind register machines generate exactly $PsMAT$ (Parikh sets of languages of matrix grammars without appearance checking).

3 Minimal Context-free Insertion-Deletion P Systems

It has been shown, [24], that systems in $INS_1^{0,0}DEL_*^{0,0}$ only generate strings obtained by inserting any number of specific symbols anywhere in words of a finite language; this is included in the regular languages family; strictly as, e.g., for

$L = \{a^*b^*\}$ the system has no control on the place of insertion or deletion in the string and the initial language is finite. Therefore, $INS_1^{0,0}DEL_1^{0,0} \subset REG$.

When a membrane structure is added to minimal insertion-deletion systems without context, their computational power is increased.

Theorem 1. $PsStP_*(ins_1^{0,0}, del_1^{0,0}) = PsMAT$.

Proof. It is not difficult to see that dropping the requirement of the uniqueness of the instructions with the same label, the power of partially blind register machines does not change, see, e.g., [4]. We use this fact for the proof.

The inclusion $PsStP_*(ins_1^{0,0}, del_1^{0,0}) \subseteq PsMAT$ follows from the simulation of minimal context-free insertion-deletion P systems by partially blind register machines, which are known to characterize $PsMAT$ [4]. Indeed, any rule $(\lambda, a, \lambda; q)_a \in R_p$ is simulated by instructions $p : (ADD(a), q)$. Similarly, rule $(\lambda, a, \lambda; q)_e \in R_p$ is simulated by instructions $p : (SUB(a), q)$.

The output region i_0 is associated to the final state, while the halting is represented by absence of the corresponding symbols (final zero-test) as follows. We assume that R_{i_0} has no insertion rules ($\emptyset$ can be generated by a trivial partially blind register machine), and the output registers correspond to those symbols that cannot be deleted by rules from R_{i_0}.

The converse inclusion follows from the simulation of partially blind register machines by P systems. Indeed, with every instruction p of the register machine we associate a cell. Instruction $p : (ADD(A_k), q)$ is simulated by rule $(\lambda, A_k, \lambda; q)_a \in R_p$, and instruction $p : (SUB(A_k), q)$ by $(\lambda, A_k, \lambda; q)_e \in R_p$. Final zero-tests: rules $(\lambda, A_k, \lambda; \#)_e \in R_h$, $k \geq m$, should be inapplicable ($R_\# = \emptyset$).

As the membrane structure is a tree, one-way inclusion follows.

Corollary 1. $PsSP_*(ins_1^{0,0}, del_1^{0,0}) \subseteq PsMAT$.

In terms of the generated language the above systems are not too powerful, even with priorities. Like in the case of insertion-deletion systems there is no control on the position of insertion. Hence, the language $L = \{a^*b^*\}$ cannot be generated, for insertion strings of any size. Hence we obtain:

Theorem 2. $REG \backslash LStP_*(ins_n^{0,0} < del_1^{0,0}) \neq \emptyset$, *for any* $n > 0$.

However, there are non-context-free languages that can be generated by such P systems (even without priorities and deletion).

Theorem 3. $LStP_*(ins_1^{0,0}, del_0^{0,0}) \setminus CF \neq \emptyset$.

Proof. It is easy to see that the language $\{w \in \{a, b, c\}^* : |w|_a = |w|_b = |w|_c\}$ is generated by such a system with 3 nodes, inserting consecutively a, b and c.

For the tree case the language $\{w \in \{a, b\}^* \mid |w|_a = |w|_b\}$ can be generated in a similar manner.

We show a more general inclusion:

Theorem 4. *$ELStP_*(ins_n^{0,0}, del_1^{0,0}) \subset MAT$, for any $n > 0$.*

Proof. As in [11] we can suppose that there are no deletions of terminal symbols. We also suppose that there is only one initial string in the system, because there is no interaction between different evolving strings and the result matches the union of results for the systems with only one string. Consider a tissue P system Π with alphabet O, terminal symbols T, the set H of unique cell labels and the initial string w in cell labeled p_0. Such a system can be simulated by the following matrix grammar $G = (O \cup H, T, S, P)$.

For insertion instruction $(\lambda, a_1 \cdots a_n, \lambda; q)_a$ in cell p, the matrix $(p \to q, D \to Da_1D \cdots Da_nD) \in P$. For any deletion instruction $(\lambda, A, \lambda; q)_e$ in cell p, the matrix $(p \to q, A \to \lambda) \in P$. Three additional matrices $(h \to \lambda)$, $(D \to \lambda)$ and $(S \to q_0Da_1D \cdots Da_mD)$ $(w = a_1 \cdots a_m)$ shall be also added to P.

The above construction correctly simulates the system Π. Indeed, symbols D represent placeholders for all possible insertions. The first rule in the matrix permits simulates the navigation between cells.

Nevertheless, minimal context-free insertion-deletion systems with priorities do generate $PsRE$. This is especially clear for the tissue P systems: jumping to an instruction corresponds to sending a string to the associated region, and the entire construction is a composition of graphs shown in Figure 1. The decrement instruction works correctly because of priority of deletion over insertion.

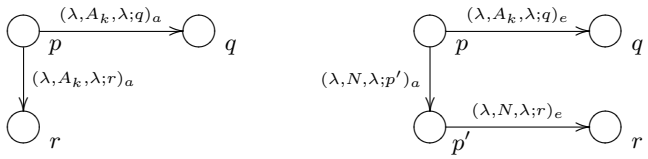

Fig. 1. Simulating $p : (ADD(k), q, r)$ (left) and $p : (SUB(k), q, r)$ (right).

We now give a more sophisticated proof for the tree-like membrane structure.

Theorem 5. *$PsSP_*(ins_1^{0,0} < del_1^{0,0}) = PsRE$.*

Proof. The proof is done by showing that for any register machine $M = (n, Q, q_0, h, P)$ there is a P system $\Pi \in PsSP_*(ins_1^{0,0} < del_1^{0,0})$ with $Ps(M) = Ps(\Pi)$, and the result follows from the existence of register machines generating $PsRE$.

Let Q_+ (Q_-) be the sets of labels of increment (conditional decrement, respectively) instructions of a register machine, and let $Q = Q_+ \cup Q_- \cup \{h\}$ represent all labels. Consider a P system with alphabet $Q \cup \{A_i \mid 1 \le i \le d\} \cup \{Y\}$ and the following structure (illustrated in Figure 2)

$$\mu = [\,[\,[\prod_{p \in Q_+} \mu_{\langle p+ \rangle} \prod_{p \in Q_-} \mu_{\langle p- \rangle}\,]_3\,]_2\,]_1, \text{ where}$$

$$\mu_{\langle p+ \rangle} = [\,[\,[\,]_{p_3^+}\,]_{p_2^+}\,]_{p_1^+}\ ,\ p- \text{ increment},$$

$$\mu_{\langle p- \rangle} = [\,[\,[\,]_{p_3^-}\,]_{p_2^-}\,[\,[\,]_{p_3^0}\,]_{p_2^0}\,]_{p_1^-}\ ,\ p- \text{ conditional decrement}.$$

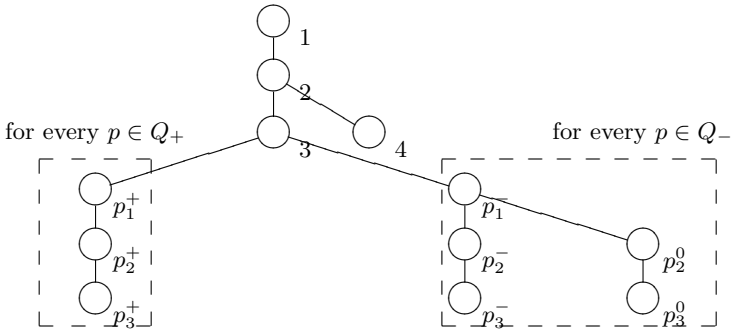

Fig. 2. Membrane structure for Theorem 5. The structures in the dashed rectangles are repeated for every instruction of the register machine.

Initially there is a single string q_0 in membrane 3. The rules are the following.

$$
\begin{aligned}
R_1 = \{ \quad & 1 : (\lambda, Y, \lambda; out)_e \}, \\
R_2 = \{ \quad & 2.1 : (\lambda, Y, \lambda; out)_a, && 2.2 : (\lambda, Y, \lambda; in_4)_e \}, \\
R_3 = \{ \quad & 3.1 : (\lambda, p, \lambda; in_{p_1^+})_e \mid p \in Q_+ \} \cup && \{3.2 : (\lambda, p, \lambda; in_{p_1^-})_e \mid p \in Q_- \} \\
\cup \{ \quad & 3.3 : (\lambda, Y, \lambda; here)_e, && 3.4 : (\lambda, h, \lambda; out)_e \},
\end{aligned}
$$

For any instruction $p : (ADD(k), q, s)$, $R_{p_3^+} = \emptyset$ and

$$
\begin{aligned}
R_{p_1^+} = \{ \quad & a.1.1 : (\lambda, A_k, \lambda; in_{p_2^+})_a, && a.1.2 : (\lambda, Y, \lambda; out)_a \}, \\
R_{p_2^+} = \{ \quad & a.2.1 : (\lambda, q, \lambda; out)_a, && a.2.1' : (\lambda, s, \lambda; out)_a, \\
& a.2.2 : (\lambda, q, \lambda; in_{p_3^+})_e, && a.2.2' : (\lambda, s, \lambda; in_{p_3^+})_e \},
\end{aligned}
$$

For any instruction $p : (SUB(k), q, s)$, $R_{p_3^-} = R_{p_3^0} = \emptyset$ and

$$
\begin{aligned}
R_{p_1^-} = \{ \quad & e.1.1 : (\lambda, A_k, \lambda; in_{p_2^-})_e, && e.1.2 : (\lambda, Y, \lambda; in_{p_2^-})_a, && e.1.3 : (\lambda, Y, \lambda; out)_e \}, \\
R_{p_2^-} = \{ \quad & e.2.1 : (\lambda, q, \lambda; out)_a, && e.2.2 : (\lambda, q, \lambda; in_{p_3^-})_e, \\
& e.2.3 : (\lambda, s, \lambda; in_{p_3^-})_e, && e.2.4 : (\lambda, Y, \lambda; here)_a \}, \\
R_{p_2^0} = \{ \quad & e.3.1 : (\lambda, s, \lambda; out)_a, && e.3.2 : (\lambda, q, \lambda; in_{p_3^0})_e, && e.3.3 : (\lambda, s, \lambda; in_{p_3^0})_e \}.
\end{aligned}
$$

Configurations $(p, x_1, \cdots, x_n)$ of M are encoded by strings $\Delta(pA_1^{x_1} \cdots A_n^{x_n} Y^t)$, $t \geq 0$, in membrane 3. We say that such strings have a *simulating* form. Clearly, in the initial configuration the string is already in the simulating form.

To prove that system Π correctly simulates M we prove the following claims:

1. For any transition $(p, x_1, \cdots, x_n) \Longrightarrow (q, x_1', \cdots, x_n')$ in M there exists a computation in Π from the configuration containing $\Delta(pA_1^{x_1} \cdots A_n^{x_n} Y^t)$ in membrane 3 to the configuration containing $\Delta(qA_1^{x_1'} \cdots A_n^{x_n'} Y^{t'})$, $t' \geq 0$, in membrane 3 such that during this computation membrane 3 is empty in all intermediate steps and, moreover, this computation is unique.

2. For any successful computation in Π (yielding a non-empty result), membrane 3 contains only strings of the above form.
3. The result $(x_1, \cdots, x_n)$ in Π is obtained if and only if a string of form $\Delta(hA_1^{x_1} \cdots A_n^{x_n})$ appears in membrane 3.

Now we prove each claim from above. Consider a string $\Delta(pA_1^{x_1} \cdots A_n^{x_n} Y^t)$, $t \geq 0$, in membrane 3 of Π. Take an instruction $p : (ADD(k), q, s) \in P$. The only applicable rule in Π is from group 3.1 (in the future we simply say rule 3.1) yielding the string $\Delta(A_1^{x_1} \cdots A_n^{x_n} Y^t)$ in membrane p_1^+. After that rule a.1.1 is applied yielding string $\Delta(A_1^{x_1} \cdots A_k^{x_k+1} \cdots A_n^{x_n} Y^t)$ in membrane p_2^+. After that one of rules a.2.1 or a.2.1′ is applied; then rule a.1.2 yields one of strings $\Delta(zA_1^{x_1} \cdots A_k^{x_k+1} \cdots A_n^{x_n} Y^{t+1})$, $z \in \{q, s\}$, which is in the simulating form.

Now suppose that there is an instruction $p : (SUB(k), q, s) \in P$. Then the only applicable rule in Π is 3.2 which yields the string $\Delta(A_1^{x_1} \cdots A_n^{x_n} Y^t)$ in membrane p_1^-. Now if $x_k > 0$, then, due to the priority, rule e.1.1 will be applied followed by application of rules e.2.4, e.2.1 and e.1.3 which yields the string $\Delta(qA_1^{x_1} \cdots A_k^{x_k-1} \cdots A_n^{x_n} Y^{t'})$ that is in the simulating form. If $x_k = 0$, then rule e.1.2 will be applied (provided that all symbols Y were previously deleted by rule 3.3), followed by rules e.3.1 and e.1.3 which leads to the string $\Delta(sA_1^{x_1} \cdots A_n^{x_n})$ that is in the simulating form.

To show that membrane 3 is empty during the intermediate steps, we prove the following invariant:

Invariant 1 *During a successful computation, any visited membrane p_1^+ or p_1^- is visited an even number of times as follows: first a string coming from membrane 3 is sent to an inner membrane (p_2^+, p_2^- or p_2^0) and after that a string coming from an inner membrane is sent to membrane 3.*

Indeed, since there is only one string in the initial configuration, it is enough to follow only its evolution. Hence, a string may visit the node p_1^+ or p_1^- only if in the previous step symbol p was deleted by one of rules 3.1 or 3.2. If one of rules a.1.2 or e.1.3 is applied, then membrane 3 will contain a string of form $\Delta(A_1^{x_1} \cdots A_n^{x_n} Y^t)$ which cannot evolve anymore because all rules in membrane 3 imply the presence of a symbol from the set Q. Hence, the string is sent to an inner membrane. In the next step the string will return from the inner membrane by one of rules a.2.1, a.2.1′, e.2.1 or e.3.1 inserting a symbol from Q. If the string enters an inner membrane again, then it will be sent to a trap membrane (p_3^+, p_3^- or p_3^0) by rules deleting symbols from Q. Hence the only possibility is to go to membrane 3 (a string that visited membrane p_2^- will additionally use rule e.2.4).

For the second claim, it suffices to observe that the invariant above ensures that in membrane 3 only one symbol from Q can be present in the string.

The third claim holds since a string may move to membrane 2 if and only if the final label h of M appears in membrane 3. Then, the string is checked for the absence of symbols Y by rule 2.2 (note that symbols Y can be erased in membrane 3 by rule 3.3) and sent to the environment by rules 2.1 and 1.

By induction on the number of computational steps we obtain that Π simulates any computation in M. Claim 1 and 2 imply it is not possible to generate other strings and Claim 3 implies that the same result is obtained.

We remark that an empty string may be obtained during the proof. This string can still evolve using insertion rules. If we would like to forbid such evolutions, it suffices to use a new symbol, e.g., X, in the initial configuration, add new surrounding membrane and a rule that deletes X from it.

4 Small Contextual Insertion-Deletion P Systems

Although Theorem 5 shows that the systems from the previous section are quite powerful, they cannot generate RE without control on the place where a symbol is inserted. Once we allow a context in insertion and deletion rules, they can.

Theorem 6. $LSP(ins_1^{0,1} < del_1^{0,0}) = RE$.

Proof. We simulate a register machine with WRITE instructions. We implement this instruction as an ADD instruction, except the added symbol has to be inserted to the left of a special marker, deleted at the end, as follows:

- Replace any writing instruction $p : (WRITE(A), q, s)$, $A \in T$, of the machine by instructions $p : (ADD(A), q, s)$, considering output symbols A like new dummy registers. Construct the system Π as in Theorem 5.
- Change the initial string in membrane 3 to $q_0 M$;
- Replace rules $a.1.1$ $((\lambda, A, \lambda; in_{p_2^+})_a \in R_{p_1^+})$ by $(\lambda, A, M; in_{p_2^+})_a$ for $A \in T$;
- Surround membrane 1 by a new skin membrane s and add to it the following rule $R_s = \{(\lambda, M, \lambda; out)_e\}$.

It is easy to see that the above construction permits to correctly simulate the register machine with writing instructions.

Taking M in the left context yields the mirror language. Since RE is closed with respect to the mirror operation we get the following corollary:

Corollary 2. $LSP(ins_1^{1,0} < del_1^{0,0}) = RE$.

A similar result holds if contextual deleting operation is allowed.

Theorem 7. $LSP_*(ins_1^{0,0} < del_1^{1,0}) = RE$.

Proof. As in Theorem 6, we use the construction from Theorem 5. However, an additional membrane is needed to simulate the writing instructions.

We modify the construction of Theorem 5 as follows. Let Q_s be the set of labels of WRITE instructions of a register machine. We add the following substructures $\mu_{\langle ps \rangle}$ inside membrane 3 (shown in Figure 3):

$$\mu = [\,[\,[\prod_{p\in Q_+}\mu_{\langle p+\rangle}\prod_{p\in Q_-}\mu_{\langle p-\rangle}\prod_{p\in Q_s}\mu_{\langle ps\rangle}\,]_3\,]_2\,]_1, \text{ where}$$

$\mu_{\langle p+\rangle}, \mu_{\langle p-\rangle}$ are defined as in Theorem 5 and,

$$\mu_{\langle ps\rangle} = [\,[\,[\,[\,[\,]_{p_7^s}\,]_{p_4^s}[\,]_{p_6^s}\,]_{p_3^s}[\,]_{p_5^s}\,]_{p_2^s}\,]_{p_1^s}.$$

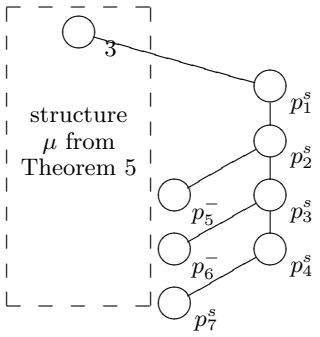

Fig. 3. Membrane structure for Theorem 7.

As in Theorem 5 the initial configuration contains a single string q_0 in region 3. The system contains sets of rules R_1, R_2, $R_{p_1^+}, R_{p_2^+}, R_{p_3^+}$, $R_{p_1^-}, R_{p_2^-}, R_{p_3^-}$, $R_{p_2^0}, R_{p_3^0}$ defined as in Theorem 5. There are also following additional rules for instructions $p : (WRITE(A), q)$ (the ruleset R_3' shall be added to R_3).

$$\begin{aligned}
R_3' = \{ &\quad 3.5 : (\lambda, p, \lambda; in_{p_1^s})_e \mid p \in Q_s\},\\
R_{p_1^s} = \{ &\quad w.1.1 : (\lambda, M, \lambda; in_{p_2^s})_a, && w.1.2 : (\lambda, M, \lambda; out)_e\},\\
R_{p_2^s} = \{ &\quad w.2.1 : (\lambda, M', \lambda; in_{p_3^s})_a, && w.2.2 : (\lambda, M', \lambda; out)_e\}\\
\cup \{ &\quad w.2.3 : (M, x, \lambda; in_{p_5^s})_e \mid x \in O\},\\
R_{p_3^s} = \{ &\quad w.3.1 : (\lambda, A, \lambda; in_{p_4^s})_a, && w.3.2 : (\lambda, Y, \lambda; out)_a\}\\
\cup \{ &\quad w.3.3 : (x, M, \lambda; in_{p_6^s})_e \mid x \in O \setminus \{M', q\}\},\\
R_{p_4^s} = \{ &\quad w.4.1 : (\lambda, q, \lambda; out)_a, && w.4.2 : (M', M, \lambda; in_{p_7^s})_e\},\\
R_{p_5^s} = \emptyset, &\quad R_{p_6^s} = \emptyset,\ R_{p_7^s} = \emptyset.
\end{aligned}$$

We simulate the WRITE instruction as follows. Suppose the configuration of register machine is $pA_1^{x_1}\cdots A_d^{x_d}$ and the word $a_1\cdots a_n$ is written on the output tape. The corresponding simulating string in Π will be of form $p\Delta w$, where $w = \Delta(A_1^{x_1}\cdots A_d^{x_d}Y^t) \sqcup\!\sqcup a_1\cdots a_n$, $t \geq 0$. After the deletion of the state symbol p, a marker M is inserted in the string by rule $w.1.1$. If M is not inserted at the right end of the string, in the next step rule $w.2.3$ is applicable and the string enters the trap membrane p_5^s. In the next step symbol M' is inserted in the string. If it is not inserted before M, then the string is sent to membrane p_6^s by rule $w.3.3$. Hence, at this moment the contents of membrane p_3^s is $wM'M$. If rule $w.3.2$ is used, then the string $Y \sqcup\!\sqcup w$ reaches membrane 3 and no rule is applicable anymore. Otherwise,

symbol A is inserted by rule $w.3.1$. If it is not between M' and M, then rule $w.4.2$ is applicable and the string enters membrane p_7^s. After that q is inserted between A and M, otherwise the trapping rule $w.3.3$ is applicable. At this moment, the configuration of the system consists of the string $w_t M' AqM$ in membrane p_3^s. Now if the rule $w.3.1$ is used, then the string is sent to the trap membrane by rule $w.4.1$. Otherwise, rule $w.3.2$ should be used followed by the application of rules $w.2.2$ and $w.1.2$, leading to string $Y \sqcup\!\sqcup wAq$ in membrane 3. Hence, the symbol A is appended at the end of the string. At the end of the computation, all symbols from $O - T$ are deleted and a word generated by M is obtained.

Since RE is closed with respect to the mirror operation we obtain:

Corollary 3. $LSP(ins_1^{0,0} < del_1^{0,1}) = RE$.

We remark that the contextual deletion was used only to check for erroneous evolutions. Therefore we can replace it by a context-free deletion of two symbols.

Theorem 8. $LSP_*(ins_1^{0,0} < del_2^{0,0}) = RE$.

Proof. We modify the proof of Theorem 7 as follows.

- Replace rules $w.2.3$ $((M, x, \lambda; in_{p_5^s})_e \in R_{p_2^s})$ by rules $(\lambda, Mx, \lambda; in_{p_5^s})_e$,
- Replace rules $w.3.3$ $((M, x, \lambda; in_{p_6^s})_e \in R_{p_3^s})$ by rules $(\lambda, xM, \lambda; in_{p_6^s})_e$,
- Replace rules $w.4.2$ $((M', M, \lambda; in_{p_7^s})_e \in R_{p_4^s})$ by rules $(\lambda, M'M, \lambda; in_{p_7^s})_e$.

The role of the new rules is the same as the role of the rules that were replaced. More exactly, the system checks whether two certain symbols are consecutive and if so, the string is blocked in a non-output region.

We mention that the counterpart of Theorem 8 obtained by interchanging parameters insertion and deletion rules is not true, see Theorem 2.

5 Conclusions

We showed several results concerning P systems with insertion and deletion rules of small size. Surprisingly, systems with context-free rules inserting and deleting only one symbol are quite powerful and generate $PsRE$ if the priority of deletion over insertion is used. From the language generation viewpoint such systems are not too powerful and no language specifying the order of symbols can be generated. To be able to generate more complicated languages we considered systems with one-symbol one-sided insertion or deletion contexts. In both cases we obtained that any recursively enumerable language can be generated. The same result holds if a context-free deletion of two symbols is allowed. The counterpart of the last result is not true, moreover Theorem 2 shows that the insertion of strings of an arbitrary size still cannot lead to generating languages like a^*b^*.

We also have considered one-symbol context-free insertion-deletion P systems without the priority relations and we showed that in terms of Parikh sets these

systems characterize the $PsMAT$ family. However, in terms of the generated language such systems are strictly included in MAT.

Most of results above were obtained using rules with target indicators. It is interesting to investigate the computational power of systems with non-specific target indicators *in* or *go*. Another open problem is to replace the priority relation by some other mechanism without decreasing the computational power.

Acknowledgments

The first author acknowledges the support of the Japan Society for the Promotion of Science, and the Grant-in-Aid, project 20·08364. The first, the third and the fourth authors acknowledge the support by the Science and Technology Center in Ukraine, project 4032. The third author gratefully acknowledges the support of the European Commission, project MolCIP, MIF1-CT-2006-021666. The second author acknowledges the support PIF program of University Rovira i Virgili, and project no. MTM2007-63422 from the Ministry of Science and Education of Spain.

References

1. A. Alhazov, R. Freund, A. Riscos-Núñez: One and two polarizations, membrane creation and objects complexity in P systems. *Proc. SYNASC'05*, IEEE Computer Society, 2005, 385–394.
2. A. Alhazov, R. Freund, Yu. Rogozhin: Computational power of symport/antiport: History, advances, and open problems. *Proc. WMC 2005*, Vienna, LNCS 3850, Springer, 2006, 1–30.
3. M. Daley, L. Kari, G. Gloor, R. Siromoney: Circular contextual insertions/deletions with applications to biomolecular computation. *Proc. of 6th Int. Symp. on String Processing and Information Retrieval*, SPIRE'99, Cancun, 47–54.
4. R. Freund, O.H. Ibarra, Gh. Păun, H.-C. Yen: Matrix languages, register machines, vector addition systems. *Third Brainstorming Week on Membrane Computing*, Sevilla, 2005, 155–168.
5. R. Freund, M. Oswald: GP systems with forbidding context. *Fundamenta Informaticae*, 49, 1–3 (2002), 81–102.
6. B.S. Galiukschov: Semicontextual grammars. *Matematika Logica i Matematika Linguistika*, Tallin University, 1981, 38–50 (in Russian).
7. S.A. Greibach: Remarks on blind and partially blind one-way multicounter machines. *Theoretical Computer Science*, 7 (1978), 311–324.
8. L. Kari: *On Insertion and Deletion in Formal Languages*. PhD Thesis, University of Turku, 1991.
9. L. Kari, Gh. Păun, G. Thierrin, S. Yu: At the crossroads of DNA computing and formal languages: Characterizing RE using insertion-deletion systems. *Proc. DNA Based Computers, 1997*, Philadelphia, 318–333.
10. L. Kari, G. Thierrin: Contextual insertion/deletion and computability. *Information and Computation*, 131, 1 (1996), 47–61.

11. A. Krassovitskiy, Yu. Rogozhin, S. Verlan: Further results on insertion-deletion systems with one-sided contexts. *Proc. LATA 2008*, LNCS 5196, Springer, 2008, 347–358.
12. A. Krassovitskiy, Yu. Rogozhin, S. Verlan: One-sided insertion and deletion: Traditional and P systems case. *Proc. CBM 2008*, Vienna, 53–64.
13. A. Krassovitskiy, Yu. Rogozhin, S. Verlan: Computational power of P systems with small size insertion and deletion rules. *Proc. CSP 2008*, Cork, 137–148.
14. S. Marcus: Contextual grammars. *Rev. Roum. Math. Pures Appl.*, 14 (1969), 1525–1534.
15. M. Margenstern, Gh. Păun, Yu. Rogozhin, S. Verlan: Context-free insertion-seletion Systems. *Theoretical Computer Science*, 330 (2005), 339–348.
16. C. Martin-Vide, Gh. Păun, A. Salomaa: Characterizations of recursively enumerable languages by means of insertion grammars. *Theoretical Computer Science*, 205, 1–2 (1998), 195–205.
17. A. Matveevici, Yu. Rogozhin, S. Verlan: Insertion-deletion systems with one-sided contexts. LNCS 4664, Springer, 2007, 205–217.
18. M.L.Minsky: *Computation. Finite and Infinite Machines*, Prentice Hall, Englewood Cliffs, NJ, 1967.
19. Gh. Păun: *Marcus Contextual Grammars*. Kluwer, Dordrecht, 1997.
20. Gh. Păun: *Membrane Computing. An Introduction*. Springer, Berlin, 2002.
21. Gh. Păun, G. Rozenberg, A. Salomaa: *DNA Computing. New Computing Paradigms*. Springer, Berlin, 1998.
22. G. Rozenberg, A. Salomaa, eds.: *Handbook of Formal Languages*. Springer, Berlin, 1997.
23. A. Takahara, T. Yokomori: On the computational power of insertion-deletion systems. *Proc. DNA Based Computers 2002*, Sapporo, LNCS 2568, Springer, 2003, 269–280.
24. S. Verlan: On minimal context-free insertion-deletion systems. *Journal of Automata, Languages and Combinatorics*, 12, 1-2 (2007), 317-328.
25. D.Wood: *Theory of Computation*. Harper and Row, New York, 1987.