Nat Comput (2014) 13:257–268
DOI 10.1007/s11047-013-9396-3

Generating and accepting P systems with minimal left and right insertion and deletion

Rudolf Freund · Yurii Rogozhin · Sergey Verlan

Published online: 17 January 2014

Abstract In this paper 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, both in the generating as well as in the accepting case.

Keywords Computational completeness · Deletion · Insertion · Matrix grammars · P systems

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) (Castellanos et al. 2001). Such networks are a special type of networks of language processors (Csuhaj-Varjú and Salomaa 1997) 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 (point mutations): *insertion*, *deletion*, and *substitution*. The corresponding systems are quite powerful and we refer to Dassow et al. (2011) for more details. The redistribution of the contents of a node is based on a regular condition, which is a very powerful operation. In accepting hybrid networks of evolutionary processors (AHNEP) as considered in Dassow and Manea (2010) and Margenstern et al. (2005a), this redistribution condition is replaced by random context conditions, and moreover, the set of operations is changed now including the operations of insertion and deletion at the extremities of the strings.

The operations of insertion and deletion at the extremities of a string can also be seen as a restricted case of the 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 Haussler (1982, 1983) and after that related insertion and deletion operations were investigated in Kari (1991) and Kari et al. (1997). 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 Galiukschov (1981) and Marcus (1969) and with a biological motivation in Benne (1993), Biegler et al. (2007), Kari et al. (1997) and Păun et al. (1998). 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

R. Freund (✉)
Faculty of Informatics, Vienna University of Technology, Favoritenstr. 9, 1040 Vienna, Austria
e-mail: rudi@emcc.at

Y. Rogozhin
Institute of Mathematics and Computer Science, Academy of Sciences of Moldova, Str. Academiei 5, 2028 Chişinău, Moldova
e-mail: rogozhin@math.md

S. Verlan
LACL, Département Informatique, Université Paris Est, 61, av. Général de Gaulle, 94010 Créteil, France
e-mail: verlan@univ-paris12.fr

descriptional complexity parameters sufficient to achieve this goal, we refer to Verlan (2010a, b) 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 (Freund et al. 2010; Ivanov and Verlan 2011; Petre and Verlan 2010). 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 Păun (2002) and Păun et al. (2010) and to the web page The P systems Web page (http://ppage.psystems.eu/) 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 (Margenstern et al. 2005b). When one of these parameters is decreased, this result is not true anymore (Verlan 2007); moreover, even the graph-controlled variant cannot achieve computational completeness (Krassovitskiy et al. 2011). This changes when a graph control with appearance checking is used (Alhazov et al. 2011a) or in the case of a random context control (Ivanov and Verlan 2011). In both variants, minimal operations (involving only one symbol) were considered, leading to *RE* (the family of recursively 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 Alhazov et al. (2011a, 2012), 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 paper, which is an extended version of our paper (Freund et al. 2012) presented at the conference Unconventional Computation and Natural Computation 2012 in Orléans, we continue these investigations by considering P systems with minimal left and right insertion and deletion, and we prove that computational completeness can be achieved even in this case, this time showing computational completeness for both the generating case and the accepting case. 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 prove that using additional minimal substitutions (substitutions of one symbol by another one) allows for reducing the height of the tree structure of the P system to the minimal possible size, i.e., to one. In addition, we show that we can even avoid the target *here* in the case of channel type P systems using minimal left and right insertion and deletion, where the applications of the rules can be interpreted as being carried out when objects (strings) are passing through a membrane, in the sense of molecules being modified when passing through a specific membrane channel from one membrane (region) to another one.

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 $\{a_1, \ldots, a_n\}$ 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*. Two string languages are considered to be equal if they differ at most in the empty string. For more details of

formal language theory the reader is referred to the monographs and handbooks in this area as Dassow and Păun (1989) and Rozenberg and Salomaa (1997).

We here consider string rewriting rules only working at the ends of a string (they can be seen as restricted variants of Post rewriting rules as already introduced by Emil Post in Post (1943)):

Simple Post rewriting rule
$P[u\$x/y\$v]$ with $u,x,y,v \in V^*$:
$P[u\$x/y\$v](uwx) = ywv$ for $w \in V^*$.

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

For a simple Post rewriting rule $P[u\$x/y\$v]$ we also write $u\$x \to y\$v$. A normal Post rewriting rule $P[x/y]$ is a special case of a Post rewriting rule $P[u\$x/y\$v]$ with $u = v = \lambda$ (we here consider the mirror version $P[\$x/y\$]$ of the normal form rules $P[u\$/\$v]$ as originally considered in Post (1943) for Post canonical systems; the variant we take in this paper has already been used several times for proving specific results in the area of P systems, e.g., see Freund et al. (2004). Left substitution $S_L[u/y]$ and right substitution $S_R[x/v]$ are special cases of simple Post rewriting rules as well, with $x = v = \lambda$ and $u = y = \lambda$, respectively.

If in a (left or right) substitution $S_L[x/y]$ or $S_R[x/y]$ x is empty, then we also 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 also 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, we assume that any of these operations considered in this paper—(left or right) insertion, deletion, and substitution—at least involves one symbol.

A *(string rewriting) grammar* G of type X is a construct (V,T,A,P) where V is a (finite) 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., $L(G) = \{v \in T^* \mid A \Longrightarrow_G^* v\}$. The *language accepted by* G is the set of all terminal strings deriving the axiom, i.e., $L(G) = \left\{v \in T^* \mid v \Longrightarrow_G^* A\right\}$. The family of languages generated (accepted) by grammars of type X is denoted by $\mathcal{L}(X)(\mathcal{L}_a(X))$.

Instead of a single axiom A we may also allow a finite set of axioms; in this case, we put an A in front of the type X for this variant of grammars thus obtaining the family of languages generated (accepted) by grammars of type X denoted by $\mathcal{L}(A\text{-}X)(\mathcal{L}_a(A\text{-}X))$.

In general, we write $S_L^{k,m}$ for a type of grammars using only substitution rules $S_L[x/y]$ with $|x| \leq k$ and $|y| \leq m$. In the same way, we define the type $S_R^{k,m}$ for a type of grammars using only substitution rules $S_R[x/y]$ with $|x| \leq k$ and $|y| \leq m$, 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 $\leq m$ on either side of a string. If, in addition, we also allow substitutions $S_L[x/y]$ and $S_R[x/y]$ 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 $D^1 I^1 S^{1,1}$ have a rather limited computational power. Indeed, we can show the following representation of languages generated or accepted by grammars of type $D^1 I^1 S^{1,1}$:

Theorem 1 *Every language $L \subseteq T^*$ in $\mathcal{L}(D^1 I^1 S^{1,1})$ 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 $D^1 I^1 S^{1,1}$ 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 over V of length at most $|A|$). 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 over V of length at most $|A|$, 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

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

which yields the desired form $T_l^* S T_r^*$ for describing $L(G)$. □

Corollary 1 $\mathcal{L}(A\text{-}D^1I^1S^{1,1}) = \mathcal{L}(A\text{-}I^1) = \mathcal{L}_a(A\text{-}D^1I^1S^{1,1}) = \mathcal{L}_a(A\text{-}D^1)$.

Proof The representation of languages in $\mathcal{L}(D^1I^1S^{1,1})$ elaborated in the preceding proof means that for the type $D^1I^1S^{1,1}$ we could forget minimal deletions and substitutions and instead consider finite subsets of axioms instead of a single axiom, i.e., we have proved that $\mathcal{L}(A\text{-}D^1I^1S^{1,1}) = \mathcal{L}(A\text{-}I^1)$.

Similar arguments as outlined for the generating case immediately show that

$$\mathcal{L}\big(A\text{-}D^1I^1S^{1,1}\big) = \mathcal{L}_a\big(A\text{-}D^1I^1S^{1,1}\big) = \mathcal{L}_a\big(A\text{-}D^1\big)$$

as well, because in the accepting case insertions have the same effect as deletions in the generating case and vice versa, i.e., using $D_\alpha[b], \alpha \in \{l, r\}$, in the accepting grammar means using $I_\alpha[b]$ in the corresponding generating grammar. □

2.1 Post systems

A *simple/normal Post system* is a grammar using only simple/normal Post rewriting rules, i.e., is a grammar of *type SPS / NPS*. 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, $A \in V \setminus T$ 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 $P[Z/\lambda]$.

Basic results concerning Post systems are folklore since many years, e.g., see Minsky (1967). For the accepting case we have the following characterizations of *RE*:

Theorem 2 *For every recursively enumerable language $L \subseteq T^*$ there exists a simple Post rewriting system $G, G = (V, T, A, P)$, accepting L, i.e., $\mathcal{L}_a(SPS) = RE$.*

As having an idea how the simulations of the actions of a Turing machine by an accepting grammar and a Post system work may help for a better understanding of the proofs elaborated in the following sections, we sketch a special proof for the following well-known result:

Theorem 3 *For every recursively enumerable language $L \subseteq T^*$ there exists a Post rewriting system in normal form $G', G' = (V', T, A', P')$, such that G' accepts $L\{Z\}$, where $Z \in V' \setminus T$.*

Proof Let $M = \big(Q, V, T, Z_0, B, \delta, q_0, q_f\big)$ be a Turing machine accepting L with Q being the set of states, V the tape alphabet, $T \subset V$ the terminal alphabet, $Z_0 \in V$ the left end marker, $B \in V$ the blank symbol, δ the transition function, q_0 and q_f the initial and final state, respectively. A configuration of M may be represented as a string from $\{Z_0\}V^*QV^*\{Z_1\}$ where Z_1 is the right end marker (a new symbol not in V) and the symbol to the left of the state symbol from Q is the current symbol to be read by M. Without loss of generality, we may assume that the transitions of M are either (i) rewriting a symbol, (ii) going one step to the left or (iii) going one step to the right on the tape. Moreover, we assume that, given the input $w \in T^*$, M accepts w if and only if it halts with the final configuration represented by $Z_0q_fZ_1$. For M, we now construct a grammar $G'' = (V'', T, A'', P'')$ accepting $\{Z_0q_0\}L\{Z_1\}$ as follows:

$V'' = Q \cup Q' \cup \{Z_1\} \cup V$, $A'' = Z_0q_f'$, and P'' contains the following grammar rules obtained by translating the transition rules of M:

- rewriting a symbol $X \in V \setminus \{Z_0\}$ to $Y \in V \setminus \{Z_0\}$ going from state q to p is simulated by the rule $Xq \rightarrow Yp$;
- going one step to the left is simulated by the rule $Xq \rightarrow pX, X \in V \setminus \{Z_0\}$;
- going one step to the right is simulated by the rule $qX \rightarrow Xp, X \in V$, in P''; intending to go to the right of the right end marker Z_1 can only happen when trying to simulate a rule $qB \rightarrow Bp, q \neq q_f$; the simulation then has to be carried out by inserting an additional blank symbol, i.e., by the rules $qZ_1 \rightarrow p'Z_1$ and $p' \rightarrow Bp$;

– the final configuration $Z_0q_fZ_1$ of M may be represented in G'' by any string $Z_0q_fB^nZ_1$ for $n \geq 0$; hence, we add the rules $q_fB \to q_f$ and $q_fZ_1 \to q'_f$.

Using the well-known technique of "simulate and rotate", from the accepting grammar $G'' = (V'', T, A'', P'')$ we can easily obtain a Post system in normal form $G' = (V', T, A', P')$ accepting $L\{Z\} : V' = V'' \cup Q \times V \cup V \times Q \cup \{Z, Z'\}$, $A'' = A' = Z_0q'_f$, and the rules in P' are obtained as follows: as we can only act on the right-hand side of a string with the result of the application of a rule being put to the left-hand side, we add the rotation rules $\$X \to X\$$ for all $X \in V'' \setminus \{q'_f\}$; the initial configuration is obtained by the rules $\$Z \to Z_1Z'\$$ and $\$Z' \to Z_0q_0\$$; a rewriting rule $Xq \to Yp$ is simulated by the rules $\$Xq \to (Y,p)\$$ and $\$(Y,p) \to Yp\$$; the rule $Xq \to pX$ is simulated by the rules $\$Xq \to (p,X)\$$ and $\$(p,X) \to pX\$$; the rule $qX \to Xp, X \in V$, is simulated by the rules $\$qX \to (X,p)\$$ and $\$(X,p) \to Xp\$$; the rule $qZ_1 \to p'Z_1$ is simulated by the rules $\$qZ_1 \to (p',Z_1)\$$ and $\$(p',Z_1) \to p'Z_1\$$ as well as $p' \to Bp$ by $\$p' \to Bp\$$; finally, the rule $q_fB \to q_f$ is simulated by $\$q_fB \to q_f\$$ and $q_fZ_1 \to q'_f$ by $\$q_fZ_1 \to q'_f\$$. As this construction shows, in the accepting case we do not need rules of the form $\$X \to \$$. □

For the proof of our main theorem we need the special Z-normal form; the following result is an immediate consequence of the proof given for Lemma 1 in Freund et al. (2004):

Theorem 4 *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., $\mathcal{L}(SPS) = \mathcal{L}(PSNF) = \mathcal{L}(PSZNF) = RE$.*

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 strings $w_i \in V^*$, $1 \leq i \leq n+1$, 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 5
$\mathcal{L}(D^2I^2\text{-}MAT_2) = \mathcal{L}(D^1I^1\text{-}MAT_3) = \mathcal{L}(PSNF) = RE$.

Proof From Theorem 4 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 $D^1I^1S^{1,1}$ 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 completeness.

2.3 P systems

We now introduce another variant to guide the derivations in a grammar using rules of those types introduced above, especially minimal 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 to be the nodes of the tree representing μ and to be uniquely labeled by labels from a set *Lab*; the number of membranes in μ usually is called the *degree* of μ (and the degree of Π itself, too);
– R is a set of rules of the form (h, r, tar) where $h \in Lab, r \in P$, and tar is called the *target (indicator)* and 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 membrane h_2 immediately outside membrane h_1; if $tar = in_{h_2}$, then the string is moved from membrane h_1 to the membrane h_2 immediately inside membrane h_1; if $tar = in$, then the string w_2 is sent to one of the membranes immediately inside membrane 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 then 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 families of languages are denoted by $\mathcal{L}(X\text{-}LsP)$ $(\mathcal{L}(X\text{-}LsP^{\langle n\rangle}))$. If even only the targets *in* and *out* are used, then the P system is called a *channel type P system*, as any change taking place in such a P system can be interpreted as only happening when an object (a string) passes through a membrane; the corresponding families of languages are denoted by $\mathcal{L}(X\text{-}LcP)(\mathcal{L}(X\text{-}LcP^{\langle n\rangle}))$.

For the accepting case, i_0 is the initial membrane where the axiom A together with the input $w \in T^*$ is put as wA at the beginning of a computation, and the input w is accepted if and only if there exists a halting computation from the initial configuration (wA,i_0). In contrast to the accepting variant of sequential grammars, in the case of P systems we do not care about the contents of the membranes at the end of a halting computation. By $\mathcal{L}_a(X\text{-}LP)$, $\mathcal{L}_a(X\text{-}LsP)$, and $\mathcal{L}_a(X\text{-}LcP)$ $(\mathcal{L}_a(X\text{-}LP^{\langle n\rangle})$, $\mathcal{L}_a(X\text{-}LsP^{\langle n\rangle})$, and $\mathcal{L}_a(X\text{-}LcP^{\langle n\rangle}))$ we then denote the families of languages accepted by P systems, simple P systems, and channel type P systems (of tree height at most n) using rules of type X.

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

$$G = (\{a,b\},\{a,b\},a,\{I_R[b],I_L[a]\}),$$
$$R = \{(1,I_R[b],in),(2,I_L[a],out)\}$$

The computations in Π start with a in membrane 1. In general, starting with a string $a^{n+1}b^n$, $n \geq 0$, in membrane 1, we may either add b on the right-hand side by the rule $(1,I_R[b],in)$, getting $a^{n+1}b^{n+1}$ as the terminal result in the elementary membrane 3 (a membrane is called *elementary* if and only if it contains no inner membrane) or else go into membrane 2, from there going out again into membrane 1 with adding another a thus obtaining $a^{n+2}b^{n+1}$. These considerations show that the language generated by Π is $\{a^{n+1}b^{n+1} \mid n \geq 0\}$.

The P system $\Pi' = (G, [_1[_2]_2[_3[_4[_5]_5]_4]_3]_1, R, 2)$ of type $D_R^1 D_L^1 I_R^1$ with

$$G = (\{a,b\},\{a,b\},\lambda,\{D_L[a],D_R[b],I_R[\#],D_R[\#]\}),$$
$$R = \{(2,D_L[a],out),(1,D_R[b],in),(3,D_L[a],in),(3,D_R[b],in)\}$$
$$\cup\{(1,I_R[\#],in),(2,I_R[\#],out),(3,D_R[\#],in)\}$$
$$\cup\{(4,I_R[\#],in),(5,D_R[\#],out)\}$$

accepts the same language: traveling between membranes 2 and 1 allows for deleting a symbol a on the left-hand side and a symbol b on the right-hand side. The last symbol b is deleted by going into membrane 3. The rules with the trap symbol # guarantee that an infinite loop, finally looping between membranes 4 and 5, is entered whenever the input was not of the form $a^{n+1}b^{n+1}$, $n \geq 0$.

Hence, in sum we have shown

$$\{a^{n+1}b^{n+1} \mid n \geq 0\} \in \mathcal{L}(I_R^1 I_L^1\text{-}LsP^{\langle 1\rangle}) \cap \mathcal{L}_a(D_R^1 D_L^1 I_R^1\text{-}LsP^{\langle 3\rangle}).$$

i.e., $\mathcal{L}(I_R^1 I_L^1\text{-}LsP^{\langle 1\rangle})$ as well as $\mathcal{L}_a(D_R^1 D_L^1 I_R^1\text{-}LsP^{\langle 3\rangle})$ contain a non-regular language. □

Example 3 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

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

The computations in Π start with aB in membrane 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 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 $(2,I_L[aa],out)$ 2^n times, finally obtaining $a^{2^{n+1}}B$. In sum we get $L(\Pi) = \{a^{2^n} \mid n \geq 0\}$ for the language generated by the P system Π; hence, we have shown that

$$\{a^{2^n} \mid n \geq 0\} \in \mathcal{L}(D_R^1 I_L^2\text{-}LP^{\langle 1\rangle}),$$

i.e., $\mathcal{L}\left(D_R^1 I_L^2\text{-}LP^{\langle 1\rangle}\right)$ contains a non-context-free language. □

3 Computational completeness of P systems with minimal 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, as the grammars considered there correspond to P systems with only one membrane, i.e., with tree height zero), of type $D_R^1 I_L^1 S_R^{1,1}$, i.e., using minimal left insertions and minimal right deletions and substitutions.

Theorem 6

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

Proof From Theorem 4 we know that $\mathcal{L}(PSZNF) = RE$, hence, we first show that for every Post system $G = (V, T, A, P)$ in Z-normal form we are able to construct an equivalent P system Π of type $D_R^1 I_L^1 S_R^{1,1}$. We assume that the rules in P are labeled 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 labeled by 1, and all the other membranes being elementary membranes inside the skin membrane being labeled by labels from

$$\begin{aligned} Lab' =& \{\#\} \cup Lab \\ & \cup \left\{\bar{h} \mid h : P[a_h/b_h c_h] \in P\right\} \cup \left\{\bar{h} \mid h : P[a_h b_h/c_h] \in P\right\}. \end{aligned}$$

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

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

the additional membrane # is used to trap all computations not leading to a terminal string in an infinite loop by the rules $\left(1, I_L[\#], in_\#\right)$ and $(\#, I_L[\#], out)$; for this purpose, the rule $(h, I_L[\#], out)$ is used in case of $h : P[a_h b_h/c_h]$, 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 $L(\Pi) = L(G)$.

For the accepting case, in a similar way we construct a P system Π' of type $D_R^1 I_L^1 S_R^{1,1}$ equivalent to a Post system $G = \left(V, T, Z_0 q_f', P\right)$ in normal form constructed for a given recursively enumerable language according to the proof outlined for Theorem 3. $\Pi' = (G'', \mu'', R', z)$ practically has the same ingredients as the P system Π described above, with two main exceptions: we now start a computation with the input word put into membrane z and instead of $(D_R[Z], in_z)$ take the rule $(I_R[Z], out)$; moreover, we take an additional membrane f to finish the simulation of an accepting derivation in G having lead to the final string $Z_0 q_f'$. According to the proof outlined for Theorem 3, we now may even omit the rules for simulating rules of the form $P[a_h/\lambda]$. Hence, for the labels of the membranes inside the skin membrane we take

$$\begin{aligned} Lab'' =& \{\#, f\} \cup Lab \\ & \cup \left\{\bar{h} \mid h : P[a_h/b_h c_h] \in P\right\} \cup \left\{\bar{h} \mid h : P[a_h b_h/c_h] \in P\right\}; \end{aligned}$$

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

$$\begin{aligned} &h : P[a_h b_h/c_h] : (1, D_R[b_h], in_h), \left(h, S_R\left[a_h/\vec{a}_h^{\,h}\right], out\right), (h, I_L[\#], out), \\ &\qquad \left(1, D_R\left[\vec{a}_h^{\,h}\right], in_{\bar{h}}\right), (\bar{h}, I_L[c_h], out); \\ &h : P[a_h/b_h c_h] : \left(1, S_R\left[a_h/\vec{a}_h^{\,h}\right], in_h\right), (h, I_L[c_h], out), \\ &\qquad \left(1, D_R\left[\vec{a}_h^{\,h}\right], in_{\bar{h}}\right), (\bar{h}, I_L[b_h], out); \\ &h : P[a_h/b_h] : (1, D_R[a_h], in_h), (h, I_L[b_h], out); \\ &z : P[\lambda/Z] : (I_R[Z], out); \end{aligned}$$

as well as $\left(1, I_L[\#], in_\#\right), (\#, I_L[\#], out)$, and $\left(D_R\left[q_f'\right], in_f\right)$ as the additional rules. □

Summarizing the results of Theorems 1 and 6, we get:

Corollary 2 *For all* $n \geq 1$,

$$\begin{aligned} \mathcal{L}\left(D^1 I^1 S^{1,1}\right) &= \mathcal{L}\left(D^1 I^1 S^{1,1}\text{-}LP^{\langle 0\rangle}\right) = \mathcal{L}_a\left(D^1 I^1 S^{1,1}\text{-}LP^{\langle 0\rangle}\right) \\ &\subset REG \subset \mathcal{L}\left(D_R^1 I_L^1 S_R^{1,1}\text{-}LP^{\langle n\rangle}\right) \\ &= \mathcal{L}_a\left(D_R^1 I_L^1 S_R^{1,1}\text{-}LP^{\langle n\rangle}\right) = RE. \end{aligned}$$

If we want to restrict ourselves to the simple targets *here*, *in*, *out* and as well to use only minimal insertions and deletions, then we have to use a more difficult proof technique than in the proof of Theorem 6.

Theorem 7 $\mathcal{L}\left(D^1I^1\text{-}LsP^{\langle 8\rangle}\right) = \mathcal{L}_a\left(D^1I^1\text{-}LsP^{\langle 8\rangle}\right) = RE.$

Proof In order to show the inclusion $RE \subseteq \mathcal{L}\left(D^1I^1\text{-}LsP^{\langle 8\rangle}\right)$, as in the proof of Theorem 6 we start from a Post system $G = (V, T, A, P)$ in Z-normal form with assuming the rules in P to be labeled 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 $\Pi, \Pi = (G', \mu, R, 1)$, 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 inner 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
$$\left[_{(h,1)}\left[_{(h,2)}\cdots\left[_{(h,8)}\right]_{(h,8)}\cdots\right]_{(h,2)}\right]_{(h,1)};$$
- 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
$$\left[_{(h,1)}\left[_{(h,2)}\cdots\left[_{(h,6)}\right]_{(h,6)}\cdots\right]_{(h,2)}\right]_{(h,1)};$$
- for getting the terminal results by erasing the symbol Z, 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 rules from P are accomplished by the procedures as shown in Tables 1, 2, 3, 4 and 5, 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 $(z, 3)$ by starting with a string $wZ, w \in T^*$, from the skin membrane (Table 1).

Tables 2, 3, 4 and 5 are to be interpreted in the same way as above; yet mostly 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

Table 1 Getting the terminal string $w \in T^*$

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

Table 2 Simulation of $h : P[ab/c]$

$(h, 8)$		$ScwDH$		$H-, F+$
$(h, 7)$	$+S$	$cwDH$	$ScwD$	$E+, F+$
$(h, 6)$	$+c$	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	

Table 3 Simulation of $h : P[a/bc]$

$(h, 8)$		$SbcwDH$		$H-, F+$
$(h, 7)$	$+S$	$bcwDH$	$SbcwD$	$E+, F+$
$(h, 6)$	$+b$	$cwDH$	$SbcwDE$	$M+, F+$
$(h, 5)$	$+c$	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	

Table 4 Simulation of $h : P[a/\lambda], a \neq Z$

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

Table 5 Simulation of $h : P[a/b]$

$(h,6)$		$SbwD$		$D-, F+$
$(h,5)$	$+S$	bwD	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	

membrane is the only one in the whole system which uses the target *here*, i.e., it has to be interpreted as $(1, D-, here)$.

From the descriptions given in Tables 2, 3, 4 and 5, 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 $L(\Pi) = L(G)$.

As in the proof of Theorem 6, the construction for an accepting P system Π' is only slightly different from what has already been discussed above:

The initial input string is put into membrane $(z,3)$ in the linear structure of 3 membranes $\left[_{(z,1)}\left[_{(z,2)}\left[_{(z,3)}\right]_{(z,3)}\right]_{(z,2)}\right]_{(z,1)}$, which allows us to insert a symbol Z on the right-hand side of the input string (Table 6).

Only the last two columns describe the computations to happen at the beginning, whereas whenever a string v gets back into membrane $(z,1)$ from membrane 1 by using the rule $K+$ there, we end up with introducing the trap symbol F.

Checking for the final string $Z_0q'_f$ is accomplished in the additional linear structure of 3 membranes $\left[_{(f,1)}\left[_{(f,2)}\left[_{(f,3)}\right]_{(f,3)}\right]_{(f,2)}\right]_{(f,1)}$ (Table 7).

For any $v \neq Z_0q'_f$ we return back to membrane 1 with vFF, thus finally ending up in an infinite loop. The remaining ingredients for Π' are exactly the same as for Π in the generating case, except that we need not take into account rules of the form $h : P[a/\lambda]$. Hence, we have shown $RE \subseteq \mathcal{L}_a\left(D^1I^1\text{-}LsP^{\langle 8\rangle}\right)$, too. □

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 7, yet the direct transformation of the construction given in the proof of this theorem would yield a lot of matrices with

Table 6 Getting the initial string wZ from $w \in T^*$ in $(z,3)$

$(z,3)$		$vZKFF$	w	$Z+, F-$
$(z,2)$	$F+$	$vZKF$	wZ	$D+, F+$
$(z,1)$	$F+$	vZK	wZD	$D-, F+$
1	$K+$	vZ	wZ	

Table 7 Checking for the final string $Z_0q'_f$

$(f,3)$		Z_0		
$(f,2)$	q'_f-	$Z_0q'_f$	v	$F+$
$(f,1)$	$K-$	$Z_0q'_fK$	vF	$F+$
1	$K+$	$Z_0q'_f$	vFF	

lengths more than 3, whereas the direct proof given in Theorem 5 only needed matrices of length at most 3.

In the construction given in the preceding proof, there was only one rule using the target *here*, but this could not be avoided at all when simulating normal Post rules not keeping the parity of symbols. In order to avoid this, we now use an encoding for the strings which instead of one symbol a uses two symbols $\hat{a}\bar{a}$.

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

Proof For any arbitrary alphabet Σ, let $g : \Sigma \to \hat{\Sigma}\bar{\Sigma}$ be the morphism defined by $g(a) = \hat{a}\bar{a}$ for $a \in \Sigma$. We first consider the accepting case, i.e., we want to show the inclusion $RE \subseteq \mathcal{L}_a\left(D^1I^1\text{-}LcP^{\langle 8\rangle}\right)$. The main idea for avoiding the target *here* now is to work on strings $g(w)$ instead of w, as the length of a string $g(w)$ is always even. As a first example, consider the Post rewriting rule $h : P[a/bc]$, which then extends to $g(h) : \$\hat{a}\bar{a} \to \hat{b}\bar{b}\hat{c}\bar{c}\$$. Yet any rule of the form $l : \$uv \to wxyz\$$ can be replaced by the rules $l' : \$uv \to z\$D'_l, l'' : \$D'_l \to xy\$D_l$, and $l''' : \$D_l \to w\$$. Whereas l''' already is a normal Post rewriting rule, it is quite easy to simulate these rules l' and l'' based on the tables explained in the proof of Theorem 7 (Tables 1, 2, 3, 4, 5, 6, 7). For example, simulating a rule of the form $\$a \to bc\$D_h$ corresponds to take the simulation table of the rule $\$a \to bc\$$ ending up with the symbol D_h on the right-hand-side of the string, i.e., in the table for the simulation of $h : P[a/bc]$ we replace D by the specific D_h and, of course, have no rule with target *here* working on D_h in membrane 1 (Table 8).

In the same way, the simulation of any rule of the form $\$ab \to c\$D_h$ corresponds to take the simulation table of the rule $\$ab \to c\$$ ending up with the symbol D_h on the right-hand-side of the string (Table 9).

The Post rewriting rule $h : P[ab/c]$ extends to $g(h) : \$\hat{a}\bar{a}\hat{b}\bar{b} \to \hat{c}\bar{c}\$$. Yet any rule of the form $l : \$uvwx \to yz\$$ can be replaced by the rules $l' : \$wx \to z\$D'_l, l'' :$

Table 8 Simulation of $h : \$a \to bc\$D_h$

$(h,8)$		$SbcwD_hH$		$H-, F+$
$(h,7)$	$+S$	$bcwD_hH$	$SbcwD_h$	$E+, F+$
$(h,6)$	$+b$	cwD_hH	$SbcwD_hE$	$M+, F+$
$(h,5)$	$+c$	wD_hH	$SbcwD_hEM$	$-S, F+$
$(h,4)$	$H+$	wD_h	$bcwD_hEM$	$M-, F+$
$(h,3)$	D_h+	w	$bcwD_hE$	$J+, F+$
$(h,2)$	$a-$	wa	$bcwD_hEJ$	$J-, F+$
$(h,1)$	$K-$	waK	$bcwD_hE$	$E-, F+$
1	$K+$	wa	$bcwD_h$	

Table 9 Simulation of $h : \$ab \to c\$D_h$

$(h,8)$		$ScwD_hH$		$H-, F+$
$(h,7)$	$+S$	cwD_hH	$ScwD_h$	$E+, F+$
$(h,6)$	$+c$	wD_hH	$ScwD_hE$	$M+, F+$
$(h,5)$	$H+$	wD_h	$ScwD_hEM$	$-S, F+$
$(h,4)$	D_h+	w	cwD_hEM	$M-, F+$
$(h,3)$	$a-$	wa	cwD_hE	$J+, F+$
$(h,2)$	$b-$	wab	cwD_hEJ	$J-, F+$
$(h,1)$	$K-$	$wabK$	cwD_hE	$E-, F+$
1	$K+$	wab	cwD_h	

Table 10 Simulation of $h : \$ab \to \$$

$(h,5)$		wD		$H+, F+$
$(h,4)$	$D+$	w	wDH	$J+, F+$
$(h,3)$	$a-$	wa	$wDHJ$	$J-, F+$
$(h,2)$	$b-$	wab	wDH	$H-, F+$
$(h,1)$	$K-$	$wabK$	wD	$D-, F+$
1	$K+$	wab	w	

$\$vD'_l \to z\$D_l$, and $l''' : \$uD_l \to \$$. Whereas l' and l'' are rules of the form $\$ab \to c\$D_h$, l'' is a new kind of rule, but easily to be simulated in the P system Π to be constructed following the constructions elaborated in the proof of Theorem 7 (Table 10).

The rule $h : P[a/\lambda]$ extends to $g(h) : \$\hat{a}\bar{a} \to \$$, which is of the form just described above. The rule $h : P[a/b]$ extends to $g(h) : \$\hat{a}\bar{a} \to \hat{b}\bar{b}\$$, which can be replaced by the rules $h' : \$\hat{a}\bar{a} \to \bar{b}\$D_h$ and $h'' : \$D_h \to \hat{b}\$$; both are of forms already described above.

The main challenge to finish the proof for the accepting case is to encode a given input string w into $g(w)$, as we need some other technical tricks to stay within the framework of linear membrane structures inside the skin membrane and to avoid the target *here*. A given recursively enumerable language L therefore is divided into two languages L_1 and L_2 containing exactly those strings from L which are of odd and even lengths, respectively. According to Theorem 3, for both L_1 and L_2 there exist Post systems in normal form G_1 and G_2 accepting $\{e\}L_1\{Z(1)\}$ and $L_2\{Z(2)\}$, where e is an additional (terminal) symbol.

Given an input string w, in the P system Π to be constructed for accepting $L_1 \cup L_2$, we start with the initialization procedure, with the input string being put into the input membrane $(I,4)$ (Table 11).

We then, from $wU\tilde{U}$, non-deterministically generate $\hat{U}wU_2$ for strings w of even lengths and $\hat{U}ewU_1$ for strings w of odd lengths (if this guess is wrong, no successful computation in Π will be possible). In the case of input strings of odd lengths, we have to insert a single symbol e at the beginning of w (as then ew is of even length). In sum, we have to simulate the Post rewriting rules $\$U\tilde{U} \to \hat{U}\$U_2$ and $\$U\tilde{U} \to \hat{U}e\$U_1$; the latter rule can be split into the sequence of rules $\$\tilde{U} \to \hat{U}e\$\tilde{U}_1$ and $\$\tilde{U}_1 \to \$U_1$, which itself can be simulated easily (Table 12).

The initial encoding now in principle works on strings of even lengths between $\hat{U}$ and $U_i, i \in \{1,2\}$, where for U_1 we have the terminal alphabet extended by the special symbol e. In order to avoid that the simulations in the P system Π of computations in G_1 and G_2 interfere, we assume the alphabets of G_1 and G_2 to be disjoint, i.e., even for terminal symbols a we have different copies $a(1)$ and $a(2)$. Hence, we have to simulate Post rewriting rules of the form $\$abU_i \to \hat{a}(i)\bar{a}(i)\hat{b}(i)\bar{b}(i)\$U_i$ and finally $\$\hat{U}U_i \to \$\hat{Z}(i)\ \bar{Z}(i)$. In sum, we then have obtained an encoded string $g_1(ewZ)$ or $g_2(wZ)$ where $g_i(a) = \hat{a}(i)\bar{a}(i)$. The rules $\$abU_i \to \hat{a}(i)\bar{a}(i)\hat{b}(i)\bar{b}(i)\$U_i$ can be split into a sequence of rules $\$bU_i \to \bar{b}(i)\$U_i(b), \$U_i(b) \to \bar{a}(i)\hat{b}(i)\$U'_i(a)$ (here, we non-deterministically guess which will be the next symbol a), and $\$aU'_i(a) \to \hat{a}(i)\$U_i$.

Table 11 Initialization for input w

$(I,4)$			w	$U+$
$(I,3)$			wU	$\tilde{U}+$
$(I,2)$			$wU\tilde{U}$	$D+$
$(I,1)$	vK	vK	$wU\tilde{U}D$	$D-, F+$
1	$K+$	v	$wU\tilde{U}$	

Table 12 Simulation of the rule $i : \$\tilde{U}_1 \to \$U_1$

$(i,6)$		SwU_1D		$H+, F+$
$(i,5)$	$D+$	SwU_1	SwU_1DH	$J+, F+$
$(i,4)$	$+S$	wU_1	SwU_1DHJ	$J-, F+$
$(i,3)$	U_1+	w	SwU_1DH	$S-, F+$
$(i,2)$	$\tilde{U}_1-$	$w\tilde{U}_1$	wU_1DH	$H-, F+$
$(i,1)$	$K-$	$w\tilde{U}_1K$	wU_1D	$D-, F+$
1	$K+$	$w\tilde{U}_1$	wU_1	

According to the simulation tables, the P system Π now can simulate the corresponding Post systems in normal form G_1 and G_2.

Checking for the final string corresponding to $Z_0q'_f$ means checking for the final string $\hat{Z}_0(i)\bar{Z}_0(i)\hat{q}'_f(i)\bar{q}'_f(i)$ with $i \in \{1,2\}$. As in the proof of Theorem 7 it is sufficient to check for the appearance of the last symbol (Table 13).

The last two columns show what happens if we enter membrane $(f,1)$ with a wrong string v.

Hence, in sum we have shown $RE \subseteq \mathcal{L}_a\big(D^1I^1\text{-}LcP^{\langle 8\rangle}\big)$.

In the generating case, again a given recursively enumerable language L is divided into two languages L_1 and L_2 containing exactly those strings from L which are of odd and even lengths, respectively. According to Theorem 4, for both L_1 and L_2 there exist Post systems in Z-normal form G_1 and G_2 generating $\{e\}L_1\{Z(1)\}$ and $L_2\{Z(2)\}$, where e is an additional (terminal) symbol. Again we assume the alphabets of G_1 and G_2 to be disjoint, i.e., even for terminal symbols a we have different copies $a(1)$ and $a(2)$. Let $G_i = (V_i, T_i, S_i, P_i)$, $i \in \{1,2\}$, with $S_i \in V_i \setminus T_i$. The P system Π we now describe simulates the computations in both Post systems G_1 and G_2. Π in principle has the same structure as the one already described in Theorem 7, and it starts with the initial string $\hat{S}\bar{S}$ in membrane 1, then simulating the Post rewriting rules $\$\hat{S}\bar{S} \to \hat{S}_i\bar{S}_i\$$ (as already shown above, these rules can be simulated by the rules $\$\hat{S}\bar{S} \to \bar{S}_i\$\hat{S}'_i$ and $\$\hat{S}'_i \to \hat{S}_i\$$). The simulations of the rules from G_1 and G_2 can be performed as already described above.

Finally, we end up with the strings $g_1(ew_1Z_1)$ and $g_2(w_2Z_2)$, respectively, with the strings ew_1 and w_2 being of even lengths; these strings $g_1(ew_1Z_1)$ and $g_2(w_2Z_2)$ now have to be decoded to the terminal results w_1 and w_2, respectively: As soon as the simulation of G_1 or G_2 has yielded $g_1(ew_1Z_1)$ or $g_2(w_2Z_2)$, respectively, we start the decoding procedure with simulating the Post rewriting rule $\$\hat{Z}_i\bar{Z}_i \to \tilde{Z}_i\$Z_i$. Then, the decoding for two consecutive symbols is executed, i.e., we have to simulate the rule $\$\hat{a}(i)\bar{a}(i)\hat{b}(i)\bar{b}(i)Z \to ab\$Z$, which can be decomposed into the sequence of rules $\$\hat{b}(i)\bar{b}(i)Z_i \to \$Z_i(b), \$\bar{a}(i)Z_i(b) \to b\$Z'_i(a)$, as well as $\$\hat{a}(i)Z'_i(a) \to a\$Z_i$ for $(i,a) \neq (1,e)$; the rule $\$\hat{b}(i)\bar{b}(i)Z_i \to \$Z_i(b)$ can be simulated easily, too (Table 14).

Table 13 Checking for the final string corresponding to $Z_0q'_f$

$(f,3)$		$\hat{Z}_0(i)\bar{Z}_0(i)\hat{q}'_f(i)$		
$(f,2)$	$\bar{q}'_f(1)-, \bar{q}'_f(2)-$	$\hat{Z}_0(i)\bar{Z}_0(i)\hat{q}'_f(i)\bar{q}'_f(i)$	v	$F+$
$(f,1)$	$K-$	$\hat{Z}_0(i)\bar{Z}_0(i)\hat{q}'_f(i)\bar{q}'_f(i)K$	vF	$F+$
1	$K+$	$\hat{Z}_0(i)\bar{Z}_0(i)\hat{q}'_f(i)\bar{q}'_f(i)$	vFF	

Table 14 Simulation of $h : \$\hat{b}(i)\bar{b}(i)Z_i \to \$Z_i(b)$

$(h,8)$		$SwZ_i(b)D$		$H+, F+$
$(h,7)$	$D+$	$SwZ_i(b)$	$SwZ_i(b)DH$	$E+, F+$
$(h,6)$	$+S$	$wZ_i(b)$	$SwZ_i(b)DHE$	$E-, F+$
$(h,5)$	$Z_i(b)+$	w	$SwZ_i(b)DH$	$-S, F+$
$(h,4)$	$\hat{b}(i)-$	$w\hat{b}(i)$	$wZ_i(b)DH$	$H-, F+$
$(h,3)$	$\bar{b}(i)-$	$w\hat{b}(i)\bar{b}(i)$	$wZ_i(b)D$	$J+, F+$
$(h,2)$	Z_i-	$w\hat{b}(i)\bar{b}(i)Z_i$	$wZ_i(b)DJ$	$J-, F+$
$(h,1)$	$K-$	$w\hat{b}(i)\bar{b}(i)Z_iK$	$wZ_i(b)D$	$D-, F+$
1	$K+$	$w\hat{b}(i)\bar{b}(i)Z_i$	$wZ_i(b)$	

Table 15 Extracting the terminal string w_2 from $w_2\tilde{Z}_2Z_2$

$((f,2),4)$		w_2	
$((f,2),3)$	$\tilde{Z}_2-$	$w_2\tilde{Z}_2$	$F+$
$((f,2),2)$	Z_2-	$w_2\tilde{Z}_2Z_2$	$F+$
$((f,2),1)$	$K-$	$w_2\tilde{Z}_2Z_2K$	$F+$
1	$K+$	$w_2\tilde{Z}_2Z_2$	

Table 16 Extracting the terminal string w_1 string from $w_1\tilde{Z}_1\hat{e}Z'_1(e)$

$((f,1),5)$		w_1	
$((f,1),4)$	$\tilde{Z}_1-$	$w_1\tilde{Z}_1$	$F+$
$((f,1),3)$	$\hat{e}-$	$w_1\tilde{Z}_1\hat{e}$	$F+$
$((f,1),2)$	$Z'_1(e)-$	$w_1\tilde{Z}_1\hat{e}Z'_1(e)$	$F+$
$((f,1),1)$	$K-$	$w_1\tilde{Z}_1\hat{e}Z'_1(e)K$	$F+$
1	$K+$	$w_1\tilde{Z}_1\hat{e}Z'_1(e)$	

At the end, we reach one of the situations $w_2\tilde{Z}_2Z_2$ or $w_1\tilde{Z}_1\hat{e}Z'_1(e)$, which leads to the following final procedures in Π (Tables 15, 16)

Terminal strings of even lengths appear in membrane $((f,2),4)$, terminal strings of odd lengths arrive in membrane $((f,1),5)$.

Hence, we have also shown $RE \subseteq \mathcal{L}\big(D^1I^1\text{-}LcP^{\langle 8\rangle}\big)$. □

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 Alhazov et al.

(2011b). Computational completeness was not only proved for the generating case, but for the accepting case as well. In Theorem 6 we have shown that using minimal left insertion, minimal right deletion, and, in addition, minimal right substitution (substitution of one symbol by another one on the right-hand side of a string) we can obtain a P system with the height of the tree structure being the minimum 1, at the same time even avoiding the use of the target *here*. The P systems constructed in the other proofs showing computational completeness with only using the operations of minimal left and right insertion and deletion, had rather large tree height; it remains an open question to reduce this complexity parameter.

References

Alhazov A, Krassovitskiy A, Rogozhin Yu, Verlan S (2011a) P systems with minimal insertion and deletion. Theor Comput Sci 412(1–2):136–144

Alhazov A, Krassovitskiy A, Rogozhin Yu, Verlan S (2011b) P systems with insertion and deletion exo-operations. Fundam Inform 110(1–4):13–28

Alhazov A, Krassovitskiy A, Rogozhin Yu (2012) Circular post machines and P systems with exo-insertion and deletion. In: Gheorghe M et al (eds) Membrane computing—12th international conference, CMC 2011, Fontainebleau, France, August 23–26, 2011, revised selected papers. Lecture notes in computer science 7184. Springer, New York, pp 73–86

Benne R (1993) RNA editing: the alteration of protein coding sequences of RNA. Ellis Horwood, Chichester

Biegler F, Burrell MJ, Daley M (2007) Regulated RNA rewriting: modelling RNA editing with guided insertion. Theor Comput Sci 387(2):103–112

Castellanos J, Martín-Vide C, Mitrana V, Sempere JM (2001) Solving NP-complete problems with networks of evolutionary processors. In: Mira J, Prieto AG (eds) IWANN 2001. Lecture notes in computer science 2084. Springer, New York, pp 621–628

Csuhaj-Varjú E, Salomaa A (1997) Networks of parallel language processors. In: Păun Gh, Salomaa A (eds) New trends in formal languages. Lecture notes in computer science 1218. Springer, New York, pp 299–318

Dassow J, Manea F (2010) Accepting hybrid networks of evolutionary processors with special topologies and small communication. In: McQuillan I, Pighizzini G (eds) Proceedings of the 12th workshop on descriptional complexity of formal systems, EPTCS 31, pp 68–77

Dassow J, Păun Gh (1989) Regulated rewriting in formal language theory. Springer, New York

Dassow J, Manea F, Truthe B (2011) On normal forms for networks of evolutionary processors. In: Calude CS, Kari J, Petre I, Rozenberg G (eds) Proceedings of the unconventional computation 10th international conference, UC 2011, Turku, Finland, June 6–10, 2011. Lecture notes in computer science 6714, pp 89–100

Freund R, Oswald M, Păun A (2004) Gemmating P systems are computationally complete with four membranes. In: Ilie L, Wotschke D (eds) Pre-proceedings DCFS 2004. The University of Western Ontario, Rep. No. 619, pp 191–203

Freund R, Kogler M, Rogozhin Yu, Verlan S (2010) Graph-controlled insertion–deletion systems. In: McQuillan I, Pighizzini G (eds) Proceedings of the 12th workshop on descriptional complexity of formal systems, EPTCS 31, pp 88–98

Freund R, Rogozhin Yu, Verlan S (2012) P systems with minimal left and right insertion and deletion. In: Durand-Lose J, Jonoska N (eds) Unconventional computation and natural computation, 11th international conference, UCNC 2012, Orleans, France, September 3–7, 2012. Lecture notes in computer science 7445. Springer, New York, pp 82–93

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

Haussler D (1982) Insertion and iterated insertion as operations on formal languages. PhD thesis, University of Colorado at Boulder, Boulder

Haussler D (1983) Insertion languages. Inf Sci 31(1):77–89

Ivanov S, Verlan S (2011) Random context and semi-conditional insertion–deletion systems. arXiv, CoRR abs/1112.5947

Kari L (1991) On insertion and deletion in formal languages. PhD thesis, University of Turku, Turku

Kari L, Păun Gh, Thierrin G, Yu S (1997) At the crossroads of DNA computing and formal languages: characterizing RE using insertion–deletion systems. In: Rubin H, Wood DH (eds) DNA based computers III. Proceedings of the 3rd DIMACS workshop on DNA based computing, University of Pennsylvania, Philadelphia. DIMACS series in discrete mathematics and theoretical computer science 48, pp 318–333

Krassovitskiy A, Rogozhin Yu, Verlan S (2011) Computational power of insertion–deletion (P) systems with rules of size two. Nat Comput 10(2):835–852

Marcus S (1969) Contextual grammars. Rev Roum Math Pures Appl 14:1525–1534

Margenstern M, Mitrana V, Pérez-Jiménez M (2005a) Accepting hybrid networks of evolutionary systems. In: Ferretti C, Mauri G, Zandron C (eds) DNA computing: 10th international workshop on DNA computing, DNA10, Milan, Italy, June 7–10, 2004. Revised selected papers. Lecture notes in computer science 3384. Springer, New York, pp 235–246

Margenstern M, Păun Gh, Rogozhin Yu, Verlan S (2005b) Context-free insertion–deletion systems. Theor Comput Sci 330(2):339–348

Minsky ML (1967) Computation: finite and infinite machines. Prentice Hall, Englewood Cliffs

Păun Gh (2002) Membrane computing. An introduction. Springer, New York

Păun Gh, Rozenberg G, Salomaa A (eds) (1998) DNA computing: new computing paradigms. Springer, New York

Păun Gh, Rozenberg G, Salomaa A (2010) The Oxford handbook of membrane computing. Oxford University Press, Oxford

Petre I, Verlan S (2010) Matrix insertion–deletion systems. arXiv, CoRR abs/1012.5248

Post EL (1943) Formal reductions of the general combinatorial decision problem. Am J Math 65(2):197–215

Rozenberg G, Salomaa A (1997) Handbook of formal languages, 3 vols. Springer, New York

Verlan S (2007) On minimal context-free insertion–deletion systems. J Autom Lang Comb 12(1–2):317–328

Verlan S (2010a) Recent developments on insertion–deletion systems. Comput Sci J Moldova 18(2):210–245

Verlan S (2010b) Study of language-theoretic computational paradigms inspired by biology. Habilitation thesis, University of Paris Est, Paris