Array Insertion and Deletion P Systems

Henning Fernau[1], Rudolf Freund[2], Sergiu Ivanov[3], Markus L. Schmid[1], and K.G. Subramanian[4]

[1] Fachbereich 4 – Abteilung Informatikwissenschaften, Universität Trier
D-54296 Trier, Germany
Email: {fernau,MSchmid}@uni-trier.de
[2] Technische Universität Wien, Institut für Computersprachen
Favoritenstr. 9, A-1040 Wien, Austria
Email: rudi@emcc.at
[3] Laboratoire d'Algorithmique, Complexité et Logique, Université Paris Est
Créteil Val de Marne, 61, Av. Gén. de Gaulle, 94010 Créteil, France
Email: sergiu.ivanov@u-pec.fr
[4] School of Computer Sciences, Universiti Sains Malaysia, 11800 Penang, Malaysia
Email: kgsmani1948@yahoo.com

Abstract. We consider the (d-dimensional) array counterpart of string insertion and deletion grammars and use the operations of array insertion and deletion in the framework of P systems where the applicability of the rules depends on the membrane region. In this paper, we especially focus on examples of two-dimensional array insertion and deletion P systems and show that we can already obtain computational completeness using such P systems with a membrane structure of tree height of at most two and only the targets *here*, *in*, and *out*.

1 Introduction

In the string case, the insertion operation was first considered in [13, 14] and after that related insertion and deletion operations were investigated, e.g., in [15, 16]. Backed by linguistic motivation, checking of insertion contexts was considered in [17]. These *contextual* grammars start from a set of strings (*axioms*), and new strings are obtained by using rules of the form (s, c), where s and c are strings to be interpreted as inserting c in the context of s, either only at the ends of strings (*external* case, [17]) or in the *interior* of strings ([20]). The fundamental difference between contextual grammars and Chomsky grammars is that in contextual grammars we do not *rewrite* symbols, but we only *adjoin* symbols to the current string, i.e., contextual grammars are pure grammars. Hence, among the variants of these grammars as, for example, considered in [3–5, 21, 22, 18], the variant where we can retain only the set of strings produced by blocked derivations, i. e., derivations which cannot be continued, is of special importance. This corresponds to the maximal mode of derivation (called t-mode) in cooperating grammar systems (see [1]) as well as to the way results in P systems are obtained by halting computations; we refer the reader to [19, 23] and to the web page [24] for more details on P systems.

With the length of the contexts and/or of the inserted and deleted strings being big enough, the insertion-deletion closure of a finite language leads to computational completeness. There are numerous results establishing the descriptional complexity parameters sufficient to achieve this goal; for an overview of this area we refer to [28, 27]. In [12] we have shown that we can also obtain computational completeness with using only insertions and deletions of just one symbol at the ends of a string using the regulating framework of P systems, where the applications of the rules depend on the membrane region.

The contextual style of generating strings was extended to d-dimensional arrays in a natural way (see [11]): a contextual array rule is a pair (s, c) of two arrays to be interpreted as inserting the new subarray c in the context of the array s provided that the positions where to put c are not yet occupied by a non-blank symbol. With retaining only the arrays produced in maximal derivations, interesting languages of two-dimensional arrays can be generated. In [8], contextual array rules in P systems are considered. A contextual array rule (s, c) can be interpreted as array insertion rule; by inverting the meaning of this operation, we get an array deletion rule (s, c) deleting the subarray c in the relative context of the subarray s.

In this paper, we exhibit some illustrative examples of P systems with (two-dimensional) array insertion rules (corresponding to contextual array rules). The main result of the paper exhibits computational completeness of (two-dimensional) array insertion and deletion P systems.

2 Definitions and Examples

The set of integers is denoted by $\mathbb{Z}$, the set of non-negative integers by $\mathbb{N}$. An *alphabet* V is a finite non-empty set of abstract *symbols*. Given V, the free monoid generated by V under the operation of concatenation is denoted by V^*; the elements of V^* are called strings, and the *empty string* is denoted by λ; $V^* \setminus \{\lambda\}$ is denoted by V^+. The family of recursively enumerable string languages is denoted by RE. For more details of formal language theory the reader is referred to the monographs and handbooks in this area such as [7] and [26].

2.1 A General Model for Sequential Grammars

In order to be able to introduce the concept of membrane systems (P systems) for various types of objects, we first define a general model ([10]) of a grammar generating a set of terminal objects by derivations where in each derivation step exactly one rule is applied (sequential derivation mode) to exactly one object.

A *(sequential) grammar* G is a construct $(O, O_T, w, P, \Longrightarrow_G)$ where O is a set of *objects*, $O_T \subseteq O$ is a set of *terminal objects*, $w \in O$ is the *axiom (start object)*, P is a finite set of *rules*, and $\Longrightarrow_G \subseteq O \times O$ is the *derivation relation* of G. We assume that each of the rules $p \in P$ induces a relation $\Longrightarrow_p \subseteq O \times O$ with respect to $\Longrightarrow_G$ fulfilling at least the following conditions: (i) for each object $x \in O$, $(x, y) \in \Longrightarrow_p$ for only finitely many objects $y \in O$; (ii) there exists a finitely described mechanism (as, for example, a Turing machine) which, given an object $x \in O$, computes all objects $y \in O$ such that $(x, y) \in \Longrightarrow_p$. A rule $p \in P$ is called *applicable* to an object $x \in O$ if and only if there exists at least one object $y \in O$ 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 $\overset{*}{\Longrightarrow}_G$.

In the following we shall consider different types of grammars depending on the components of G, especially on the rules in P; these may define a special type X of grammars which then will be called *grammars of type* X.

Usually, the *language generated by* G (in the $*$-mode) is the set of all terminal objects (we also assume $v \in O_T$ to be decidable for every $v \in O$) derivable from the axiom, i.e., $L_* (G) = \left\{ v \in O_T \mid w \overset{*}{\Longrightarrow}_G v \right\}$. The *language generated by* G *in the* t*-mode* is the set of all terminal objects derivable from the axiom in a halting computation, i.e., $L_t (G) = \left\{ v \in O_T \mid \left(w \overset{*}{\Longrightarrow}_G v \right) \wedge \nexists z \left(v \Longrightarrow_G z \right) \right\}$. The family of languages generated by grammars of type X in the derivation mode δ, $\delta \in \{*, t\}$, is denoted by $\mathcal{L}_\delta (X)$..

Let $G = (O, O_T, w, P, \Longrightarrow_G)$ be a grammar of type X. If for every G of type X we have $O_T = O$, then X is called a *pure* type, otherwise it is called *extended*.

2.2 String grammars

In the general notion as defined above, a *string grammar* G_S is represented as $\left((N \cup T)^*, T^*, w, P, \Longrightarrow_P \right)$ where N is the alphabet of *non-terminal symbols*, T is the alphabet of *terminal symbols*, $N \cap T = \emptyset$, $w \in (N \cup T)^+$ is the *axiom*, P is a finite set of *string rewriting rules*, and the derivation relation $\Longrightarrow_{G_S}$ is the classic one for string grammars defined over $V^* \times V^*$, with $V := N \cup T$. As classic types of string grammars we consider string grammars with arbitrary rules of the form $u \to v$ with $u, v \in V^*$ and context-free rules of the form $A \to v$ with $A \in N$ and $v \in V^*$. The corresponding types of grammars are denoted by ARB and CF, thus yielding the families of languages $\mathcal{L}(ARB)$ and $\mathcal{L}(CF)$, i.e., the family of recursively enumerable languages RE and the family of context-free languages, respectively.

In [12], left and right insertions and deletions of strings were considered; the corresponding types of grammars using rules inserting strings of length at most k and deleting strings of length at most m are denoted by $D^m I^k$.

2.3 Array grammars

We now introduce the basic notions for d-dimensional arrays and array grammars in a similar way as in [9, 11]. Let $d \in \mathbb{N}$; then a *d-dimensional array* $\mathcal{A}$ over an alphabet V is a function $\mathcal{A} : \mathbb{Z}^d \to V \cup \{\#\}$, where $\mathit{shape}(\mathcal{A}) = \left\{ v \in \mathbb{Z}^d \mid \mathcal{A}(v) \neq \# \right\}$ is finite and $\# \notin V$ is called the *background* or *blank symbol*. We usually write $\mathcal{A} = \{(v, \mathcal{A}(v)) \mid v \in \mathit{shape}(\mathcal{A})\}$. The set of all d-dimensional arrays over V is denoted by V^{*d}. The *empty array* in V^{*d} with empty shape is denoted by Λ_d. Moreover, we define $V^{+d} = V^{*d} \setminus \{\Lambda_d\}$. Let $v \in \mathbb{Z}^d$, $v = (v_1, \dots, v_d)$; the norm of v is defined as $\|v\| = \max\{|v_i| : 1 \leq i \leq d\}$; the *translation* $\tau_v : \mathbb{Z}^d \to \mathbb{Z}^d$ is defined by $\tau_v(w) = w + v$ for all $w \in \mathbb{Z}^d$. For any array $\mathcal{A} \in V^{*d}$ we define $\tau_v(\mathcal{A})$, the corresponding d-dimensional array translated by v, by $(\tau_v(\mathcal{A}))(w) = \mathcal{A}(w - v)$ for all $w \in \mathbb{Z}^d$. The vector $(0, \dots, 0) \in \mathbb{Z}^d$ is denoted by Ω_d.

Usually[2, 25, 29] arrays are regarded as equivalence classes of arrays with respect to linear translations, i. e., only the relative positions of the symbols different from # in the plane are taken into account: the equivalence class $[\mathcal{A}]$ of an array $\mathcal{A} \in V^{*d}$ is defined by

$$[\mathcal{A}] = \left\{\mathcal{B} \in V^{*d} \mid \mathcal{B} = \tau_v(\mathcal{A}) \text{ for some } v \in \mathbb{Z}^d\right\}.$$

The set of all equivalence classes of d-dimensional arrays over V with respect to linear translations is denoted by $\left[V^{*d}\right]$ etc.

A *d-dimensional array grammar* G_A is represented as

$$\left(\left[(N \cup T)^{*d}\right], \left[T^{*d}\right], [\mathcal{A}_0], P, \Longrightarrow_{G_A}\right)$$

where N is the alphabet of *non-terminal symbols*, T is the alphabet of *terminal symbols*, $N \cap T = \emptyset$, $\mathcal{A}_0 \in (N \cup T)^{*d}$ is the *start array*, P is a finite set of *d-dimensional array rules* over V, $V := N \cup T$, and $\Longrightarrow_{G_A} \subseteq \left[(N \cup T)^{*d}\right] \times \left[(N \cup T)^{*d}\right]$ is the derivation relation induced by the array rules in P.

A "classical" *d-dimensional array rule* p over V is a triple $(W, \mathcal{A}_1, \mathcal{A}_2)$ where $W \subseteq \mathbb{Z}^d$ is a finite set and $\mathcal{A}_1$ and $\mathcal{A}_2$ are mappings from W to $V \cup \{\#\}$. In the following, we shall also write $\mathcal{A}_1 \to \mathcal{A}_2$, because W is implicitly given by the finite arrays $\mathcal{A}_1, \mathcal{A}_2$. We say that the array $\mathcal{C}_2 \in V^{*d}$ is *directly derivable* from the array $\mathcal{C}_1 \in V^{+d}$ by $(W, \mathcal{A}_1, \mathcal{A}_2)$ if and only if there exists a vector $v \in \mathbb{Z}^d$ such that $\mathcal{C}_1(w) = \mathcal{C}_2(w)$ for all $w \in \mathbb{Z}^d - \tau_v(W)$ as well as $\mathcal{C}_1(w) = \mathcal{A}_1(\tau_{-v}(w))$ and $\mathcal{C}_2(w) = \mathcal{A}_2(\tau_{-v}(w))$ for all $w \in \tau_v(W)$, i. e., the subarray of $\mathcal{C}_1$ corresponding to $\mathcal{A}_1$ is replaced by $\mathcal{A}_2$, thus yielding $\mathcal{C}_2$; we also write $\mathcal{C}_1 \Longrightarrow_p \mathcal{C}_2$. Moreover we say that the array $\mathcal{B}_2 \in \left[V^{*d}\right]$ is *directly derivable* from the array $\mathcal{B}_1 \in \left[V^{+d}\right]$ by the d-dimensional array production $(W, \mathcal{A}_1, \mathcal{A}_2)$ if and only if there exist $\mathcal{C}_1 \in \mathcal{B}_1$ and $\mathcal{C}_2 \in \mathcal{B}_2$ such that $\mathcal{C}_1 \Longrightarrow_p \mathcal{C}_2$; we also write $\mathcal{B}_1 \Longrightarrow_p \mathcal{B}_2$.

A d-dimensional array rule $p = (W, \mathcal{A}_1, \mathcal{A}_2)$ in P is called *monotonic* if $\mathit{shape}(\mathcal{A}_1) \subseteq \mathit{shape}(\mathcal{A}_2)$ and *#-context-free* if $\mathit{shape}(\mathcal{A}_1) = \{\Omega_d\}$; if it is *#-context-free* and, moreover, $\mathit{shape}(\mathcal{A}_2) = W$, then p is called *context-free.* A d-dimensional array grammar is said to be of type X, $X \in \{d\text{-}ARBA, d\text{-}MONA, d\text{-}\#\text{-}CFA, d\text{-}CFA\}$ if every array rule in P is of the corresponding type, the corresponding families of array languages of equivalence classes of d-dimensional arrays by d-dimensional array grammars are denoted by $\mathcal{L}_*(X)$. These families form a Chomsky-like hierarchy, i.e., $\mathcal{L}_*(d\text{-}CFA) \subsetneqq \mathcal{L}_*(d\text{-}MONA) \subsetneqq \mathcal{L}_*(d\text{-}ARBA)$ and $\mathcal{L}_*(d\text{-}CFA) \subsetneqq \mathcal{L}_*(d\text{-}\#\text{-}CFA) \subsetneqq \mathcal{L}_*(d\text{-}ARBA)$. Two d-dimensional arrays $\mathcal{A}$ and $\mathcal{B}$ in $\left[V^{*d}\right]$ are called *shape-equivalent* if and only if $\mathit{shape}(\mathcal{A}) = \mathit{shape}(\mathcal{B})$.

Two d-dimensional array languages L_1 and L_2 from $\left[V^{*d}\right]$ are called shape-equivalent if and only if $\{\mathit{shape}(\mathcal{A}) \mid \mathcal{A} \in L_1\} = \{\mathit{shape}(\mathcal{B}) \mid \mathcal{B} \in L_2\}$.

2.4 Contextual, Insertion and Deletion Array Rules

A *d-dimensional contextual array rule* (see [11]) over the alphabet V is a pair of finite d-dimensional arrays $((W_1,\mathcal{A}_1),(W_2,\mathcal{A}_2))$ where $W_1 \cap W_2 = \emptyset$ and $\mathit{shape}(\mathcal{A}_1) \cup \mathit{shape}(\mathcal{A}_2) \neq \emptyset$. The effect of this contextual rule is the same as of the array rewriting rule $(W_1 \cup W_2, \mathcal{A}_1, \mathcal{A}_1 \cup \mathcal{A}_2)$, i.e., in the context of $\mathcal{A}_1$ we insert $\mathcal{A}_2$. Hence, such an array rule $((W_1,\mathcal{A}_1),(W_2,\mathcal{A}_2))$ can also be called an *array insertion rule*, and then we write $I((W_1,\mathcal{A}_1),(W_2,\mathcal{A}_2))$; if $\mathit{shape}(\mathcal{A}_i) = W_i$, $i \in \{1,2\}$, we simply write $I(\mathcal{A}_1,\mathcal{A}_2)$. Yet we may also interpret the pair $((W_1,\mathcal{A}_1),(W_2,\mathcal{A}_2))$ as having the effect of the array rewriting rule $\mathcal{A}_1 \cup \mathcal{A}_2 \to \mathcal{A}_1$, i.e., in the context of $\mathcal{A}_1$ we delete $\mathcal{A}_2$; in this case, we write $D((W_1,\mathcal{A}_1),(W_2,\mathcal{A}_2))$ or $D(\mathcal{A}_1,\mathcal{A}_2)$.

Let G_A be a d-dimensional array grammar $\left(\left[V^{*d}\right],\left[T^{*d}\right],[\mathcal{A}_0],P,\Longrightarrow_{G_A}\right)$ with P containing array insertion and deletion rules. Then we can consider the array languages $L_*(G_A)$ and $L_t(G_A)$ generated by G_A in the modes $*$ and t, respectively; the corresponding families of array languages are denoted by $\mathcal{L}_\delta(d\text{-}DIA)$, $\delta \in \{*,t\}$; if only array insertion (i.e., contextual) rules are used, we have the case of pure grammars, and we also write $\mathcal{L}_\delta(d\text{-}CA)$. For interesting relations between the families of array languages $\mathcal{L}_*(d\text{-}CA)$ and $\mathcal{L}_t(d\text{-}CA)$ as well as $\mathcal{L}_*(d\text{-}\#\text{-}CFA)$ and $\mathcal{L}_*(d\text{-}CFA)$ we refer the reader to [11].

As a first example we illustrate that the generative power of contextual array grammars can exceed that of context-free array grammars (also see [8], [11]):

Example 1. Consider the set R_H of hollow rectangles with arbitrary side lengths $p,q \geq 3$ (over the singleton alphabet $\{a\}$). By extending arguments used for similar problems in [6], it is easy to see that there cannot exist a grammar of type $d\text{-}CFA$ generating an array language which is shape-equivalent to R_H; on the other hand, the following grammar of type $2\text{-}CA$ yields such an array language in the t-mode, i.e., $\mathit{shape}(L_t(G_1)) = \mathit{shape}(R_H)$: $G_1 = \left(\{a,b\}^{*2},\{a,b\}^{*2},P,\mathcal{A}_0,\Longrightarrow_{G_1}\right)$ with $\mathcal{A}_0 = \begin{array}{cc} a & \\ a & a \end{array}$; for a graphic representation of the rules in P, we use the convention that the symbols from the two arrays $\mathcal{A}_1,\mathcal{A}_2$ in the contextual array production $I(\mathcal{A}_1,\mathcal{A}_2)$ are shown in one array and those from $\mathcal{A}_1$ are marked by a box frame:

$$p_1 = \begin{array}{c} a \\ \boxed{a} \\ \boxed{a} \end{array},\ p_2 = \begin{array}{ccc} \boxed{a} & \boxed{a} & a \end{array},\ p_3 = \begin{array}{cc} b & b \\ \boxed{a} & \\ \boxed{a} & \end{array},\ p_4 = \begin{array}{ccc} & & b \\ \boxed{a} & \boxed{a} & b \end{array},$$

$$p_5 = \begin{array}{ccc} \boxed{b} & \boxed{b} & b \end{array},\ p_6 = \begin{array}{c} b \\ \boxed{b} \\ \boxed{b} \end{array},\ p_7 = \begin{array}{ccc} \boxed{b} & \boxed{b} & a \\ & & \boxed{b} \\ & & \boxed{b} \end{array}.$$

Starting from the axiom $\mathcal{A}_0$, we can go up using the rule p_1 $p-3$ times and go to the right using the rule p_2 $q-3$ times, where $p,q \geq 3$ can be chosen arbtirarily. Then we turn to the right from the vertical line by once using the rule

p_3 and turn up from the horizontal line by once using the rule p_4, respectively; in both cases symbols b are appended to the growing lines of symbols a, whereafter the rules p_1, p_2, p_3, and p_4 cannot be applied anymore. The rules p_5 and p_6 then complete the upper and the right edge of the rectangle. The derivation only halts if the rule p_7 is applied at the end. □

The following example shows how we can generate an array language of coated filled squares with a specified middle point, which will be an essential part in the proof of our main theorem in Section 4:

Example 2. The contextual array grammar

$$G_2 = \left(\left\{ \bar{S}, E, Q \right\}^{*2}, \left\{ \bar{S}, E, Q \right\}^{*2}, P, \mathcal{A}_0, \Longrightarrow_{G_2} \right)$$

generates an array language of squares filled by the symbol E, coated by a layer of symbols Q, with the central position being marked by the symbol $\bar{S}$:

$$\mathcal{A}_0 = \begin{array}{ccccc} & E & E & E & E \\ E & E & E & E & E \\ E & E & \bar{S} & E & E \\ E & E & E & E & E \\ E & E & E & E & E \end{array} \overset{t}{\Longrightarrow}_{G_2} \begin{array}{ccccccccc} Q & Q & \cdots & Q & \cdots & Q & Q \\ Q & E & \cdots & E & \cdots & E & Q \\ \vdots & \vdots & & \vdots & & \vdots & \vdots \\ Q & E & \cdots & \bar{S} & \cdots & E & Q \\ \vdots & \vdots & & \vdots & & \vdots & \vdots \\ Q & E & \cdots & E & \cdots & E & Q \\ Q & Q & \cdots & Q & \cdots & Q & Q \end{array}$$

The final arrays are constructed in such a way that, starting from the axiom, layer by layer, another layer of symbols E is added, by applying the rules p_0 to p_7; the final layer of symbols Q is added by using the rules q_0 to q_8. In sum, P contains the following contextual rules:

$$p_0 = \begin{array}{cc} E & E \\ E & \boxed{E} \\ \boxed{E} & \boxed{E} \end{array}, \; q_0 = \begin{array}{cc} Q & Q \\ E & \boxed{E} \\ \boxed{E} & \boxed{E} \end{array}, \; p_1 = \begin{array}{ccc} \boxed{E} & E & \\ \boxed{E} & \boxed{E} & \boxed{E} \end{array}, \; q_1 = \begin{array}{ccc} \boxed{Q} & Q & \\ \boxed{E} & \boxed{E} & \boxed{E} \end{array},$$

$$p_2 = \begin{array}{ccc} \boxed{E} & E & E \\ \boxed{E} & \boxed{E} & E \end{array}, \; q_2 = \begin{array}{ccc} \boxed{Q} & Q & Q \\ \boxed{E} & \boxed{E} & Q \end{array}, \; p_3 = \begin{array}{cc} \boxed{E} & \boxed{E} \\ \boxed{E} & E \\ \boxed{E} & \end{array}, \; q_3 = \begin{array}{cc} \boxed{E} & \boxed{Q} \\ \boxed{E} & Q \\ \boxed{E} & \end{array},$$

$$p_4 = \begin{array}{cc} \boxed{E} & \boxed{E} \\ \boxed{E} & E \\ E & E \end{array}, \; q_4 = \begin{array}{cc} \boxed{E} & \boxed{Q} \\ \boxed{E} & Q \\ Q & Q \end{array}, \; p_5 = \begin{array}{ccc} \boxed{E} & \boxed{E} & \boxed{E} \\ & E & \boxed{E} \end{array}, \; q_5 = \begin{array}{ccc} \boxed{E} & \boxed{E} & \boxed{E} \\ & Q & \boxed{Q} \end{array},$$

$$p_6 = \begin{array}{ccc} E & \boxed{E} & \boxed{E} \\ E & E & \boxed{E} \end{array}, \; q_6 = \begin{array}{ccc} Q & \boxed{E} & \boxed{E} \\ Q & Q & \boxed{Q} \end{array}, \; p_7 := \begin{array}{cc} & \boxed{E} \\ E & \boxed{E} \\ \boxed{E} & \boxed{E} \end{array}, \; q_7 = \begin{array}{cc} Q & \boxed{E} \\ \boxed{Q} & \boxed{E} \end{array}, \text{ and}$$

$$q_8 = \begin{array}{cc} Q & \boxed{Q} \\ \boxed{Q} & \boxed{E} \end{array}.$$

A derivation in G_2 halts if and only if we end up with applying the rule q_8, thus finishing the coating layer of symbols Q. □

3 (Sequential) P Systems

A *(sequential) P system of type X with tree height n* is a construct $\Pi = (G, \mu, R, i_0)$ where $G = (O, O_T, A, P, \Longrightarrow_G)$ is a sequential grammar 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 labelled by labels from a set Lab;
- R is a set of rules of the form (h, r, tar) where $h \in Lab$, $r \in P$, and tar, called the *target indicator*, is taken from the set $\{here, in, out\} \cup \{in_h \mid h \in Lab\}$; 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 object during a computation of the P system, a configuration of Π can be described by a pair (w, h) where w is the current object (e.g., string or array) and h is the label of the membrane currently containing the object 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 object w_2 is sent to the region h_2 immediately outside membrane h_1; if $tar = in_{h_2}$, then the object is moved from region h_1 to the region h_2 immediately inside region h_1; if $tar = in$, then the object w_2 is sent to one of the regions immediately inside region h_1.

A sequence of transitions between configurations of Π, starting from the initial configuration (A, i_0), is called a *computation* of Π. A *halting computation* is a computation ending with a configuration (w, h) such that no rule from R_h can be applied to w anymore; (w, h) is called the result of this halting computation if $w \in O_T$. As the language generated by Π we consider $L_t(\Pi)$ which consists of all terminal objects from O_T being results of a halting computation in Π.

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

In the string case (see [12]), every language $L \subseteq T^*$ in $\mathcal{L}_*\left(D^1 I^1\right)$ can be written in the form $T_l^* S T_r^*$ where $T_l, T_r \subseteq T$ and S is a finite subset of T^*. Using the regulating mechanism of P systems, we get $\left\{a^{2^n} \mid n \geq 0\right\} \in \mathcal{L}_t\left(D^1 I^2\text{-}LP^{\langle 1 \rangle}\right)$ and even obtain computational completeness:

Theorem 1. *(see [12])* $\mathcal{L}_t\left(D^1 I^1\text{-}LsP^{\langle 8 \rangle}\right) = RE$.

One-dimensional arrays can also be interpreted as strings; left/right insertion (deletion) of a symbol a corresponds to taking the set containing all rules $I\left(a\,\boxed{b}\right)/I\left(\boxed{b}\,a\right)$ $(D\left(a\,\boxed{b}\right)/D\left(\boxed{b}\,a\right))$ for all b; hence, from Theorem 1, we immediately infer the following result, which with respect to the tree height of the simple P systems will be improved considerably in Section 4:

Corollary 1. $\mathcal{L}_t\left(1\text{-}DIA\text{-}LsP^{\langle 8\rangle}\right) = \mathcal{L}_*\left(1\text{-}ARBA\right)$.

For the array case we now restrict ourselves to the 2-dimensional case. First we go back to Example 1 and show how we can take advantage of having different contextual array rules to be applied in different membranes:

Example 3. Consider the contextual array grammar G_1 from Example 1 and the P system $\Pi_1 = (G_1, [_0\, [_1\,]_1\, [_2\,]_2\,]_0, R, 0)$ with R containing the rules $(0, p_1, in_1)$, $(1, p_2, out)$, $(0, p_3, in_2)$, $(2, p_4, out)$, $(0, p_5, in_1)$, $(1, p_6, out)$ and $(0, p_7, in_2)$. By synchronizing the growth of the left and the lower edge of the rectangle, we guarantee that the resulting rectangle finally appearing in membrane 2 is a square. Hence, we have shown that with $L_t(\Pi_1)$ in $\mathcal{L}_t\left(2\text{-}CA\text{-}LP^{\langle 1\rangle}\right)$ we can find an array language which is shape-equivalent to the set of hollow squares. □

4 Computational Completeness of Array Insertion and Deletion P Systems

We now show our main result that any recursively enumerable 2-dimensional array language can be generated by an array insertion and deletion P system which only uses the targets *here*, *in*, and *out* and whose membrane structure has only tree height 2.

Theorem 2. $\mathcal{L}_t\left(2\text{-}DIA\text{-}LsP^{\langle 2\rangle}\right) = \mathcal{L}_*\left(2\text{-}ARBA\right)$.

Proof. The main idea of the proof is to construct the simple P system Π of type 2-DIA with a membrane structure of height two generating a recursively enumerable 2-dimensional array language L_A given by a special grammar G_A of type 2-$ARBA$ in such a way that we first generate the coated squares described in Example 2 and then simulate the rules of the 2-dimensional array grammar G_A inside this square; finally, the superfluous symbols E and Q have to be erased to obtain the terminal array.

Now let $G_A = \left(\left[(N \cup T)^{*d}\right], \left[T^{*d}\right], \left[\mathcal{A}_0\right], P, \Longrightarrow_{G_A}\right)$ be an array grammar of type 2-$ARBA$ generating L_A. In order to make the simulation in Π easier, without loss of generality, we may make some assumptions on the forms of the array rules in P: First of all, we may assume that the array rules are in a kind of Chomsky normal form (e.g., compare [9]), i.e., only of the following forms: $A \to B$ for $A \in N$ and $B \in N \cup T \cup \{\#\}$ as well as $AvD \to BvC$ with $\|v\| = 1$, $A, B, C \in N \cup T$, and $D \in N \cup T \cup \{\#\}$ (we should like to emphasize that usually A, B, C, D in the array rule $AvD \to BvC$ would not be allowed to be terminal symbols, too); in a more formal way, the rule $AvD \to BvC$ represents the rule $(W, \mathcal{A}_1, \mathcal{A}_2)$ with $W = \{\Omega_d, v\}$, $\mathcal{A}_1 = \{(\Omega_d, A), (v, D)\}$, and $\mathcal{A}_2 = \{(\Omega_d, B), (v, C)\}$. As these rules in fact are simulated in Π with the symbol E representing the blank symbol $\#$, a rule $Av\# \to BvC$ now corresponds to a rule $AvE \to BvC$. Moreover, a rule $A \to B$ for $A \in N$ and $B \in N \cup T$ can be replaced by the set of all rules $AvD \to BvD$ for all $D \in N \cup T \cup \{E\}$ and $v \in \mathbb{Z}^d$ with $\|v\| = 1$, and $A \to \#$ can be replaced by the set of all rules $AvD \to EvD$ for all $D \in N \cup T \cup \{E\}$ and $v \in \mathbb{Z}^d$ with $\|v\| = 1$.

After these replacements described above, in the P system Π we now only have to deal with rules of the form $AvD \to BvC$ with $\|v\| = 1$ as well as $A, B, C, D \in N \cup T \cup \{E\}$. Yet in order to obtain a P system Π with the required features, we make another assumption for the rules to be simulated: any intermediate array obtained during a derivation contains exactly one symbol marked with a bar; as we only have to deal with sequential systems where at each moment exactly one rule is going to be applied, this does not restrict the generative power of the system as long as we can guarantee that the marking can be moved to any place within the current array. Instead of a rule $AvD \to BvC$ we therefore take the corresponding rule $\bar{A}vD \to Bv\bar{C}$; moreover, to move the bar from one position in the current array to another position, we add all rules $\bar{A}vC \to Av\bar{C}$ for all $A, C \in N \cup T \cup \{E\}$ and $v \in \mathbb{Z}^d$ with $\|v\| = 1$. We collect all these rules obtained in the way described so far in a set of array rules P' and assume them to be uniquely labelled by labels from a set of labels Lab', i.e., $P' = \left\{ l : \bar{A}_l v D_l \to B_l v \bar{C}_l \mid l \in Lab' \right\}$.

After all these preparatory steps we now are able to construct the simple P system Π with array insertion and deletion rules:

$$\Pi = \left(G, \left[_0 \left[_{I_1} \left[_{I_2} \right]_{I_2} \right]_{I_1} \cdots \left[_{l_1} \left[_{l_2} \right]_{l_2} \right]_{l_1} \cdots \left[_{F_1} \left[_{F_2} \right]_{F_2} \right]_{F_1} \right]_0, R, I_2\right)$$

with I_1 and I_2 being the membranes for generating the initial squares, F_1 and F_2 are the membranes to extract the final terminal arrays in halting computations, and l_1 and l_2 for all $l \in Lab'$ are the membranes to simulate the corresponding array rule from P' labelled by l. The components of the underlying array grammar G can easily be collected from the description of the rules in R as described below.

We start with the initial array $\mathcal{A}_0$ from Example 2 and take all rules $(I_2, I(r), here)$ with all rules $r \in \{p_i, q_i \mid 0 \leq i \leq 7\}$ taken as array insertion rules; instead of the array insertion rule q_8 we now take the rule q_8' instead and $(I_2, I(q_8'), out)$, where q_8' introduces the new control symbols K and K_I.Using $(I_1, D(q_9), out)$ we move the initial square out into the skin membrane.

$$q_8' = \begin{array}{cc} K & K_I \\ Q & \boxed{Q} \\ \boxed{Q} & \boxed{E} \end{array}, \quad q_9 = \begin{array}{cc} \boxed{K} & K_I \\ \boxed{Q} & \boxed{Q} \end{array}$$

To be able to simulate a derivation from G_A for a specific terminal array, the workspace in this initial square has to be large enough, but as we can generate such squares with arbitrary size, such an initial array can be generated for any terminal array in $L_*(G_A)$.

An array rule from $P' = \left\{ l : \bar{A}_l v D_l \to B_l v \bar{C}_l \mid l \in Lab' \right\}$ is simulated by applying the following sequence of array insertion and deletion rules in the membranes l_1 and l_2, which send the array twice the path from the skin membrane to membrane l_2 via membrane l_1 and back to the skin membrane:

$$\left(0, I\left(\boxed{K} K_l\right), in\right), \left(l_1, D\left(\boxed{\bar{A}_l} v D_l\right), in\right),$$
$$\left(l_2, I\left(\boxed{\bar{A}_l} v \bar{D}_l^{(l)}\right), out\right), \left(l_1, D\left(\boxed{\bar{D}_l^{(l)}} (-v)\, \bar{A}_l\right), out\right),$$
$$\left(0, I\left(\boxed{\bar{D}_l^{(l)}} (-v)\, B_l\right), in\right), \left(l_1, D\left(\boxed{B_l} v \bar{D}_l^{(l)}\right), in\right),$$
$$\left(l_2, D\left(\boxed{K} K_l\right), out\right), \left(l_1, I\left(\boxed{B_l} v \bar{C}_l\right), out\right).$$

Whenever reaching the skin membrane, the current array contains exactly one barred symbol. If we reach any of the membranes l_1 and/or l_2 with the wrong symbols (which implies that none of the rules listed above is applicable), we introduce the trap symbol F by the rules $\left(m, I\left(F\boxed{K}\right), out\right)$ and $\left(m, I\left(F\boxed{F}\right), out\right)$ for $m \in \{l_1, l_2 \mid l \in Lab' \cup \{I\}\}$; as soon as F has been introduced once, with $\left(0, I\left(F\boxed{F}\right), in\right)$ we can guarantee that the computation in Π will never stop.

As soon as we have obtained an array representing a terminal array, the corresponding array computed in Π is moved into membrane F_1 by the rule $(0, D(K), in)$ (for any X, $D(X)$ / $I(K)$ just means deleting/inserting X without taking care of the context). In membrane F_1, all superfluous symbols E and Q as well as the marked blank symbol $\bar{E}$ (without loss of generality we may assume that at the end of the simulation of a derivation from G_A in Π the marked symbol is $\bar{E}$) are erased by using the rules $(F_1, D(X), here)$ with $X \in \left\{E, \bar{E}, Q\right\}$. The computation in Π halts with yielding a terminal array in membrane F_1 if and only if no other non-terminal symbols have occurred in the array we have moved into F_1; in the case that non-terminal symbols occur, we start an infinite loop between membrane F_1 and membrane F_2 by introducing the trap symbol F: $(F_1, D(X), in)$ for $X \notin T \cup \left\{E, \bar{E}, Q\right\}$ and $(F_2, I(F), out)$.

As can be seen from the description of the rules in Π, we can simulate all terminal derivations in G_A by suitable computations in Π, and a teminal array $\mathcal{A}$ is obtained as the result of a halting computation (always in membrane F_1) if and only if $\mathcal{A} \in L_*(G_A)$, hence we conclude $L_t(\Pi) = L_*(G_A)$. □

5 Conclusion

In this paper, we have extended the notions of insertion and deletion from the string case to the case of d-dimensional arrays. Array insertion grammars have already been considered as contextual array grammars in [11], whereas the inverse interpretation of a contextual array rule as a deletion rule has newly been introduced here. Moreover, we have also introduced P systems using these array insertion and deletion rules, thus continuing the research on P systems with left and right insertion and deletion of strings, see [12].

In the main part of our paper, we have restricted ourselves to exhibit examples of 2-dimensional array languages that can be generated by array insertion (contextual array) grammars and P systems using array insertion rules as well as to show that array insertion and deletion P systems are computationally complete.

As can be seen from the proof of our main result, Theorem 2, the second part of the proof showing how to simulate array rules of the form $\bar{A}vD \to Bv\bar{C}$ is not restricted to the 2-dimensional case and can directly be taken over to the d-dimensional case for arbitrary d; hence, for $d > 2$, the main challenge is to generate d-dimensional cuboids of symbols E coated by a layer of symbols Q, with the central position marked by the start symbol $\bar{S}$. With regulating mechanims such as matrix or programmed grammars without the feature of appearance checking, this challenge so far has turned out to be intractable for $d > 2$ when using #-context-free array rules, e.g., see [9]; when using array insertion rules, this problem seems to become solvable. Moreover, we would also like to avoid the target *here* in the simple P systems using array insertion and deletion rules constructed in Theorem 2, as with avoiding the target *here*, the applications of the rules could be interpreted as being carried out when passing a membrane, in the sense of molecules passing a specific membrane channel from one region to another one. We shall return to these questions and related ones in an extended version of this paper.

References

1. E. Csuhaj-Varjù, J. Dassow, J. Kelemen, and Gh. Păun, *Grammar Systems. A Grammatical Approach to Distribution and Cooperation*, Gordon and Breach, London, 1994.
2. C. R. Cook and P. S.-P. Wang, A Chomsky hierarchy of isotonic array grammars and languages, *Computer Graphics and Image Processing* **8** (1978), pp. 144–152.
3. A. Ehrenfeucht, Gh. Păun, and G. Rozenberg, On representing recursively enumerable languages by internal contextual languages, *Theoretical Computer Science* **205**, 1–2 (1998), pp. 61–83.
4. A. Ehrenfeucht, A. Mateescu, Gh. Păun, G. Rozenberg, and A. Salomaa, On representing RE languages by one-sided internal contextual languages, *Acta Cybernetica* **12**, 3 (1996), pp. 217–233.
5. A. Ehrenfeucht, Gh. Păun, and G. Rozenberg, The linear landscape of external contextual languages, *Acta Informatica* **35**, 6 (1996), pp. 571–593.
6. J. Dassow, R. Freund, and Gh. Păun, Cooperating array grammar systems, *International Journal of Pattern Recognition and Artificial Intelligence* **9**, 6 (1995), pp. 1029–1053.
7. J. Dassow, Gh. Păun: *Regulated Rewriting in Formal Language Theory.* Springer, 1989.

8. H. Fernau, R. Freund, M.L. Schmid, K.G. Subramanian, and P. Wiederhold, Contextual Array Grammars and Array P Systems, *to be submitted*.
9. R. Freund, Control mechanisms on #-context-free array grammars, in Gh. Păun, Ed., *Mathematical Aspects of Natural and Formal Languages*, World Scientific, Singapore, 1994, pp. 97–137.
10. R. Freund, M. Kogler, and M. Oswald, A General Framework for Regulated Rewriting Based on the Applicability of Rules, in J. Kelemen and A. Kelemenová, Eds., *Computation, Cooperation, and Life - Essays Dedicated to Gheorghe Păun on the Occasion of His 60th Birthday*, Lecture Notes in Computer Science **6610**, Springer, 2011, pp. 35-53.
11. R. Freund, Gh. Păun, and G. Rozenberg, Contextual array grammars, in K.G. Subramanian, K. Rangarajan, and M. Mukund, Eds., *Formal Models, Languages and Applications*, Series in Machine Perception and Artificial Intelligence **66**, World Scientific, 2007, pp. 112–136.
12. R. Freund, Yu. Rogozhin, and S. Verlan, P systems with minimal left and right insertion and deletion, in J. Durand-Lose and N. Jonoska, Eds., *Unconventional Computation and Natural Computation - 11th International Conference, UCNC 2012*, Orléans, France, September 3-7, Lecture Notes in Computer Science **7445**, Springer, 2012, pp. 82–93.
13. D. Haussler, *Insertion and Iterated Insertion as Operations on Formal Languages.* PhD thesis, Univ. of Colorado at Boulder, 1982.
14. D. Haussler, Insertion languages, *Information Sciences* **31,** 1 (1983), pp. 77–89.
15. L. Kari, *On Insertion and Deletion in Formal Languages*, PhD thesis, University of Turku, 1991.
16. L. Kari, Gh. Păun, G. Thierrin, S. Yu, At the crossroads of DNA computing and formal languages: Characterizing RE using insertion-deletion systems, in *Proc. of 3rd DIMACS Workshop on DNA Based Computing,* Philadelphia, 1997, pp. 318–333.
17. S. Marcus, Contextual grammars, *Rev. Roum. Math. Pures Appl.* **14** (1969), pp. 1525–1534.
18. Gh. Păun, *Marcus Contextual Grammars*, Kluwer, Dordrecht, 1997.
19. Gh. Păun, *Membrane Computing. An Introduction,* Springer, 2002.
20. Gh. Păun and X. M. Nguyen, On the inner contextual grammars, *Rev. Roum. Math. Pures Appl.* **25** (1980), pp. 641–651.
21. Gh. Păun, G. Rozenberg, and A. Salomaa, Contextual grammars: erasing, determinism, one-sided contexts, in G. Rozenberg and A. Salomaa, Eds., *Developments in Language Theory*, World Scientific Publ., Singapore, 1994, pp. 370–388.
22. Gh. Păun, G. Rozenberg, and A. Salomaa, Contextual grammars: parallelism and blocking of derivation, *Fundamenta Informaticae* **25** (1996), pp. 381–397.
23. Gh. Păun, G. Rozenberg, A. Salomaa, *The Oxford Handbook of Membrane Computing*, Oxford University Press, 2010.
24. The P systems Web page, `http://ppage.psystems.eu/`.
25. A. Rosenfeld, *Picture Languages*, Academic Press, Reading, MA, 1979.
26. G. Rozenberg, A. Salomaa, Eds., *Handbook of Formal Languages* (3 volumes), Springer, Berlin, 1997.
27. S. Verlan, Recent developments on insertion-deletion systems, *Comp. Sci. J. of Moldova* **18,** 2 (2010), 210–245.
28. S. Verlan, *Study of language-theoretic computational paradigms inspired by biology*, Habilitation thesis, University of Paris Est, 2010.
29. P. S.-P. Wang, Some new results on isotonic array grammars, *Information Processing Letters* **10** (1980), pp. 129–131.