ROMANIAN JOURNAL OF INFORMATION SCIENCE AND TECHNOLOGY
Volume **17**, Number 1, 2014, 103–116

Parallel Array-rewriting P Systems

K.G. SUBRAMANIAN[1], Pradeep ISAWASAN[1],
Ibrahim VENKAT[1], Linqiang PAN[2]

[1] School of Computer Sciences, Universiti Sains Malaysia,
11800 Penang, Malaysia

[2] Key Laboratory of Image Information Processing and Intelligent Control,
School of Automation, Huazhong University of Science and Technology,
Wuhan 430074, Hubei, China

E-mail: lqpan@mail.hust.edu.cn

Abstract. A variant of array-rewriting P systems that uses parallel rewriting of arrays in the regions is introduced. The generative power of this model is compared with certain array grammars generating array languages. It is shown that geometric arrays such as hollow rectangles and hollow squares can be generated by parallel array P systems with a priority relation on the rules. The advantage of the proposed variant is that the number of membranes necessary for generating certain array language is reduced in the constructions as array generation devices.

Key-words: Two-dimensional array; Array grammar; Array P system; Parallel rewriting

1. Introduction

Inspired by the structure and functioning of living cells, Păun introduced a computational model, called P system [8]. P systems have been proved to be versatile models. Many variants have been proposed for investigating different kinds of problems in a variety of areas related to computing [9]. The basic model is made of several membranes, hierarchically arranged within a membrane called the skin membrane. The regions delimited by the membranes can contain objects, which can evolve according to certain evolution rules associated with the regions. In a computation step, the objects in all regions are simultaneously processed by the rules in the regions in a nondeterministic and maximally parallel manner, and at each step all objects

which can evolve should evolve. The evolved objects can then be communicated to other regions, with the communication being specified by a target indication *in* or *out* associated with each rule. If the target indication is *here,* the object remains in the same membrane. A computation halts when no further rule can be applied. Among different kinds of P systems, string-rewriting P systems have been investigated extensively [1,7,8,16]. In these systems, objects are given as finite strings over an alphabet and the evolution rules are given as context-free rewriting rules.

On the other hand, the study of two-dimensional grammar models is an area of investigation motivated by different problems in the framework of image analysis and picture processing [6]. Extending the rewriting feature in Chomsky's grammars in formal language theory [10], several array grammar models have been proposed (see for example [6] and references therein) for generation of picture arrays in the two-dimensional plane. Extending the string-rewriting P systems to arrays, Ceterchi et al. introduced array P systems using context-free rules [3]. Subsequently, several P system models of isometric and non-isometric variety for array generation have been considered in the literature (see for example [4, 12–14]).

In formal language theory, parallel rewriting of strings has been extensively studied. The different kinds of Lindenmayer systems constitute a classic example of such parallel rewriting systems [10]. In the area of string-rewriting P systems, the case of parallel rewriting of a string in a region has been investigated [1, 2, 7]. Here, we consider the feature of parallel rewriting of arrays in the regions of an array P system [3] and introduce a variant of the array P systems, called parallel array P systems. We would like to point out that the array P systems proposed here are mathematical constructs just as the P systems considered in [1–3, 7] are mathematical formulations describing string or array languages. The interesting aspect is that these studies including the present study illustrate the versatile nature and the adaptability of the biologically inspired P systems model.

We compare the generative power of parallel array P systems with certain array generating grammars. The advantage of the proposed array P system model is that the number of membranes is small in many constructions, although by definition, a P system can have any finite number of membranes or regions. We also examine the problem of generating digitized geometric objects such as hollow rectangles and hollow squares which cannot be generated even by context-free array grammars [15]. In order to generate such geometric arrays, we need a priority relation on the array rules, but regular array rules are sufficient. We note that the kind of array grammar rules that we consider in this paper is of the isometric one consistent with the array P systems in [3], in the sense that the arrays in the left and right sides of a context-free array rule are geometrically identical in shape. In the non-isometric case, array P systems that involve sequential as well as parallel rewriting of arrays by suitable rules have been studied (see for example [12] and references therein). In [11, 14], a class of array P systems of the non-isometric type, called sequential/parallel rectangular array P system, is considered in order to generate languages of rectangular picture arrays. As an application of our parallel array P systems, we discuss the generation of a language of certain "floor designs", which enables us to make a comparison with the array P systems in [11, 14].

2. Basic Definitions and Results

For notions and results related to formal language theory we refer to [10] and for those related to array grammars and two-dimensional languages we refer to [5, 6]. A pixel in the two-dimensional plane $\mathbf{Z}^2$ is a point in the plane with integer coordinates. A labelled pixel has a symbol associated with it, with the symbol belonging to a given alphabet V. An array $\mathcal{A}$ in the plane $\mathbf{Z}^2$ consists of finitely many labelled pixels. All other pixels of the plane that are not marked with the symbols of V are assumed to be marked with the *blank symbol* $\# \notin V$. More formally, an array is a mapping $\mathcal{A} : \mathbf{Z}^2 \longrightarrow V \cup \{\#\}$ with a finite support $supp(\mathcal{A})$, where $supp(\mathcal{A}) = \{v \in \mathbf{Z}^2 \mid \mathcal{A}(v) \neq \#\}$. An array can therefore be specified by giving the *pixels* v of the support, along with their associated symbols from V. For example, the non-blank labels of a T shaped array are pictorially indicated in Fig. 1.

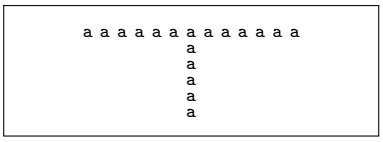

Fig. 1. A T-shaped array.

In more rigorous terms, arrays are regarded as equivalence classes of arrays with respect to linear translations in the plane, i.e. only the relative positions of the non-blank symbols are taken into account, where a translation $\tau_v : \mathbf{Z} \to \mathbf{Z}$ is given by $\tau_v(w) = w + v$ for all $v \in \mathbf{Z}$. Given two arrays α, β, array β is called a *subarray* of α if there exists $v \in \mathbf{Z}$ such that for the translation τ_v it follows that $\tau_v(supp(\beta)) \subseteq supp(\alpha)$. In other words, all labelled pixels of β coincide with the corresponding labelled pixels of α when β is placed on α after a suitable translation of the pattern $supp(\beta)$. The set of all two-dimensional non-empty finite arrays over V is denoted by V^{+2}. The empty array is denoted by λ. The set of all arrays over V is $V^{*2} = V^{+2} \cup \{\lambda\}$. Any subset of V^{*2} is called an *array language*.

An *array production or array rule* p over V is a triple $p = (W, \mathcal{A}, \mathcal{B})$, where W is a finite subset of $\mathbf{Z}^2$ and $\mathcal{A}, \mathcal{B}$ are arrays with their supports included in W. We write the rule as $\mathcal{A} \to \mathcal{B}$ and in the pictorial representation, all pixels of W which are not in $supp(\mathcal{A})$ will be indicated by $\#$, where arrays $\mathcal{A}, \mathcal{B}$ are *isotonic*: $\mathcal{A}$ and $\mathcal{B}$ cover the same pixels, no matter whether they are marked with symbols in V or with $\#$. When graphically representing an array, usually we ignore the blank pixels, but, when representing rewriting rules, the pixels marked with $\#$ are also explicitly shown. An array production is used to rewrite an array in the following way. For two arrays $\mathcal{C}, \mathcal{D}$ over V and a production p as above, we write $\mathcal{C} \Longrightarrow_p \mathcal{D}$ if $\mathcal{D}$ can be obtained by replacing by $\mathcal{B}$ a subarray of $\mathcal{C}$ identical to $\mathcal{A}$, in the sense that the subarray of $\mathcal{C}$ is geometrically identical to $\mathcal{A}$ and the corresponding pixels in the subarray and $\mathcal{A}$ have the same label. In other words, $\mathcal{A}$ and the subarray of $\mathcal{C}$ coincide under a suitable translation in the plane. The reflexive and transitive closure of the relation $\Longrightarrow$ is denoted by $\Longrightarrow^*$.

For example, the array $\begin{smallmatrix} B & a & A \\ & C & \end{smallmatrix}$ derives the array $\begin{smallmatrix} a & a & a & a & a \\ & & a & & \\ & & a & & \end{smallmatrix}$ by applying the rules

$$1.\ A\ \ \# \ \rightarrow\ a\ \ A,\ \ 2.\ \#\ \ B\ \rightarrow\ B\ \ a,\ \ 3.\ \begin{matrix} C \\ \# \end{matrix} \rightarrow \begin{matrix} a \\ C \end{matrix},$$

$$4.\ A \rightarrow a,\ 5.\ B \rightarrow a,\ 6.\ C \rightarrow a.$$

The derivation is shown in Fig. 2, where the rules are applied in the order 1, 2, 3, 4, 5, 6.

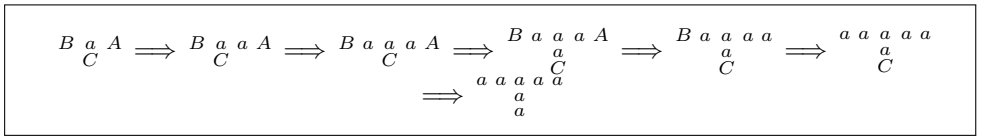

Fig. 2. Illustration of an array derivation.

Let N be a nonterminal alphabet and T be a terminal alphabet. An array production $p = (W, \mathcal{A}, \mathcal{B})$ is called *context-free*, if $supp(\mathcal{A}) \subseteq supp(\mathcal{B})$ and $card(supp(\mathcal{A})) = 1$, where $card(Z)$ is the number of labelled cells in the array Z and the label of the only non-blank symbol of $\mathcal{A}$, is a nonterminal symbol in N. A *regular* rule over the alphabets N and T, is a rule of one of the following forms: $A\ \# \rightarrow a\ B$, $\#\ A \rightarrow B\ a$, $\begin{matrix} \# \\ A \end{matrix} \rightarrow \begin{matrix} B \\ a \end{matrix}$, $\begin{matrix} A \\ \# \end{matrix} \rightarrow \begin{matrix} a \\ B \end{matrix}$, $A \rightarrow B$, $A \rightarrow a$, where $A, B \in N$, $a \in T$ and $\# \notin N \cup T$ is the blank symbol.

The array grammars with context-free or regular array rules are known as the isometric variety in the sense that in an array production, the arrays in the left and right sides of the rule are geometrically identical in shape. In contrast to this, in the non-isometric variety, the rules that rewrite or generate arrays are analogous to the string grammar rules in the sense that the application of a rule $u \rightarrow v$, u, v are either strings or arrays, would mean that enough 'space' is created by 'pushing' symbols, if needed, for v to replace u. Since we deal with context-free and regular array rules only in what follows, we have recalled only these two kinds of array rules. For other kinds of array productions, we refer to [5].

An array grammar is a construct $G = (N, T, \#, S, P)$, where $N \cap T = \emptyset$, N is the nonterminal alphabet, T is the terminal alphabet, $\# \notin N \cup T$ is the blank symbol, $S \in N$ is an axiom array and P is a finite set of array rewriting rules of the form $\mathcal{A} \rightarrow \mathcal{B}$ such that at least one pixel of $\mathcal{A}$ is marked with an element of N.

An array grammar is context-free or regular if all its rules are context-free (CF) or regular, respectively. There is a unique non-blank pixel marked with a nonterminal in the left hand array of each context-free or regular rule. The array language generated by G is

$$L(G) = \{\mathcal{A} \in T^{*2} \mid S \Longrightarrow^* \mathcal{A}\}.$$

The families of array languages generated by context-free and regular array grammars are denoted by ACF and $AREG$, respectively. The following strict inclusion is known: $AREG \subset ACF$ [3].

We now recall the basic model of an array-rewriting P system introduced in [3].

Definition 1. *An array P system* (of degree $m \geq 1$) is a construct

$$\Pi = (V, T, \#, \mu, F_1, \ldots, F_m, R_1, \ldots, R_m, i_o),$$

where V is the alphabet of nonterminals and terminals, $T \subseteq V$ is the terminal alphabet, $\# \notin V$ is the blank symbol, μ is a membrane structure with m membranes labelled in a one-to-one way with $1, 2, \ldots, m$; $F_1, \ldots, F_m$ are finite sets of axiom arrays over V associated with the m regions of μ; $R_1, \ldots, R_m$ are finite sets of array rewriting rules over V associated with the m regions of μ; the array-rewriting rules (context-free or regular) of the form $\mathcal{A} \to \mathcal{B}(tar)$ have attached targets *"here", "out", "in",* "in_j" (in general, we omit mentioning *"here"*); finally, i_o is the label of an elementary membrane of μ which is the output membrane.

A computation in an array P system is defined in the same way as in a string rewriting P system [8] but with the successful computations being the halting ones. Every array, from each region of the system, which can be rewritten by a rule associated with that region (membrane), should be rewritten; the rewriting is *sequential* at the level of arrays, which means that one rule is applied; the array obtained by rewriting is placed in the region indicated by the target associated with the rule used; *here* means that the array remains in the same region, *out* means that the array exits the current membrane and thus, if the rewriting was imposed in the skin membrane, then it exits the system; arrays leaving the system are "lost" in the environment, in_j means that the array is immediately sent to the directly lower membrane with label j and *in* means that the array is immediately sent to one of the directly lower membranes, nondeterministically chosen if several exist; if no internal membrane exists, then a rule with the target indication *in* cannot be used.

A computation is successful only if it stops; that is, a configuration is reached where no rule can be applied to the existing arrays. The result of a halting computation consists of the arrays composed only of symbols from T received in the output membrane with label i_o in the halting configuration. The set of all such arrays computed or generated by a system Π is denoted by $AL(\Pi)$. The families of all array languages generated by array P systems as above, with at most m membranes, with CF or regular array rules, are respectively denoted by $EAP_m(CF)$ and $EAP_m(REG)$.

An array P system generating the array language L_{star} consisting of the star-shaped arrays over $\{a\}$, with each of the stars having four arms of equal length, is given in [13]. In other words,

$$L_{star} = \{ \begin{smallmatrix} & a & \\ a & a & a \\ & a & \end{smallmatrix}, \quad \begin{smallmatrix} & & a & & \\ & & a & & \\ a & a & a & a & a \\ & & a & & \\ & & a & & \end{smallmatrix}, \quad \begin{smallmatrix} & & & & a & & & & \\ & & & & a & & & & \\ & & & & a & & & & \\ a & a & a & a & a & a & a & a & a \\ & & & & a & & & & \\ & & & & a & & & & \\ & & & & a & & & & \end{smallmatrix}, \cdots \}.$$

We now describe the array P system generating the star-shaped arrays of L_{star}. Let $\Pi_{s1} = (V, \{a\}, \#, \mu, F_1, F_2, F_3, F_4, F_5, R_1, R_2, R_3, R_4, R_5, 5)$, where

$$V = \{A, B, C, D, B', C', D', X_1, X_2, X_3, X_4, a\}, \mu = [_1[_2[_3\ [_4\ [_5\ ]_5]_4]_3]_2]_1.$$

The axiom sets are given by

$$F_1 = \{M_1, M_2\}, M_1 = \begin{matrix} & A & \\ D & a & B \\ & C & \end{matrix}, \; M_2 = \begin{matrix} & X_1 & \\ X_4 & a & X_2 \\ & X_3 & \end{matrix}, \; F_2 = F_3 = F_4 = F_5 = \emptyset.$$

The sets of rules are given by

$$R_1 = \{r_{1,1} : \begin{matrix} \# \\ A \end{matrix} \rightarrow \begin{matrix} A \\ a \end{matrix} \; (in), \; r_{1,2} : X_1 \rightarrow a \; (in)\},$$

$$R_2 = \{r_{2,1} : B \; \# \rightarrow a \; B'(in), \; r_{2,2} : B' \rightarrow B \; (out), \; r_{2,3} : X_2 \rightarrow a \; (in)\},$$

$$R_3 = \{r_{3,1} : \begin{matrix} C \\ \# \end{matrix} \rightarrow \begin{matrix} a \\ C' \end{matrix} \; (in), \; r_{3,2} : C' \rightarrow C \; (out), \; r_{3,3} : X_3 \rightarrow a \; (in)\},$$

$$R_4 = \{r_{4,1} : \# \; D \rightarrow D \; a \; (out), \; r_{4,2} : \# \; D \rightarrow D' \; a \; (in), \; r_{4,3} : X_4 \rightarrow a \; (in)\},$$

$$R_5 = \{r_{5,1} : A \rightarrow a, r_{5,2} : B' \rightarrow a, r_{5,3} : C' \rightarrow a, r_{5,4} : D' \rightarrow a\}.$$

Intuitively, a computation in Π_{s1}, that starts with the array M_1, moves the array from region 1 to region 4 through regions 2, 3 with each arm of the star-shaped array growing by the symbol a for each step. The array either comes back to region 1 from region 4 through regions 2, 3 or is sent to region 5. In the former case, the process repeats; while in the latter case, the desired array is formed over $\{a\}$ and is collected in the language.

Lemma 1. ([13]) $L_{star} \in EAP_5(REG)$.

There are many array grammar models of the non-isometric variety generating $m \times n$ $(m, n \geq 1)$ rectangular arrays of symbols with m rows and n columns and among these we recall here the two-dimensional right-linear grammar (2RLG). We follow the description in [6]. There are two sets of rules in a 2RLG grammar: horizontal and vertical rules that correspond to Chomsky regular grammars. This model first generates a (horizontal) string using the horizontal rules and then the vertical rules are applied in parallel generating a rectangular array.

Definition 2. *A two-dimensional right-linear grammar (2RLG)* is a construct $G = (V_h, V_v, \Sigma_I, \Sigma, S, R_h, R_v)$, where

- V_h is a finite set of horizontal nonterminals,
- V_v is a finite set of vertical nonterminals,
- $\Sigma_I \subset V_v$ is a finite set of intermediates,
- Σ is a finite set of terminals,
- $S \in V_h$ is the start symbol,
- R_h is a finite set of horizontal rules of the form $X \rightarrow AY, X \rightarrow A$, $X, Y \in V_h, A \in \Sigma_I$,

- R_v is a finite set of vertical rules of the form of $B \to aD, B \to a,\ B, D \in V_v, a \in \Sigma$.

There are two phases of derivation in a 2RLG. In the first phase, starting with S, the horizontal rules are applied (as in a regular grammar) generating strings over intermediates. In the second phase, each intermediate in such a string serves as the start symbol for the second phase. The vertical rules are applied in parallel, downwards, in this phase for generating the columns of the rectangular arrays over terminals. Note that a 2RLG generates rectangular arrays of symbols. We denote by $L(2RLG)$ the family of array languages generated by two-dimensional right-linear grammars.

3. Parallel Array P Systems

In formal language theory, the concept of rewriting all symbols in a string in parallel is well-known and well-investigated [10]. In the area of string-rewriting P systems, the case of all strings in a region being processed with application of a rewriting rule being sequential in the level of the string, has been studied in detail [8, 16]. The case of rewriting in parallel all symbols in a string in a region has also been investigated [1, 2, 7]. In the array P systems introduced in [3], the sequential analog is used in the sense that only one context-free array rule is used in processing an array in a region. So, it is natural to introduce and examine the feature of parallel rewriting of arrays in the regions of the array P system. This is done in the following definition by introducing a variant of array P systems, called parallel array P system.

Defintion 3. *A parallel array P system* (*PAP*, for short) is a construct

$$\Pi = (V, T, \#, \mu, F_1, \ldots, F_m, R_1, \ldots, R_m, i_o),$$

where the components are defined as in an array P system (Definition 1). In an array $\mathcal{A}$ processed in a region, if there is a context-free array rule that can rewrite a subarray containing a nonterminal A while other pixels, if any, in this subarray have the blank symbol #, then a set of such rules is used to rewrite all nonterminals in the array $\mathcal{A}$. Also, all the context-free array rules applied to an array in a region should have the same target indication, *"here", "in", "out"*. If in a region, no set of rules having the same target indication is available for rewriting all the nonterminals in an array in that region, then the array is not processed and remains in the same region. Also, if two context-free array rules $\mathcal{A} \to \mathcal{B}, \mathcal{C} \to \mathcal{D}$ are applied to an array overlap in the sense that arrays $\mathcal{A}$ and $\mathcal{C}$ have to use some common pixels for successfully applying the rules, then the array is not rewritten. In other words, we consider only the overlap-free case.

The families of all array languages generated by parallel array P systems as above, with at most m membranes, with CF and regular array rules are respectively denoted by $PAP_m(CF)$ and $PAP_m(REG)$.

Remark 1. The requirement that all the rules applied in parallel to an array have the same target indication, is imposed in order to take care of a kind of *deadlock*

situation that might arise when a set of context-free rules with different target indications are applied to an array. This feature of deadlock has been suitably handled in the string case [2].

We now illustrate the working of a parallel array P system.

Example 1. Consider the following parallel array P system
$\Pi_1 = (\{A, B, a\}, \{a\}, \#, [_1[_2]_2]_1, \{ \begin{smallmatrix} A \\ a \ B \end{smallmatrix} \}, \emptyset, R_1, R_2, 2)$, with
$R_1 = \{1. \begin{smallmatrix} \# \\ A \end{smallmatrix} \rightarrow \begin{smallmatrix} A \\ a \end{smallmatrix}$, 2. $B \# \rightarrow a\, B$, 3. $B \rightarrow a\, (in)$, 4. $A \rightarrow a(in)\}$, $R_2 = \emptyset$.

Fig. 3. L-shaped array with equal arms.

Starting with the axiom array $\begin{smallmatrix} A \\ a \ B \end{smallmatrix}$ in region 1, the vertical arm and the horizontal arm are grown together, one symbol at a time, by applying in parallel the rules 1 and 2 as many times as needed. Note that both these rules have the same target indication (*here*). When the rules 3 and 4 (having the same target indication) are used, the derivation halts and the array in the shape of L with equal arms (Fig. 3) enters region 2, where it is collected in the language generated. Note that there are no rules in region 2 and hence no array in region 2 can evolve further.

As noted in Lemma 1, an array P system with five membranes and regular array rules generates the language L_{star}. If we use a parallel array P system, the number of membranes used is only 2, thus yielding a reduction in the number of membranes required.

Lemma 2. $L_{star} \in PAP_2(REG)$.

Proof. We now define a parallel array P system generating the star-shaped arrays of L_{star}. Let $\Pi_{s2} = (V, \{a\}, \#, \mu, A_1, A_2, R_1, R_2, 2)$, where $V = \{A, B, C, D, a\}$, $\mu = [_1[_2\,]_2]_1$. The axiom sets are given by $A_1 = \{M_1 = \ _D \begin{smallmatrix} A \\ a \\ C \end{smallmatrix} \ _B\}$, $A_2 = \emptyset$. The sets of rules are given by

$R_1 = \{r_1 : \begin{smallmatrix} \# \\ A \end{smallmatrix} \rightarrow \begin{smallmatrix} A \\ a \end{smallmatrix}$, $r_2 : B \# \rightarrow a\, B$, $r_3 : \begin{smallmatrix} C \\ \# \end{smallmatrix} \rightarrow \begin{smallmatrix} a \\ C \end{smallmatrix}$, $r_4 : \# D \rightarrow D\, a$,
$r_5 : A \rightarrow a(in)$, $r_6 : B \rightarrow a(in)$, $r_7 : C \rightarrow a(in)$, $r_8 : D \rightarrow a(in)\}$; $R_2 = \emptyset$.
The axiom array M_1 is in region 1. The context-free array rules r_1, r_2, r_3, r_4 in region 1 can be used in parallel as they have the same target indication. The parallel application of these rules grows all the four arms together, one symbol in each arm at one step. When the rules r_5, r_6, r_7, r_8 having the same target indication *in* are applied in parallel, the resulting array that is an element of L_{star} enters region 2, where it will be collected in the language generated. Note that there are no rules in region 2. □

We now compare the generative power of PAP with $AREG$ and $L(2RLG)$.

Theorem 1. $PAP_2(REG) - AREG \neq \emptyset$.

Proof. The array language consisting of arrays over $\{a\}$ in the shape of L with equal horizontal and vertical arms (Fig. 3) is generated by a PAP with 2 membranes and regular array rules as seen in Example 1. But it is straightforward to see that no regular array grammar can generate this language, since in a regular array grammar, a sentential form array contains at most one nonterminal symbol, which means that the horizontal and vertical arms cannot be grown equally by regular array productions. □

Lemma 3. $L(2RLG) \subseteq PAP_3(CF)$.

Proof. Given a $2RLG$, $G = (V_h, V_v, \Sigma_I, \Sigma, S, R_h, R_v)$, we construct a PAP with three membranes and CF array rules as follows:
$\Pi_2 = (V_h \cup V_v \cup \{A' \mid A \in \Sigma_I\} \cup \Sigma, \Sigma, \#, [_1[_2[_3\ ]_3]_2]_1, \{S\}, \emptyset, \emptyset, R_1, R_2, R_3, 3)$, with

$$R_1 = \{X\# \to A'Y,\ A' \to A' \mid X \to AY \in R_h,\ X, Y \in V_h, A \in \Sigma_I\}$$
$$\cup\{X \to A(in) \mid X \to A \in R_h,\ X \in V_h, A \in \Sigma_I\}$$
$$\cup\{A' \to A(in) \mid A \in \Sigma_I\},$$
$$R_2 = \{ \begin{matrix} B \\ \# \end{matrix} \to \begin{matrix} a \\ D \end{matrix} \mid B \to aD \in R_v,\ B, D \in V_v, a \in \Sigma\}$$
$$\cup\{B \to a(in) \mid B \to a \in R_v,\ B \in V_v, a \in \Sigma\},$$
$$R_3 = \emptyset.$$

We note that the rules of R_1 simulate the derivations in the horizontal phase of G generating strings of intermediates. In fact, the rules with target indication *in* terminate a derivation whenever termination happens in the first phase of G and the string is sent to region 2. In this region, the rules of R_2 simulate the parallel derivation of the second vertical phase of G, generating rectangular arrays of the array language generated by G with the arrays being sent to region 3, where they are collected in the language. These arrays cannot further evolve as there are no rules in region 3. □

Lemma 4. $PAP_3(CF) - L(2RLG) \neq \emptyset$.

Proof. Consider the parallel array P system

$$\Pi_3 = (V, \Sigma, \#, [_1[_2[_3\ ]_3]_2]_1, \{AB\}, \emptyset, \emptyset, R_1, R_2, R_3, 3),$$

with $V = \{A, B, X, Y, X', Y', a, b\}$, $\Sigma = \{a, b\}$,

$$R_1 = \{\#A \to AX', B\# \to Y'B, X' \to X', Y' \to Y',\}$$
$$\cup\{A \to X(in), B \to Y(in), X' \to X(in), Y' \to Y(in),\},$$
$$R_2 = \{ \begin{matrix} X \\ \# \end{matrix} \to \begin{matrix} a \\ X \end{matrix},\ \begin{matrix} Y \\ \# \end{matrix} \to \begin{matrix} b \\ Y \end{matrix}, X \to a(in), Y \to b(in)\},$$
$$R_3 = \emptyset.$$

Starting with the axiom array AB in region 1, the array rules of R_1 generate strings (one dimensional arrays) of the form X^nY^n, $n \geq 1$ with the processing ending in region 1 by the application of array rules with target indication *in*. These strings are sent to region 2 where the array rules applied to X^nY^n generate in parallel n columns of strings of a's followed by n columns of strings of b's. In other words, Π_3 generates an array language $L(\Pi_3)$ consisting of $m \times 2n$, $m, n \geq 1$ rectangular arrays with n ($n \geq 1$) columns of a's followed by an equal number of columns of b's. A member of $L(\Pi_3)$ is shown in Figure 4. Thus, $L(\Pi_3) \in PAP_3(CF)$. It can be seen that no $2RLG$ can generate the language of such arrays. In fact, if there is a $2RLG$ generating the array language $L(\Pi_3)$, it will have two phases with both phases involving only regular (string grammar) rules. The first horizontal phase can generate only a regular (string) language, whereas any array in $L(\Pi_3)$ has an equal number of columns of a's and columns of b's, and would therefore involve a non-regular language. □

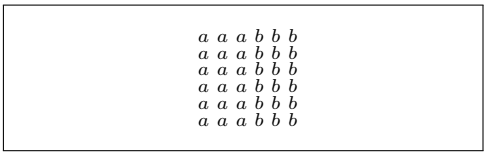

Fig. 4. A rectangular array with 3 columns of a's followed by 3 columns of b's.

As a consequence of Lemmas 3 and 4, we obtain the following Theorem.

Theorem 2. $L(2RLG) \subset PAP_3(CF)$.

Note that the inclusion in Theorem 2 follows from Lemma 3 and the proper inclusion from Lemma 4.

Generation of geometric figures such as rectangles and squares is one of the problems of interest in the study of two-dimensional picture grammars. It is known that hollow rectangles (Figure 5) or squares (Figure 6) made of a's (rectangular frames with hollow region inside the rectangle or square) cannot be generated by even a context-free array grammar [15]. We examine the problem of generation of such hollow rectangles of a's as well as hollow squares of a's in the context of parallel array P system. We use the technique of having a strong priority of application of rules in a region with the same target indication. For example, specifying $r_1 : A\# \rightarrow a\,A,\ r_2 : A \rightarrow a$ with $r_1 > r_2$ would mean that rule r_2 can be applied only when rule r_1 cannot be applied. This kind of specifying priority has been studied in string rewriting P systems [8]. We denote the families of all array languages generated by parallel array P systems with at most m membranes, with context-free or regular array rules and a priority relation $>$ (as mentioned above) on the rules, respectively by $PAP_m(CF, pri)$ and $PAP_m(REG, pri)$.

Lemma 5. *The array language R_H consisting of hollow rectangles of a's belongs to $PAP_2(REG, pri)$.*

Proof. We construct a parallel array P system Π_3 having two nested membranes with regular array rules and a priority specified on the rules, in order to generate R_H. Define

$$\Pi_4 = (V, \Sigma, \#, [_1[_2\ ]_2]_1, \{AaB\}, \emptyset, R_1, R_2, 2),$$

where $V = \{A, B, C, D, A', B', D', a\}, \Sigma = \{a\}$,

$$R_1 = \{A \to A, B\# \to aB, \begin{matrix}\# \\ A\end{matrix} \to \begin{matrix}A' \\ a\end{matrix} (in), \begin{matrix}\# \\ B\end{matrix} \to \begin{matrix}B' \\ a\end{matrix} (in), C \to C,$$

$$\#D \to Da, C \to a(in), \#D \to D'a(in) > D \to a(in)\},$$

$$R_2 = \{ \begin{matrix}\# \\ A'\end{matrix} \to \begin{matrix}A' \\ a\end{matrix}, \begin{matrix}\# \\ B'\end{matrix} \to \begin{matrix}B' \\ a\end{matrix}, A'\# \to aC(out), \#B' \to Da(out)\}.$$

Starting with the axiom AaB, the array rules in region 1 generate a horizontal string of the form Aa^nB which will be the bottom row of the hollow rectangle to be generated. When the leftmost and rightmost ends of this string begin growing one step up by the application of the rules with target indication *in*, the array is sent to region 2. The leftmost and rightmost columns of the hollow rectangle are grown equally, until $A'\# \to aC(out), \#B' \to Da(out)$ are applied, which begin growing the upper row one step from the left and right. The array is sent back to region 1 where upper row is grown from the right. The correct application of the rules $C \to a(in), D \to a(in)$ generates the hollow rectangle over a's (Figure 5), which is then sent to the output region 2. The priority relation $\#D \to D'a(in) > D \to a(in)$ ensures that any premature application of the rules $C \to a(in), \#D \to D'a(in)$ will result in the nonterminal D' appearing in the row which cannot be removed and hence this array is not collected in the array language. □

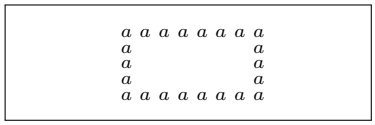

Fig. 5. A hollow rectangle of a's.

Lemma 6. *The array language S_H consisting of hollow squares of a's belongs to $PAP_3(REG, pri)$.*

Proof. We construct a parallel array P system Π_5 having three nested membranes with regular array rules and a priority specified on the rules, in order to generate S_H. Define $\Pi_5 = (\{A, B, C, D, D', a\}, \{a\}, \#, [_1[_2[_3\ ]_3]_2]_1, \{\begin{matrix}A \\ a\ B\end{matrix}\}, \emptyset, \emptyset, R_1, R_2, R_3, 3)$,

$$R_1 = \{ \begin{matrix}\# \\ A\end{matrix} \to \begin{matrix}A \\ a\end{matrix}, B\# \to aB, \begin{matrix}\# \\ B\end{matrix} \to \begin{matrix}D \\ a\end{matrix} (in), A\# \to aC(in)\},$$

$$R_2 = \{C\# \to aC, \begin{matrix}\# \\ D\end{matrix} \to \begin{matrix}D \\ a\end{matrix}, C\# \to aa(in), \begin{matrix}\# \\ D\end{matrix} \to \begin{matrix}D' \\ a\end{matrix} (in) > D \to a(in)\},$$

$$R_3 = \emptyset.$$

Starting with the axiom array $\begin{smallmatrix} A \\ a\ B \end{smallmatrix}$, the left column and bottom row of the hollow square to be generated are grown equally using the array rules in region 1 until the rules with target indication *in* are used, which start the generation of the upper row and right column. The array is sent to region 2 where the upper row and right column are grown until they meet in a correct sequence of application of the rules in this region, with the hollow square of a's generated. Again, the priority relation $\begin{smallmatrix} \# \\ D \end{smallmatrix} \rightarrow \begin{smallmatrix} D' \\ a \end{smallmatrix} (in) > D \rightarrow a(in)$ will ensure that a wrong application will result in the nonterminal D' getting stuck in the upper row and thus such an array is not collected in the language. □

Remark 2. It remains open whether the priority relation in the Lemmas 5 and 6 can be relaxed.

4. An Application and Concluding Remarks

As an application of the parallel P system model, we consider the problem of generating the language L_f of picture arrays describing certain "floor designs". The first two members of the floor designs (in terms of increasing size) are shown in Fig. 7. With the symbol a standing for the floor design $\boxed{\text{✠}}$ in a unit square and b standing for a blank square, we construct a PAP system Π_f with three membranes to generate L_f. Define

$$\Pi_f = (V, \{a, b\}, \#, [_1[_2[_3\ ]_3]_2]_1, F_1, F_2, F_3, R_1, R_2, R_3, 3),$$

where $V = \{A, B, C, D, A_1, B_1, C_1, D_1, X, Y, Z, W, a, b\}$,

$$F_1 = \left\{ \begin{smallmatrix} A & b & B \\ b & a & b \\ C & b & D \end{smallmatrix} \right\},\ F_2 = \left\{ \begin{smallmatrix} X & b & Y \\ b & a & b \\ Z & b & W \end{smallmatrix} \right\},\ F_3 = \emptyset,$$

$$R_1 = \left\{ r_{1,1} : \begin{smallmatrix} \# & \# & \# \\ \# & \# & \# \\ \# & \# & A \end{smallmatrix} \rightarrow \begin{smallmatrix} A_1 & b & a \\ b & a & b \\ a & b & a \end{smallmatrix} (in),\ r_{1,2} : \begin{smallmatrix} \# & \# & \# \\ \# & \# & \# \\ B & \# & \# \end{smallmatrix} \rightarrow \begin{smallmatrix} a & b & B_1 \\ b & a & b \\ a & b & a \end{smallmatrix} (in) \right\}$$

$$\cup \left\{ r_{1,3} : \begin{smallmatrix} \# & \# & C \\ \# & \# & \# \\ \# & \# & \# \end{smallmatrix} \rightarrow \begin{smallmatrix} a & b & a \\ b & a & b \\ C_1 & b & a \end{smallmatrix} (in),\ r_{1,4} : \begin{smallmatrix} D & \# & \# \\ \# & \# & \# \\ \# & \# & \# \end{smallmatrix} \rightarrow \begin{smallmatrix} a & b & a \\ b & a & b \\ a & b & D_1 \end{smallmatrix} (in) \right\},$$

$$R_2 = \{r_{2,1} : A_1 \rightarrow A(out), r_{2,2} : B_1 \rightarrow B(out)\}$$

$$\cup \{r_{2,3} : C_1 \rightarrow C(out), r_{2,4} : D_1 \rightarrow D(out)\}$$

$$\cup \{r_{2,5} : A_1 \rightarrow a(in), r_{2,6} : B_1 \rightarrow a(in), r_{2,7} : C_1 \rightarrow a(in), r_{2,8} : D_1 \rightarrow a(in)\}$$

$$\cup \{r_1 : X \rightarrow a(in), r_2 : Y \rightarrow a(in), r_3 : Z \rightarrow a(in), r_4 : W \rightarrow a(in)\},$$

$$R_3 = \emptyset.$$

The parallel array P system generates the language L_f consisting of arrays over $\{a, b\}$, the first two members (in terms of increasing size) of which are shown in Fig. 6. With

a replaced by ✠ and b by a blank square, the first two members of the language L_f describing "floor design" patterns are shown in Fig. 7.

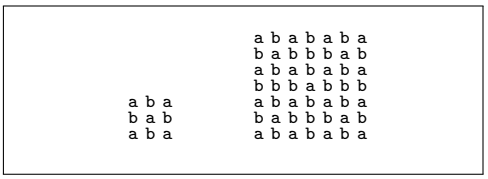

Fig. 6. The first two arrays generated by Π_f.

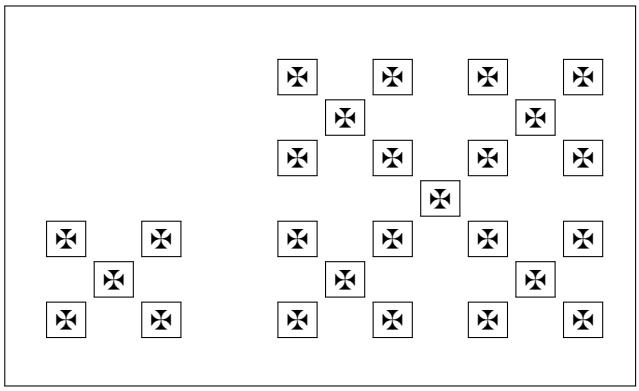

Fig. 7. The first two floor design patterns generated by Π_f.

In [11, 14], a sequential/parallel array P system (S/P RAP) is considered. In this system, the rules in a membrane are either context-free or regular string type rules with (horizontal) sequential rewriting as in Chomsky grammars [10] or the rules in a membrane are string type right-linear rules with rewriting in parallel in the vertical direction as in the two-dimensional (2D) matrix grammars [6]. Pictures that are rectangular arrays are generated by an S/P RAP system. We note that the language L_f cannot be generated by any S/P RAP system, since any array in L_f is a square array and has the feature of having a's in all the cells of the diagonals of the array. An S/P RAP system can at most generate arrays that involve a row having the same symbol since the rewriting in parallel in the vertical direction uses only right-linear type rules. Also, an S/P RAP system has no component that can keep track of the varying depth of a symbol in a diagonal counting from the top row of the square array.

Comparisons with other array generating models can also be made. It will be of interest to examine the possibility of using the parallel array P system models to generate more complex objects.

Acknowledgements. The authors are grateful to the referees for their very useful comments which improved the presentation of the paper very much. Linqiang Pan was supported by National Natural Science Foundation of China (61033003,

91130034, and 61320106005), Ph.D. Programs Foundation of Ministry of Education of China (20100142110072 and 2012014213008), and Natural Science Foundation of Hubei Province (2011CDA027).

References

[1] BESOZZI D., MAURI G., ZANDRON C., *Hierarchies of parallel rewriting P systems - a survey*, New Generation Computing, 2004, Vol. **22**(4), pp. 331–347.

[2] BESOZZI D., FERRETTI C., MAURI G., ZANDRON C., *Parallel rewriting P systems with deadlock*, Lecture Notes in Computer Science, 2003, Vol. **2568**, pp. 302–314.

[3] CETERCHI R., MUTYAM M., PĂUN Gh., SUBRAMANIAN K.G., *Array-rewriting P systems*, Natural Computing, 2003, Vol. **2**, pp. 229–249.

[4] FERNAU H., FREUND R., SCHMID M.L., SUBRAMANIAN K.G., WIEDERHOLD P., *Contextual array grammars and array P systems*, Annals of Mathematics and Artificial Intelligence, 2014 (To appear).

[5] FREUND R., *Array Grammars,* Technical Rep. 15/00, Research Group on Mathematical Linguistics, Rovira i Virgili University, Tarragona, 2000, 164 pages.

[6] GIAMMARRESI D., RESTIVO A., *Two-dimensional languages*. In: Rozenberg G., Salomaa, A. (Eds.), Handbook of Formal Languages, Vol. **3**, Springer Verlag, 1997, pp. 215–267.

[7] KRISHNA S.N., RAMA R., *A note on parallel rewriting in P systems*, Bulletin of the EATCS, 2001, Vol. **73**, pp. 147–151.

[8] PĂUN Gh., *Computing with membranes*, Journal of Computer and System Sciences, 2000, Vol. **61**, pp. 108–143.

[9] PĂUN Gh., ROZENBERG G., SALOMAA A. (Eds.), *The Oxford Handbook of Membrane Computing*, Oxford University Press, Inc., New York, NY, USA, 2010.

[10] ROZENBERG G., SALOMAA A., (Eds.): *Handbook of Formal Languages* (3 volumes), Springer-Verlag, Berlin, 1997.

[11] SARAVANAN R., *A study on two-dimensional grammars based on grammar systems and P Systems*, Ph.D Thesis, Bharath University, India, 2009.

[12] SUBRAMANIAN K.G., *P systems and picture languages*, Lecture Notes in Computer Science, 2007, Vol. **4664**, pp. 99–109.

[13] SUBRAMANIAN K.G., ISAWASAN P., VENKAT I., PAN L., NAGAR A.K., *Array P systems with permitting features*, Journal of Computational Science, 2014 , Vol. **5**, pp. 243–250.

[14] SUBRAMANIAN K.G., SARAVANAN R., ROBINSON T., *P system for array generation and application to kolam patterns*, Forma, 2007, Vol. **22**, pp. 47–54.

[15] WANG P.S.P. (Ed.), *Array Grammars, Patterns and Recognizers*, Series in Computer Science, World Scientific, 1989, Vol. **18**.

[16] ZANDRON C., FERRETTI C., MAURI G., *Two normal forms for rewriting P systems*, Lecture Notes in Computer Science, 2001, Vol. **2055**, pp. 53–164.