# On Parallel Array P Systems

Linqiang Pan[1], Gheorghe Păun[2]

[1] Key Laboratory of Image Processing and Intelligent Control
School of Automation
Huazhong University of Science and Technology
Wuhan 430074, Hubei, China
lqpan@mail.hust.edu.cn

[2] Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucureşti, Romania
gpaun@us.es, curteadelaarges@gmail.com

**Summary.** We further investigate the parallel array P systems recently introduced by K.G. Subramanian, P. Isawasan, I. Venkat, and L. Pan. We first make explicit several classes of parallel array P systems (with one or more axioms, with total or maximal parallelism, with rules of various types). In this context, some results from the above mentioned paper by Subramanian et al. are improved. A series of open problems are formulated.

## 1 Introduction

The generality/versatility of membrane computing is already a well known fact, the computing framework abstracted from the cell structure and functioning can cover a large variety of processes, dealing – in particular – with a large variety of objects processed in the compartments of membrane structures. The arrays (in general, 2D and 3D figures of various types) are one of the types of objects considered already since 2001, see [3]. A direct extension from string objects to two-dimensional arrays was introduced in [1] and then investigated in a series of papers.

A recent contribution to this research area is [6], where a natural counterpart of the array P systems from [1] is considered: parallel rewriting of arrays, instead of the sequential rewriting from [1]. Actually, the kind of parallelism investigated in [6] is that suggested by Lindenmayer systems: all nonterminals of an array should be rewritten in each step. A possible alternative, closer to the style of membrane computing, is to consider the maximal parallelism: a multiset of rules is used which is maximal among the multisets of applicable rules in a given moment.

In the present paper, we explicitly consider these two kinds of parallelism, and we prove that most of the results from [6] hold true for both kinds of parallelism,

also improving those results (less membranes are used in some of them, while the powerful priority relation is avoided in other results).

Several questions remain open; several topics for further research are formulated.

## 2 Definitions and Notations

It is useful for the reader to be familiar with basic elements of membrane computing, e.g., from [4] (with up-dated information available at [7]), and of array grammars, but the used notions will be recalled below. Actually, in what concerns the arrays, we will usually use the pictorial representation, hence we need a minimal formalism (otherwise, cumbersome if rigorously formulated).

The arrays we consider consist of finitely many symbols from a specified alphabet $V$ placed in the points (we call them *pixels*) of $\mathbf{Z}^2$ (the plane); the points of the plane which are not marked with elements of $V$ are supposed to be marked with the *blank symbol* $\# \notin V$. Given an array $W$ over $V$, $supp(W)$ denotes the set of points in $\mathbf{Z}^2$ marked with symbols in $V$. In order to specify an array, it is usual to specify the pixels of the support, by giving their coordinates, together with their associated symbols from $V$, but, as we said above, we will pictorially represent the arrays, indicating their non-blank pixels. These pictures should be interpreted as arrays placed in any position of the plane (congruent, possible to be superposed by means of a translation).

We denote by $V^{*2}$ the set of all two-dimensional arrays of finite support over $V$, including the empty array, denoted by $\lambda$. Any subset of $V^{*2}$ is called an *array language*.

We handle the arrays by means of rewriting rules. An *array rewriting rule* (over an alphabet $V$) is written as a usual string rewriting rule, in the form $W_1 \to W_2$, where $W_1, W_2$ are *isotonic* arrays over $V$: $W_1$ and $W_2$ 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. A rule as above is used to rewrite an array $W$ in the natural way: a position in $W$ is identified where $W_1$ can be superposed, with all pixels matching, whether or not they are marked with symbols in $V$ or with #, and then those pixels are replaced with $W_2$ (the fact that $W_1$ and $W_2$ are isotonic ensures the fact that this replacement is possible). If the result is the array $W'$, we write $W \Longrightarrow W'$. The reflexive and transitive closure of the relation $\Longrightarrow$ is denoted by $\Longrightarrow^*$.

Similar to string rewriting rules, the array productions can be classified according to their form. We consider here only two types of rules, *context-free* and *regular*. Remember that all rules we work with are isotonic (the shapes of the left hand side and the right hand side are identical, only the marking, by blank or non-blank symbols, differs). Thus, a context-free rule is an isotonic one with only one non-blank pixel in its left hand side.

Note that we have not distinguished between terminal and nonterminal symbols, like in Chomsky grammars; for regular rules we need such a distinction. Thus, a regular rule over the alphabets $T$ and $N$, $N$ being the nonterminal one, is a rule of one of the following forms: $A\ \# \to a\ B,\ \#\ A \to B\ a,\ \begin{matrix}\# \\ A\end{matrix} \to \begin{matrix}B \\ a\end{matrix},\ \begin{matrix}A \\ \#\end{matrix} \to \begin{matrix}a \\ B\end{matrix},\ A \to B,\ A \to a$, where $A, B \in N$ and $a \in T$.

Because in what follows we work, like in [6], in a Chomsky framework, with terminal and nonterminal symbols, we also impose that in a context-free rule the single non-blank pixel in the left hand side is marked with a nonterminal symbol.

Before introducing the array P systems, we recall a notion useful below: *two-dimensional right-linear grammars*.

Such a grammar [2] is a construct $G = (V_h, V_v, V_i, T, S, R_h, R_v)$, where $V_h, V_v, V_i$ are the horizontal, vertical, and intermediate alphabets of nonterminals, $V_i \subseteq V_v$, $T$ is the terminal alphabet, $S \in V_h$ is the axiom, $R_h$ is the finite set of horizontal rules, of the forms $X \to AY, X \to A$, for $X, Y \in V_h, A \in V_i$, and $R_v$ is the finite set of vertical rules, of the forms $A \to aB, A \to a$, for $A, B \in V_v, a \in T$.

A derivation in $G$ has two phases, an horizontal one, which uses rules from $R_h$, and a vertical one, which uses rules from $R_v$. The horizontal derivation is as usual in a string grammar. In the vertical phase, the rules are used in parallel, downwards, with the restriction that the terminal rules are used simultaneously for all vertical nonterminals. Thus, in the end, a rectangle is obtained, filled with symbols in $T$. The set of all rectangles generated in this way by $G$ is denoted by $L(G)$ and the family of all languages of this form is denoted by $2RLG$.

Two array languages which will be used below are $L_R$, of all hollow rectangles with the edges marked with $a$ (one element of this language is shown in Fig. 1), and $L_S$, of all hollow squares with the edges marked with $a$.

$$\begin{array}{ccccccccc} a & a & a & a & a & a & a & a & a \\ a & & & & & & & & a \\ a & & & & & & & & a \\ a & & & & & & & & a \\ a & a & a & a & a & a & a & a & a \end{array}$$

**Fig. 1.** A hollow rectangle in $L_R$.

## 3 Parallel Array P Systems

We pass now to define the *parallel array P systems*. Such a device (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 total alphabet, $T \subseteq V$ is the terminal alphabet, $\#$ is the blank symbol, $\mu$ is a membrane structure with $m$ membranes labeled in a one-to-one way

with $1, 2, \ldots, m$, $F_1, \ldots, F_m$ are finite sets of arrays over $V$ associated with the $m$ regions of $\mu$, $R_1, \ldots, R_m$ are finite sets of array rewriting rules over $V$ associated with the $m$ regions of $\mu$; the rules have attached targets *here, out, in* (in general, *here* is omitted), hence they are of the form $W_1 \to W_2(tar)$; finally, $i_o$ is the label of a membrane of $\mu$ specifying the output region.

In what follows, we only consider array P systems with regular (REG) and context-free (CF) rules – with the symbols in $V - T$ considered as nonterminals.

A computation in an array P system is defined in the same way as in a symbol object P system, with the following details. Each array from a compartment of the system must be rewritten by the rules in that compartment. The rewriting is parallel, with two types of parallelism: (1) *the total one*, indicated by *allP*, which means that all nonterminal symbols from the array are rewritten, and (2) *the maximal one*, indicated by *maxP*, which means that a multiset of rules is applied which is maximal, no further rule can be added to it. For any two rules used simultaneously, no pixel of their left hand sides may overlap (i.e., cover the same pixel of the rewritten array).

An important point appears here in what concerns the target indications of the rules: in each compartment, in a step we apply a multiset of rules with the same target indication. This is a very strong restriction, because it refers to all arrays from the compartment. In this paper, we work under this restriction. A weaker and somewhat more natural condition, which remains to be investigated (e.g., are the results proved below valid also in this case?), is to impose the restriction to use rules with the same target separately for each rewritten array (thus, separate arrays may be rewritten by rules with different targets). Of course, the two variants coincide for systems with only one axiom in the initial configuration.

The arrays obtained by an *allP* or a *maxP* rewriting are placed in the region indicated by the target associated with the used rules, in the usual way in membrane computing. It is important to stress the fact that all arrays from a given compartment travel together during a computation.

A computation is successful only if it halts; that is, it reaches a configuration 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 $T$ placed in the region with label $i_o$ in the halting configuration. The set of all such arrays computed (we also say *generated*) by a system $\Pi$ is denoted by $A(\Pi)$.

Note that a computation which produces a terminal array (hence no rule can be applied to it), but still can rewrite another array, is not halting; if the rewriting of one array continues forever, no matter how many terminal arrays were produced, then no result is obtained.

We denote by $PAP_m(ax_k, \alpha, \beta)$ the family of all array languages $A(\Pi)$ generated by systems $\Pi$ as above, with at most $m$ membranes, at most $k$ initial arrays in its compartments ($\Sigma_{i=1}^{m} card(F_i) \leq k$), with rules of type $\alpha \in \{REG, CF\}$, working in the $\beta \in \{allP, maxP\}$ mode. When $m$ or $k$ is not bounded, then it is replaced with $*$.

The following results were proved in [6] (*pri* indicates the use of a priority relation on the rules):

**Lemma 1 (Lemma 3 in [6]).** $2RLG \subseteq PAP_3(ax_1, CF, allP)$.

**Lemma 2 (Lemma 4 in [6]).** $PAP_3(ax_1, CF, allP) - 2RLG \neq \emptyset$.

**Lemma 3 (Theorem 3 in [6]).** $L_R \in PAP_2(ax_1, REG, allP, pri)$.

**Lemma 4 (Theorem 4 in [6]).** $L_S \in PAP_3(ax_1, REG, allP, pri)$.

In what follows, we will improve all these results in terms of the number of membranes in the first two lemmas and avoiding the priority relation in the last two lemmas (these two results are obtained at the price of using more than one axioms or using the *maxP* way of applying the rules).

## 4 Results

The first two lemmas above can be easily improved.

**Proposition 1.** $2RLG \subseteq PAP_2(ax_1, CF, \beta), \beta \in \{allP, maxP\}$.

*Proof.* Let $G = (V_h, V_v, V_i, T, S, T_h, R_v)$ be a two-dimensional right-linear grammar. We construct the array P system $\Pi$ indicated in Figure 2. The horizontal derivation is done in membrane 2. When the horizontal phase is completed, the array is moved to the skin membrane, where the vertical phase is performed. After the use of terminal rules, the array is moved back into the inner membrane. If it is not terminal, then the computation continues forever, by means of the rules $A \to A, A \in V_i$. These rules are also introduced in order to make the *maxP* computations to be *allP* computations. The equality $L(G) = A(\Pi)$ is clear. □

**Proposition 2.** $PAP_2(ax_1, CF, \beta) - 2RLG \neq \emptyset,\ \beta \in \{allP, maxP\}$.

*Proof.* Let us consider the array P system from Figure 3. From the axiom $AB$, we generate a string $X^nY^n$ ($A$ goes to the left, simultaneously with $B$ going to the right), then, like in a two-dimensional right-linear grammar, in the skin region we go vertically, marking the pixels with $a$ in the columns of $X$ and with $b$ in the columns of $Y$. We obtain (moved in the central membrane) a rectangle with the same number of columns marked with $a$ and with $b$, which is not in the family $2RLG$ (the same example was used also in [6]). The system works identically in the *allP* and *maxP* modes. □

Removing the priority from the other two results from [6] can be done, but making use of the possibility of having two axioms in the initial configuration of the systems. One of them will generate the desired arrays, the other one will generate "twin" arrays, which control the computation of the former arrays.

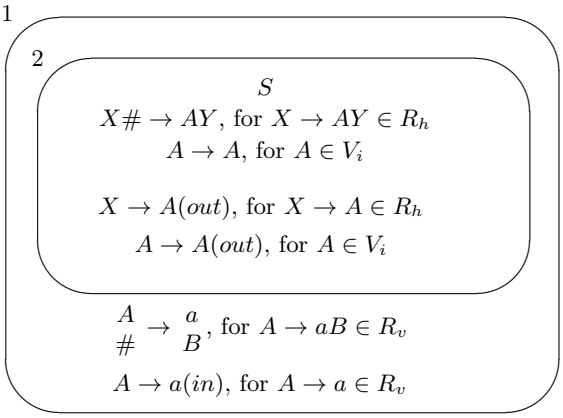


**Fig. 2.** The array P system from the proof of Proposition 1

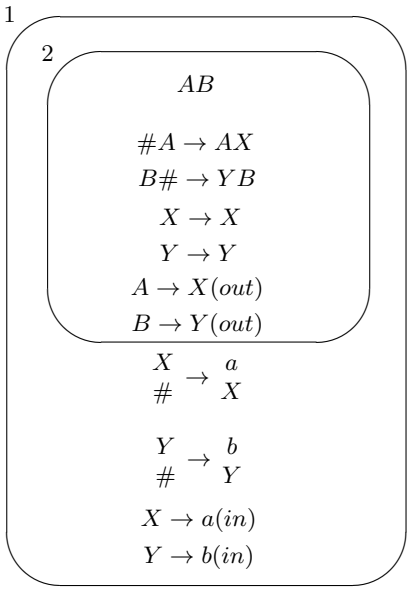


**Fig. 3.** The array P system from the proof of Proposition 2

**Proposition 3.** $L_R \in PAP_2(ax_2, REG, \beta),\ \beta \in \{allP, maxP\}$.

*Proof.* We consider the array P system from Figure 4. The axioms and the rules are written on two columns, in the left one those which lead to the desired arrays, in the right one those handling the "twin" (control) arrays. The rules are similar, with the right column ones dealing with primed nonterminal symbols.

The axiom in the left column is placed in the skin region, the one in the right column is placed in the inner membrane. In the first step, we move the inner axiom to the skin membrane, while removing the subscript 0 from all symbols $A, B, A', B'$. Now we start the generation of the hollow rectangles.

The bottom horizontal line of the arrays is generated in the skin membrane, by moving $B$ and $B'$ to the right. At a given step, we switch to moving vertically, with the arrays moved to membrane 2. We go upwards, synchronously, then the arrays are again moved to the skin membrane, with $D$ and $D'$ being the current nonterminals. Both $D$ and $D'$ go to left and, at some moment, $D$ is replaced with $a$ (hence the "left" array is terminal (but we do not know whether the rectangle is completed). Moved again in membrane 2, the left array remains idle, while the right one can be rewritten – and, because of the parallelism, this must be done – if (and only if) $D'$ has a non-marked pixel in its left. This happens if and only if the terminal array is not a complete hollow rectangle. The symbol $D''$ will go up indefinitely, hence the computation never halts. Thus, only the halting computations produce elements in the language $L_R$. □

A similar result can be obtained for the language of hollow squares.

**Proposition 4.** $L_S \in PAP_2(ax_2, REG, \beta),\ \beta \in \{allP, maxP\}$.

*Proof.* We consider the array P system from Figure 5, again with the axioms and the rules written on two columns, with the same significance as above. In the first step, we move the axiom from the skin region to the inner membrane, while also removing the subscript 0.

In the inner membrane we start constructing the squarea, from the left bottom corner, growing here the left and the bottom edges. At some moment, the two arrays are moved to the skin membrane, where the upper and the right edges are constructed. The work on the "left" array is terminated at the moment when the arrays are moved to the inner membrane. We check here whether or not the square is completed. It is not completed if and only if the nonterminal $B''$ has a non-marked pixel above it. If this is the case, the computation will continue forever, with $B'''$ going to the right, hence the computation never halts. Checking all details remains as an exercise for the reader. □

The *maxP* mode of using the rules is, intuitively speaking, able of "appearance checking", which is known to be a powerful feature of regulated grammars. This is confirmed also in our framework: we can generate the languages $L_R, L_S$ by means of array P systems with only one axiom, at the price of using context-free rules – actually, in the construction below we have only three non-regular rules, of rather simple forms – used in the *maxP* mode.

**Proposition 5.** $L_R \in PAP_2(ax_1, CF, maxP)$.

*Proof.* We consider the array P system from Figure 6. We start in the skin membrane, growing the bottom edge. At some moment, we pass to growing the vertical edges, and the array is moved to the inner membrane.

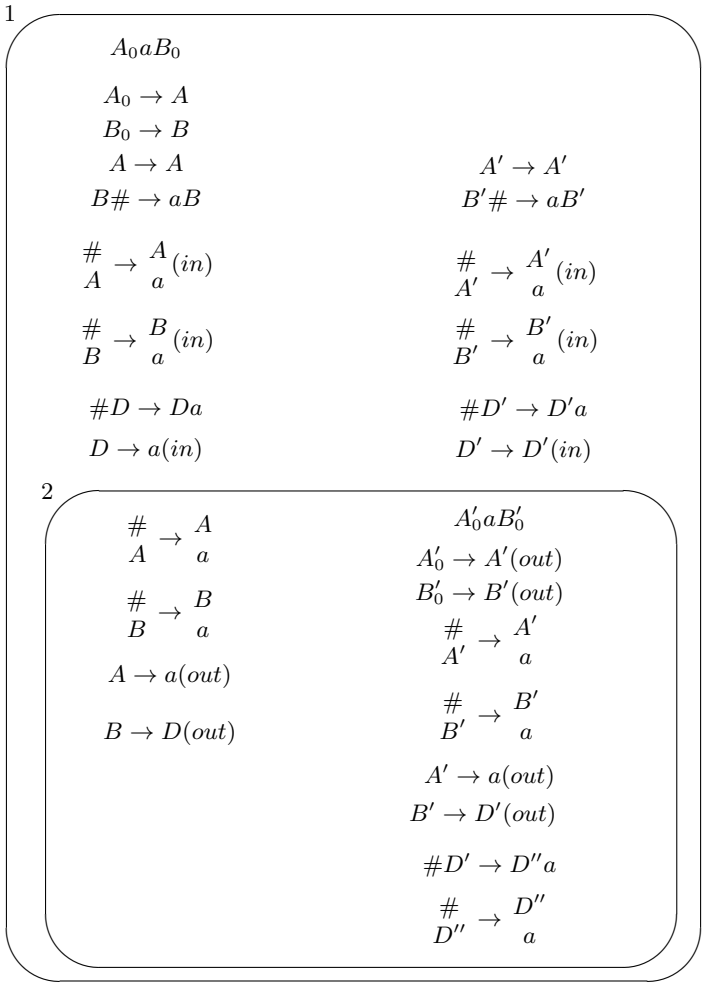


**Fig. 4.** The array P system from the proof of Proposition 3

After the array is moved to the inner membrane, the symbol $B$ introduces two nonterminals, $X$ and $B'$. The latter one will grow the right hand edge, the symbol $X$ waits unchanged until the array is moved back to the skin membrane. Here we both grow the upper edge, by means of the nonterminal $C$, and we change $X$ to $Y$, which is moved to the left, simultaneously with the symbol $C$.

At any moment, $C$ is replaced with $D$ and $Y$ with $Y'$ and the array is sent to membrane 2. Here we check whether the rectangle is completed. $D$ is replaced with $a$. The symbol $Y'$ can be rewritten if and only if it has a non-marked pixel

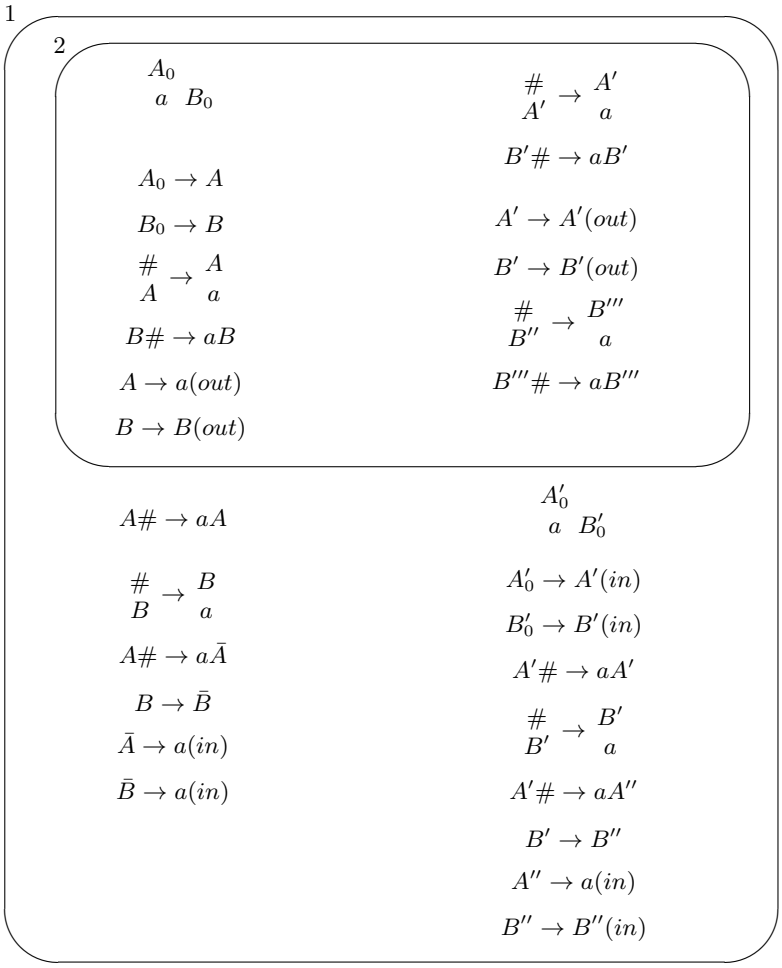


**Fig. 5.** The array P system from the proof of Proposition 4

in its left hand place. If this is the case, hence the rectangle is not complete, the symbol $Z$ is introduced; if not, the rule $\#Y' \to Z\#(out)$ cannot be applied (the *maxP* mode of using the rules allows rewriting only some nonterminals). Anyway, the array is moved back to the skin region (at least the rule $D \to a(out)$ is used). If $Y'$ is still present, then it is simply erased by the rule $Y' \to \#$. Symbol $Z$ cannot be removed, hence the computation leads to a terminal array if and only if the rectangle is completed. □

It remains as an exercise for the reader to use the same idea in order to prove that $L_S \in PAP_2(ax_1, CF, maxP\}$. On the other hand, we see no way to replace *maxP* with *allP* in these two results: $L_R \in PAP_2(ax_1, CF, maxP)$ and $L_S \in PAP_2(ax_1, CF, maxP)$.

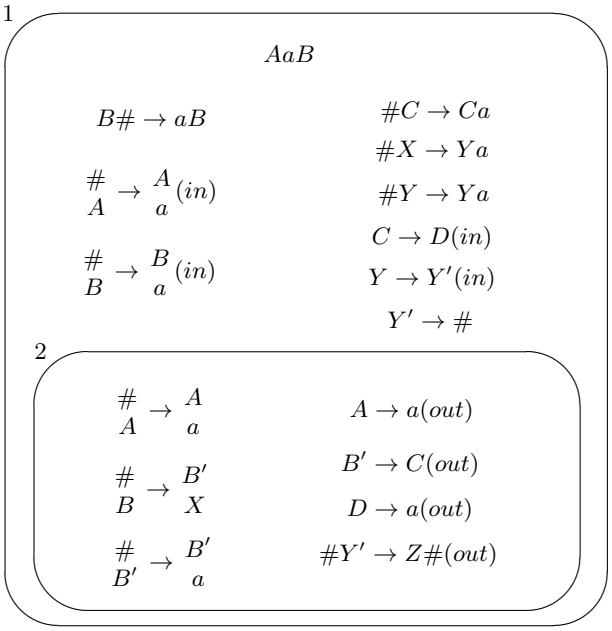


**Fig. 6.** The array P system from the proof of Proposition 5

## 5 Concluding Remarks

The goal of this note was, on the one hand, to make explicit the features involved in an array P system (especially, the number of axioms and the two types of parallelism), and to make use of them in order to improve some of the results in [6], in particular, to get rid of the powerful ingredient of the priority relation. Further research efforts are necessary, to clarify the computing power of parallel array P systems. For instance, we have the families $PAP_m(ax_k, \alpha, \beta)$, for $m \geq 1, k = 1$, $\alpha \in \{CF, REG\}, \beta \in \{allP, maxP\}$. What are the relations among them? Do the parameters $m$ and $k$ induce infinite hierarchies? Besides $CF$ and $REG$ one can also consider non-erasing $CF$ rules (no restriction in this respect was considered here).

A natural question is whether or not parallel array P systems are universal. (The sequential ones considered in [1] are universal.)

Rather natural is the possibility, already mentioned, to impose the use of rules with the same target for each array, separately, thus making possible that two arrays from a given region can go in different places after rewriting. A related issue is to consider other ways to control the communication, such as the $t$ mode from grammar systems area, already investigated, for the sequential case, for array P systems in [5].

Finally, we mention the problem of considering Lindenmayer-like array P systems, that is, pure (without nonterminals) or extended (with all symbols to be rewritten, but accepting only terminal arrays).

**Aknowledgements.** The work of the first author was supported by National Natural Science Foundation of China (61033003, 91130034 and 61320106005).

## References

1. R. Ceterchi, M. Mutyam, Gh. Păun, K.G. Subramanian: Array-rewriting P systems. *Natural Computing*, 2 (2003), 229–249.
2. D. Giammarresi, A. Restivo: Two-dimensional languages. In *Handbook of Formal Languages* (G. Rozenberg, A. Salomaa, eds.), Vol. 3, Springer-Verlag, Berlin, 1997, 215–267.
3. S.N. Krishna, K. Krithivasan, R. Rama: P systems with picture objects. *Acta Cybernetica*, 15, 1 (2001), 53–74.
4. Gh. Păun, G. Rozenberg, A. Salomaa, eds.: *The Oxford Handbook of Membrane Computing*. Oxford University Press, 2010.
5. K.G. Subramanian, R.M. Ali, A.K. Nagar, M. Margenstern: Array P systems and t communication. *Fundamenta Informaticae*, 91, 1 (2009), 145–159.
6. K.G. Subramanian, P. Isawasan, I. Venkat, L. Pan: Parallel array-rewriting P systems. *Romanian Journal of Information Science and Technology*, to appear.
7. The P Systems Website: `http://ppage.psystems.eu`.