Rectangular Arrays and Petri Nets

D. Lalitha[1], K. Rangarajan[2], and Durairaj Gnanaraj Thomas[3]

[1] Department of Mathematics, Sathyabama University
Chennai - 600 119, India
lalkrish_24@yahoo.co.in

[2] Department of Mathematics, Barath University, Chennai - 600 073, India
kr2210@hotmail.com

[3] Department of Mathematics, Madras Christian College
Tambaram, Chennai - 600 059, India
dgthomasmcc@yahoo.com

Abstract. Array Token Petri Net Structure (ATPNS) to generate rectangular arrays has been defined in [6]. We prove that this model generate the regular array languages. By introducing a control on the firing sequence, we have shown that, ATPNS with inhibitor arcs generate the context-free and context-sensitive array languages. Comparisons with other classes of array languages have been made.

Keywords: Array token, Catenation, Inhibitor arcs, Petri net, Picture languages.

1 Introduction

Picture languages generated by grammars or recognized by automata have been advocated since the seventies for problems arising in the framework of pattern recognition and image analysis [2, 7, 9]. In syntactic approaches to generation of picture patterns, several two-dimensional grammars have been proposed. Array rewriting grammars [11], controlled tabled L array grammars [10] and pure 2D context-free grammars [13] are some of the picture generating devices. Applications of these models to generation of "kolam" patterns [12] and in clustering analysis [14] are found in the literature.

On the other hand, a Petri net is an abstract formal model of information flow [4]. Petri nets have been used for analyzing systems that are concurrent, asynchronous, distributed, parallel, non deterministic and/or stochastic. Tokens are used in Petri nets to simulate dynamic and concurrent activities of the system. A language can be associated with the execution of a Petri net. By defining a labeling function for transitions over an alphabet, the set of all firing sequences, starting from a specific initial marking leading to a finite set of terminal markings, generates a language over the alphabet.

Petri net structure to generate rectangular arrays are found in [5, 6]. The two models have different firing rules and catenation rules. In [5] column row catenation petri net structure (CRCPNS) has been defined. A transition with

R.P. Barneva et al. (Eds.): IWCIA 2012, LNCS 7655, pp. 166–180, 2012.

several input places having different arrays is associated with a catenation rule as label. The label of the transition decides the order in which the arrays are joined (columnwise or rowwise) provided the condition for catenation is satisfied. In CRCPNS a transition with a catenation rule as label and different arrays in the input places is enabled to fire.

In ATPNS [6] the catenation rule involves an array language. All the input places of the transition with a catenation rule as label, should have the same array as token, for the transition to be enabled. The size of the array langauge to be joined to the array in the input place, depends on the size of the array in the input place.

In this paper we examine the generative capacity of ATPNS. We find that ATPNS is able to generate only the regular languages. To control the firing sequence inhibitor arcs are introduced. The introduction of inhibitor arcs increases the generative capacity. ATPNS with inhibitor arcs generate the context-free and context-sensitive languages.

The paper is organized as follows: Section 2 defines Array Token Petri Net Structure (ATPNS), language associated with it and gives an example. Section 3 compares ATPNS with three families of array grammars, T0L array grammar with regular control and pure 2D context-free grammar. Section 4 defines Array Token Petri Net Structure with inhibitor arcs, compares with the other six families of array grammars and T0L array grammar with context-free or context-sensitive control.

2 Array Token Petri Nets

In this section we give preliminary definitions of Petri Net and give the notations used. We define Array Token Petri Net Structure (ATPNS), language associated with it and give an example.

A Petri net is one of several mathematical models for the description of distributed systems. A Petri net is a directed bipartite graph, in which the nodes represent transitions (i.e., events that may occur, signified by bars) and places (i.e., conditions, signified by circles). The directed arcs from places to a transition denote the pre-conditions and the directed arcs from the transition to places denote the post-conditions (signified by arrows). Graphically, places in a Petri net may contain a discrete number of marks called tokens. Any distribution of tokens over the places will represent a configuration of the net called a marking. In an abstract sense relating to a Petri net diagram, a transition of a Petri net may fire whenever there are sufficient tokens at the start of all input arcs; when it fires, it consumes these tokens, and places tokens at the end of all output arcs. Transitions can be labeled with elements of an alphabet so that the firing sequence corresponds to a string over the alphabet. A labeled Petri net generates a language. Hack [3] and Baker [1] have published a report on Petri net languages. Petri net to generate string languages is also found in [8].

We now recall the basic definitions of Petri net [4] and the basic notations pertaining to rectangular arrays [11].

Definition 1. *A Petri Net structure is a four tuple $C = (P, T, I, O)$ where $P = \{p_1, p_2, \ldots, p_n\}$ is a finite set of places, $n \geq 0$, $T = \{t_1, t_2, \ldots, t_m\}$ is a finite set of transitions $m \geq 0$, $P \cap T = \phi$, $I : T \to P^\infty$ is the input function from transitions to bags of places and $O : T \to P^\infty$ is the output function from transitions to bags of places.*

Definition 2. *A Petri Net marking is an assignment of tokens to the places of a Petri Net. The tokens are used to define the execution of a Petri Net. The number and position of tokens may change during the execution of a Petri Net. In this paper arrays over an alphabet are used as tokens.*

Definition 3. *An inhibitor arc from a place p_i to a transition t_j has a small circle in the place of an arrow in regular arcs. This means the transition t_j is enabled only if p_i has no tokens. A transition is enabled only if all its regular inputs have tokens and all its inhibitor inputs have zero tokens.*

Basic Notations: Σ^{**} denotes the arrays made up of elements of Σ. If A and B are two arrays having same number of rows then $A \varobar B$ is the column wise catenation of A and B. If two arrays have the same number of columns then $A \ominus B$ is the row wise catenation of A and B. $(x)^n$ denotes a horizontal sequence of n 'x' and $(x)_n$ denotes a vertical sequence of n 'x' where $x \in \Sigma^{**}$. $(x)^{n+1} = (x)^n \varobar x$ and $(x)_{n+1} = (x)_n \ominus x$.

The Petri net model defined here has places and transitions connected by directed arcs. Rectangular arrays over an alphabet are taken as tokens to be distributed in places. Variation in firing rules and labels of the transition are listed out below.

Firing Rules in ATPNS

We define three different types of enabled transition in ATPNS. The pre and post condition for firing the transition in all the three cases are given below:

1. When all the input places of t_1 (without label) have the same array as token.
   - Each input place should have at least the required number of arrays.
   - Firing t_1 removes arrays from all the input places and moves the array to all its output places.

   The graph in Fig. 1 shows the position of the array before the transition fires and Fig. 2 shows the position of the array after transition t_1 fires.

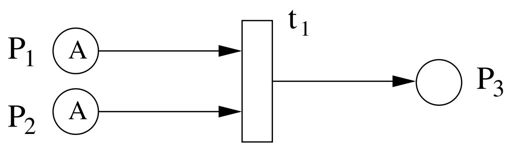

Fig. 1. Position of arrays before firing

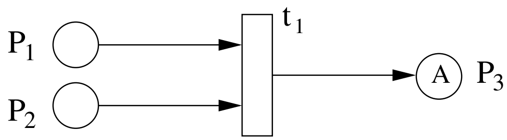

Fig. 2. Position of array after firing

2. When all the input places of t_1 have different arrays as token
   - The label of t_1 designates one of its input places.
   - The designated input place has the same array as tokens.
   - The designated input place has sufficient number of tokens.
   - Firing t_1 removes arrays from all the input places and moves the array from the designated input place to all its output places.

   The graph in Fig. 3 shows the position of the array before the transition fires and Fig. 4 shows the position of the array after transition t_1 fires. Since the designated place is p_1 the array in p_1 is moved to the output place.

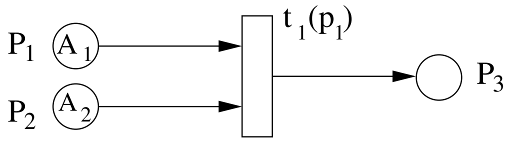

Fig. 3. Transition with label before firing

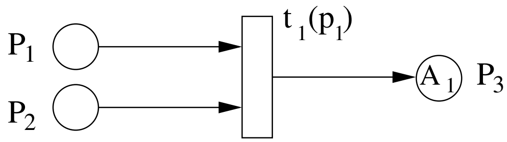

Fig. 4. Transition with label after firing

3. When all the input places of t_1 (with catenation rule as label) have the same array as token
   - Each input place should have at least the required number of arrays.
   - The condition for catenation should be satisfied.
   - The designated input place has sufficient number of tokens.
   - Firing t_1 removes arrays from all the input places p and the catenation is carried out in all its output places.

Catenation Rule as Label for Transitions: Column catenation rule is in the form $A \mathbin{\bigcirc\!\!\!\!|}\ B$. Here the array A denotes the $m \times n$ array in the input place of the transition. B is an array language whose number of rows will depend on 'm' the number of rows of A. The number of columns of B is fixed. For example $A \mathbin{\bigcirc\!\!\!\!|}\ (x \quad x)_m$ adds two columns of x after the last column of the array A which is in the input place. But $(x \quad x)_m \mathbin{\bigcirc\!\!\!\!|}\ A$ would add two columns of x before the first column of A. 'm' always denotes the number of rows of the input array A. Row catenation rule is in the form $A \ominus B$. Here again the array A denotes the $m \times n$ array in the input place of the transition. B is an array language whose number of columns will depend on 'n' the number of columns of A. The number of rows of B is always fixed. For example $A \ominus \begin{bmatrix} x \\ x \end{bmatrix}^n$ adds two rows of x after the last row of the array A which is in the input place. But $\begin{bmatrix} x \\ x \end{bmatrix}^n \ominus A$ would add two rows of x before the first row of the array A. 'n' always denotes the number of columns of the input array A.

An example to explain row catenation rule is given below. The position of the arrays before the transition fires is shown in Fig. 5 and Fig. 6 shows the position of the array afer transition t_1 fires. Since the catenation rule is associated with the transition, catenation takes place in p_3.

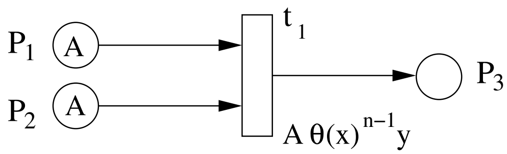

Fig. 5. Transition with catenation rule before firing

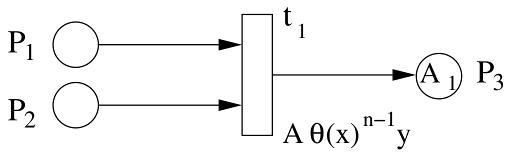

Fig. 6. Transition with catenation rule after firing

If $A = \begin{matrix} a & a & a \\ a & a & a \\ a & a & a \end{matrix}$, the number of columns of A is 3, $n-1$ is 2, firing t_1 adds the row $x \quad x \quad y$ as the last row. Hence $A_1 = \begin{matrix} a & a & a \\ a & a & a \\ a & a & a \\ x & x & y \end{matrix}$.

Definition 4. *If $C = (P, T, I, O)$ is a Petri net structure with arrays over of Σ^{**} as initial markings, $M_0 : P \to \Sigma^{**}$, label of at least one transition being catenation rule and a finite set of final places $F \subset P$, then the Petri net structure C is defined as Array Token Petri Net Structure (ATPNS).*

Definition 5. *If C is a ATPNS then the language generated by the Petri net C is defined as $L(C) = \{A \in \Sigma^{**}/A$ is in p for some p in $F\}$. With arrays of Σ^{**} in some places as initial marking all possible sequences of transitions are fired. The set of all arrays collected in the final places F is called the language generated by C.*

Example 1. $\Sigma = \{x, .\},$, $F = \{p_1\}$

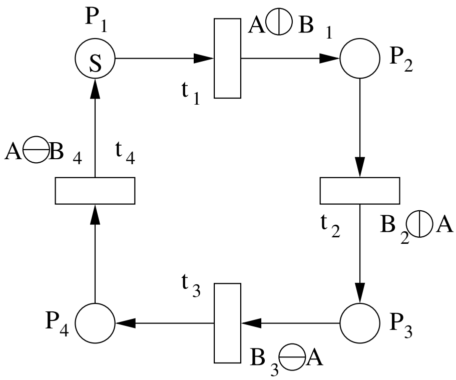

Fig. 7. ATPNS

where $S = \begin{matrix} x & x & x \\ x & . & x \\ . & . & x \end{matrix}$, $B_1 = (.\ x)_m$, $B_2 = (x\ .)_m$, $B_3 = \begin{matrix} x \\ x \end{matrix} \begin{pmatrix} x \\ . \end{pmatrix}^{n-2} \begin{matrix} x \\ x \end{matrix}$ and $B_4 = \begin{pmatrix} x \\ . \end{pmatrix}^{n-2} \begin{matrix} . & x \\ . & x \end{matrix}$.

Firing t_1 puts an array in p_2 making t_2 enabled. Firing t_2 puts an array in p_3 making t_3 enabled. Firing t_3 puts an array in p_4 making t_4 enabled. Firing t_4 puts an array in p_1. The firing sequence $(t_1t_2t_3t_4)^k$, $k \geq 0$ puts a square spiral of size $4k + 3$ in p_1. The language generated by this ATPNS is a set of square spirals. When the transitions t_1, t_2, t_3 and t_4 fire the array that reaches the output place is shown below

$$S \xRightarrow{t_1} \begin{matrix} x & x & x & . & x \\ x & . & x & . & x \\ . & . & x & . & x \end{matrix} \xRightarrow{t_2} \begin{matrix} x & . & x & x & x & . & x \\ x & . & x & . & x & . & x \\ x & . & . & . & x & . & x \end{matrix} \xRightarrow{t_3} \begin{matrix} x & x & x & x & x & x & x \\ x & . & . & . & . & . & x \\ x & . & x & x & x & . & x \\ x & . & x & . & x & . & x \\ x & . & . & . & x & . & x \end{matrix}$$

$$\stackrel{t_4}{\Longrightarrow} \begin{array}{ccccccc} x & x & x & x & x & x & x \\ x & . & . & . & . & . & x \\ x & . & x & x & x & . & x \\ x & . & x & . & x & . & x \\ x & . & . & . & x & . & x \\ x & x & x & x & x & . & x \\ . & . & . & . & . & . & x \end{array}$$

The language generated by this ATPNS is square spirals of size $4n+3$, $n \geq 0$.

3 Comparative Results

In this section we recall the definitions of Array rewriting Grammar [11], Extended Controlled Table L-array Grammar [10], pure 2D context-free grammar with regular control [13] and compare it with ATPNS.

Definition 6. *$G = (V, I, P, S)$ is an array rewriting grammar (AG), where $V = V_1 \cup V_2$, V_1 a finite set of nonterminals, V_2 a finite set of intermediates, I a finite set of terminals, $P = P_1 \cup P_2 \cup P_3$, P_1 is the finite set of nonterminal rules, P_2 is the finite set of intermediate rules, P_3 is the finite set of terminal rules. $S \in V_1$ is the start symbol. P_1 is a finite set of ordered pairs (u, v), u and v in $(V_1 \cup V_2)^+$ or u and v in $(V_1 \cup V_2)_+$.*

P_1 is context-sensitive if there is a (u, v) in P_1 such that $u = u_1 S_1 v_1$ and $v = u_1 \alpha v_1$ where $S_1 \in V_1$, u_1, α, v_1 are all in $(V_1 \cup V_2)^+$ or all in $(V_1 \cup V_2)_+$. P_1 is context-free if every (u, v) in P_1 is such that $u \in V_1$ and v in $(V_1 \cup V_2)^+$ or $(V_1 \cup V_2)_+$. P_1 is regular if $u \in V_1$ and v of the form $U \oplus V$, U in V_1 and V in V_2 or U in V_2 and V in V_1.

P_2 is a set of ordered pairs (u, v), u and v in $(V_2 \cup \{x_1, \ldots, x_p\})^+$ or u and v in $(V_2 \cup \{x_1, \ldots, x_p\})_+$; $x_1, \ldots, x_p$ in I^{++} have same number of rows in the first case and same number of columns in the second case. P_2 is called CS, CF or R as the intermediate matrix languages generated are CS, CF or R.

P_3 is a finite set of terminal rules are ordered pairs (u, v), u in $(V_1 \cup V_2)$ and v in I^{++}.

An Array Grammar is called (CS : CS)AG if the nonterminal rules are CS and at least one intermediate language is CS. An Array Grammar is called (CS : CF)AG if the nonterminal rules are CS and none of the intermediate language is CS. An Array Grammar is called (CS : R)AG if the nonterminal rules are CS and all the intermediate languages are regular. Similarly all the other six families (CF : CS)AG, (CF : CF)AG, (CF : R)AG, (R : CS)AG, (R : CF)AG and (R : R)AG are defined. (X : Y)AL refers to the language generated by the (X : Y)AG, where $X, Y \in \{R, CF, CS\}$.

Definition 7. *An extended, controlled $< k_l, k_r, k_u, k_d >$ table L-array grammar is a 5-tuple $G = (V, T, \mathcal{P}, C, S, \#)$ where V is a finite nonempty set; $T \subseteq V$ is the terminal alphabet of G; $\mathcal{P}$ is a finite set of tables $\{P_1, P_2, \ldots, P_k\}$, and each P_i, $i = 1, 2, \ldots, k$, is a left, right, up or down rules only. The rules within a table* *are all of the same type: either string rules with neighborhood context determined by* $k_l, k_r, k_u, k_d \in \{0, 1\}$*, or matrix rules. In either case, all right-hand sides of rules within the same table are of the same length;* C *is a control language over* $\mathcal{P}$*; and* $S \notin V$ *is the start matrix;* $\#$ *is an element not in* V*.*

In particular if $V = T$ *and* S *is a matrix,* G *is a controlled table L-array grammar; if* $C = \mathcal{P}^*$*, then there is no control and the order of applications of the tables is arbitrary;* G *is then an extended table L-array grammar.*

If $k_l = k_r = k_u = k_d = 0$*, then the rules are all context-free (0L) table array grammar. If at least one of* k_l, k_r, k_u, k_d *equals 1 then we get a context dependent (1L) table array grammar.*

$(\gamma)TXLAL$ *refers to the language generated by table XL array grammar with* γ *control;* X *may be 0 or 1 and* γ *may be R, CF or CS.*

Definition 8. *A pure 2D context-free grammar (P2DCFG) is a 4-tuple* $G = (\Sigma, P_c, P_r, \mathcal{M}_0)$*, where*

- Σ *is a finite set of symbols.*
- $P_c = \{t_{c_i} | 1 \leq i \leq m\}$, $P_r = \{t_{r_j} | 1 \leq j \leq n\}$.
  Each t_{c_i}*,* $(1 \leq i \leq m)$*, called a column table, is a set of context-free rules of the form* $a \to \alpha$*,* $a \in \Sigma$*,* $\alpha \in \Sigma^*$ *such that for any two rules* $a \to \alpha$*,* $b \to \beta$ *in* t_{c_i}*, we have* $|\alpha| = |\beta|$*, where* $|\alpha|$ *denotes the length of* α*.*

  Each t_{r_j}*,* $(1 \leq j \leq n)$*, called a row table, is a set of context-free rules of the form* $c \to \gamma^T$*,* $x \in \Sigma$*,* $\gamma \in \Sigma^*$ *such that for any two rules* $c \to \gamma^T$*,* $d \to \delta^T$ *in* t_{r_j}*, we have* $|\gamma| = |\delta|$*.*
- $\mathcal{M}_0 \subseteq \Sigma^{**} - \{\lambda\}$ *is a finite set of axiom arrays.*

Derivations are defined as follows. For any two arrays M_1, M_2*, we write* $M_1 \Rightarrow M_2$ *if* M_2 *is obtained from* M_1 *by either rewriting a column of* M_1 *by rules of some column table* t_{c_i} *in* P_c *or a row of* M_1 *by rules of some row table* t_{r_j} *in* P_r*.* $\Rightarrow^*$ *is the reflexive transitive closure of* $\Rightarrow$*.*

The picture array language $L(G)$ *generated by* G *is the set of rectangular picture arrays* $\{M | M_0 \Rightarrow^* M \in \Sigma^{**}$*, for some* $M_0 \in \mathcal{M}_0\}$*. The family of picture array languages generated by pure 2D context-free grammars is denoted by P2DCFL.*

Definition 9. *A pure 2D context-free grammar with a regular control is* $G_c = (G, Lab(G), \mathcal{C})$ *where* G *is a pure 2D context-free grammar,* $Lab(G)$ *is a set of labels of the tables of* G *and* $\mathcal{C} \subseteq Lab(G^*)$ *is a regular (string) language. The words in* $Lab(G)^*$ *are called control words of* G*. Derivations* $M_1 \Rightarrow_w M_2$ *in* G_c *are done as in* G*, except that if* $w \in Lab(G^*)$ *and* $w = l_1 l_2 \ldots l_m$*, then the tables of rules with labels* $l_1, l_2, \ldots, l_m$ *are successively applied starting with* M_1 *to yield* M_2*. The picture array language generated by* G_c *consists of all picture arrays obtained from the axiom array of* G *with the derivations controlled as described above. We denote by (R)P2DCFL the family of picture array languages generated by pure 2D context-free gramamrs with a regular control.*

Theorem 1. *The class of table 0L array languages without control or with regular control can be generated by ATPNS.*

Proof. Let $G = (V, \mathcal{P}, C, S)$ be a table 0L array grammar; where V is a finite set of terminals, $\mathcal{P}$ is a finite set of tables $\{P_1, P_2, \ldots, P_k\}$, and each P_i, $i = 1, 2, \ldots, k$, is a left, right, up or down rules only. S is the start array.

The rules within a table are all of the same type. The left hand side of each production is a single terminal. The right hand side of all the rules within the same table is of the same length. Each table will have at least one rule for each symbol on the boundary. If say, P_1 has left (right) rules then applying the rules of P_1 will amount to column catenation. Similarly applying the table containing up(down) rules will amount to row catenation.

Let us construct an array token Petri net structure when there is no control on the application of the tables. Let p_1 be the place with the start array S as a token. For every table $P_i \in \mathcal{P}$ have a transition t_i with the corresponding row or column catenation rule as label. Have k transitions one each for the k tables in $\mathcal{P}$ with p_1 as both input place and output place of all the transitions. $F = \{p_1\}$. Every time a table production is required to be used the corresponding transition is fired. Since there is no control the tables can be applied in any order to generate the language. In the net p_1 is the output place of every transition. Hence after the firing of any transition the array reaches p_1, so at any given time all the k transitions are enabled. Thus the Petri net constructed will generate the language generated by the grammar G.

Let us construct an array token Petri net structure when a regular control $C = (P_1 P_2 \ldots P_k)^*$ is defined on the application of tables. Have k transitions and k places. Let S the start array be a token in place p_1. Let t_1 be a transition with the catenation rule, which corresponds to the table P_1, as label; p_1 being the input place and p_2 as its output place. Let t_2 be a transition with the catenation rule, which corresponds to the table P_2, as label; p_2 being the input place and p_3 as its output place. Continuing like this have a transition t_k with the catenation rule, which corresponds to the table P_k, as label; p_k being the input place and p_1 as its output place. $F = \{p_1\}$. Firing t_k puts a token in p_1 so that t_1 is enabled again. The firing sequence $t_1 t_2 \ldots t_k$ will have the same effect as applying the production rules $P_1 P_2 \ldots P_k$ in that order once. The effect of the regular control is got by placing the transitions with those labels in the same order forming a loop in the net so that the sequence of transitions can be fired any number of times. Thus the Petri net constructed will generate the language generated by the grammar G. □

Theorem 2. *$(R)T0LAL$ is properly contained in the family generated by $ATPNS$.*

Proof. Let us give an example to prove this theorem. The language of square spirals given in Example 1 is a $(R)T1LAL$ [10]. Thus ATPNS can generate certain languages that do not belong to $(R)T0LAL$, which proves a proper containment. □

Theorem 3. *The families of $(R : Y)AL$, where $Y \in \{R, CF, CS\}$, can be generated by $ATPNS$.*

Proof. Let $G = (V, I, P, S)$ be an array grammar, where $V = V_1 \cup V_2$, V_1 a finite set of nonterminals; V_2 a finite set of intermediates; I a finite set of terminals,

$P = P_1 \cup P_2$, P_1 is the finite set of regular nonterminal rules; P_2 is the finite set of terminal rules. $S \in V_1$ is the start symbol. For each A in V_2, M_A is an intermediate language.

In array grammars the derivation is as follows. Starting with S the nonterminal rules are applied without any restriction, as in string grammar, till all the nonterminals are replaced. Then replace each intermediate A by M_A subject to the conditions imposed by the row and column catenation operator. Let the regular non-terminal rules of G generate the infinite sequence of matrices $\{M_n/n \geq 1\}$ where M_n is in any one of the following forms. For all $n > 1$, $M_n = (X \ominus M_{n-1}) ⦶ Y$ or $M_n = Y ⦶ (X \ominus M_{n-1})$ or $M_n = Y ⦶ (M_{n-1} \ominus X)$ or $M_n = (M_{n-1} \ominus X) ⦶ Y$, where X and Y are chosen from intermediate matrix languages L_X and L_Y (subject to conditions imposed by row and column catenation). The recursive definition of M_n is assumed to be unique. $S \to M_1$ is the terminal rule.

Construction of ATPNS, for the case when $M_n = (X \ominus M_{n-1}) ⦶ Y$ where X and Y are intermediates, is given below. For the other cases the construction is similar. Define the arrays B_X and B_Y corresponding to the intermediate language X and Y. Put M_1 in the start place p_1 as a token. Have a transition t_1 with the row catenation rule $B_X \ominus A$ as a label. Let p_1 be the input place of t_1. The number of rows of B_X is fixed but the number of columns of B_X is dependent on 'n' the number of columns of A. A is the array that reaches the input place p_1 of the transition t_1 during the course of the execution of the net. Let p_2 be the output place for the transition t_1. The array B_Y is defined similar to the intermediate language generated by Y. Have a transition t_2 with the column catenation rule $A ⦶ B_Y$ as a label. Let p_2 be the input place of t_2. The number of columns of B_Y is fixed but the number of rows of B_X is dependent on 'm' the number of rows of A. A is the array that reaches the input place p_2 of the transition t_2 during the course of the execution of the net. Let p_1 be the output place for the transition t_2. First time the sequence $t_1 t_2$ is executed, the matrix M_2 is put in p_1. Let $F = \{p_1\}$ be the final set of places. The firing sequence $(t_1 t_2)^k$ puts the matrix M_{k+1}, $k \geq 0$ in p_1. Thus $\{M_n/n \geq 1\}$ of matrices is the language generated. □

Theorem 4. *The families of* $(R : Y)AL$*, where* $Y \in \{R, CF, CS\}$*, are properly contained in the family generated by ATPNS.*

Proof. Let us give an example to prove this theorem. Kirsch's right triangles is a $(CF : R)AL$ [11]. But ATPNS can generate Kirsch's right triangles. Thus ATPNS generates certain languages which do not belong to $(R : Y)AL$, which proves proper containment. □

Theorem 5. *Any (R)P2DCFL can be generated by ATPNS.*

Proof. Let the language be generated by a P2DCFG with a regular control, $G_c = (G, Lab(G), \mathcal{C})$ where G is a P2DCFG, $Lab(G)$ is the set of all labels and $\mathcal{C}$ is the control language.

Application of a column table is equivalent to a column catenation. Hence for every t_{c_j}, we can define an equivalent column catenation rule. Similarly for every row table t_{r_j}, an equivalent row catenation rule can be defined.

Let M_1 be an array derived from the axiom array M_0 using the control words $w = l_1 l_2 \dots l_m$. We give the steps for constructing the ATPNS to generate the language, assuming that all the tables are used on the boundary of M_0.

Let p_0 be a place with the array M_0 as token. Let t_1 be a transition with the catenation rule corresponding to l_1 as a label, p_0 as input place and p_1 as output place. Let t_2 be a transition with the catenation rule corresponding to l_2 as label, p_1 as input place and p_2 as output place. Proceeding like this let t_m be a transition with the catenation rule corresponding to ℓ_m as label, p_{m-1} as input place and p_0 as output place. The firing sequence $t_1 t_2 \dots t_m$ has the same effect as of applying the tables $l_1, l_2, \dots, l_m$ to the array M_0. This Petri net structure generates all the arrays that can be generated by the control words of $(l_1 l_2 \dots l_m)^*$.

If all the tables are not applied on the boundary of M_0, then consider a subarray M_{01} of M_0 such that the table l_1 is applied to the boundary of M_{01}. Take M_{01} as a token in p_0 and construct the ATPNS as given above. Add a transition t_{m+1} with input place p_0 and output place p_m. The label of the transition should have the catenation rule, which joins the row/column that was removed from M_0. Required number of transitions should be added to join all the rows/columns that were removed from M_0. $\square$

4 Array Token Petri Nets with Inhibitor Arcs

Since ATPNS is able to generate only T0L with regular control and $(R : Y)AL$, where $Y \in \{R, CF, CS\}$ we use inhibitor arcs to control the firing sequence. This section introduces Array Token Petri Net Structure with inhibitor arcs and compares it with the other array languages and tabled array languages.

Firing Rules in ATPNS with inhibitor arcs is similar to the firing rules of ATPNS with the extra condition that any transition with inhibitor input can fire only if the inhibitor input does not have any array.

Definition 10. *An Array Token Petri Net Structure with at least one inhibitor arc is defined as Array Token Petri Nets with inhibitor arcs.*

The language generated by the Petri net is the set of all arrays which reach the final place.

Example 2. $\Sigma = \{x \bullet\}$, $S = \begin{matrix} x & \bullet \\ x & \bullet \end{matrix}$, $B_1 = (\bullet)_m$, $B_2 = (x)_m$, $B_3 = \begin{pmatrix} x \\ x \\ x \end{pmatrix}^{\frac{n}{2}} \begin{pmatrix} \bullet \\ \bullet \\ \bullet \end{pmatrix}^{\frac{n}{2}}$,

$B_4 = (x)^{\frac{n}{2}} (\bullet)^{\frac{n}{2}}$, $F = \{p_2\}$.

To start with both t_2 and t_5 are enabled. The sequence $t_2 t_3 t_4$ can be fired any number of times. Once t_5 is fired the inhibitor input p_6 makes sure that the sequences $t_7 t_8$ is also fired the same number of times. When p_6 is empty the transition t_9 fires to put the final array into p_2.

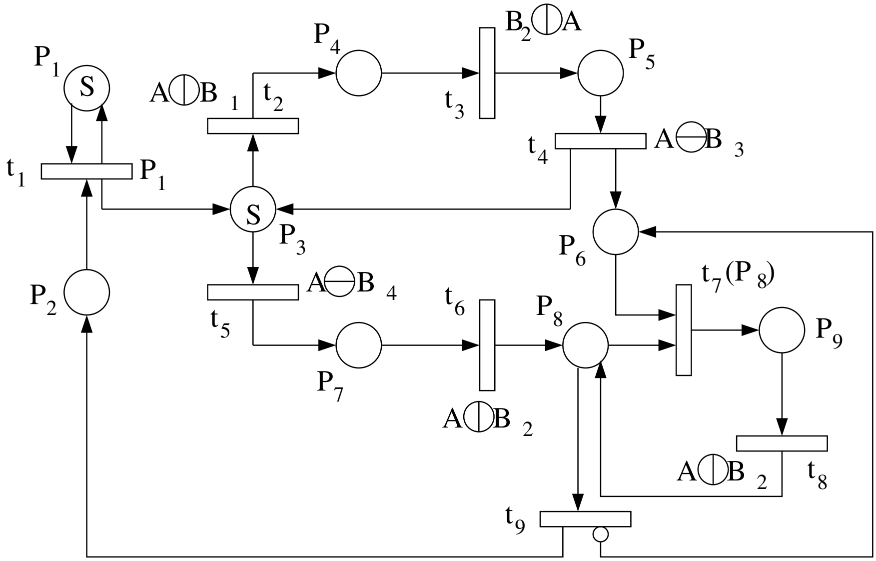

Fig. 8. ATPNS with inhibitor arc

The array generated by the firing sequence $t_1 \dots t_9$ is given below.

$$S \stackrel{t_1 \dots t_9}{\Longrightarrow} \begin{array}{cccccc} x & x & \bullet & \bullet & x & x \\ x & x & \bullet & \bullet & x & x \\ x & x & \bullet & \bullet & x & x \\ x & x & \bullet & \bullet & x & x \\ x & x & \bullet & \bullet & x & x \\ x & x & \bullet & \bullet & x & x \end{array}$$

The language generated is squares split into three equal columns.

Theorem 6. *The language generated by a table 0L array grammar with context-free or context-sensitive control can be generated by ATPNS with inhibitor arcs.*

Proof. Let $G = (V, \mathcal{P}, C, S)$ be a table 0L array grammar with context-free control, where V is the set of terminals, $\mathcal{P}$ is a finite set of tables $\{P_1, P_2, \dots, P_k\}$, $C = (P_1 \dots P_i)^n (P_j \dots P_k)^n$, $1 \leq i, j \leq k$, be a context-free control and S is the start array.

Construct an ATPNS with two subnets C_1 and C_2 as in figure. Let p_1 belong to C_1 with the start array S as a token. Have transition t_1 with the catenation rule which corresponds to P_1, p_1 being the input place and p_2 as its output place. Transition t_2 with the catenation rule which corresponds to P_2, p_2 being the input place and p_3 as its output place and so on. Transition t_i with the catenation rule which corresponds to P_i, p_i being the input place and p_1, M_1 as its output places. The subnet C_1 can be executed any number of times. The sequence $(t_1 t_2 \dots t_i)^n$ would put n different arrays as tokens in M_1. But the

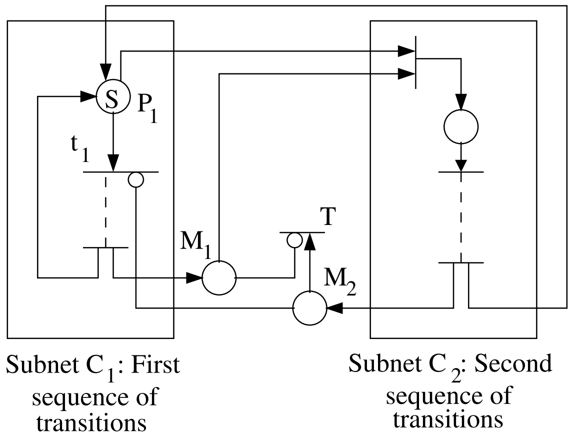

Fig. 9. Subnets of ATPNS with inhibitor arcs

place p_1 will have the array which is the array that would result in applying the tables $P_1 \dots P_i$ n times to S. Once t_j in C_2 is fired the second subnet starts its execution. Since M_1 is an input place for t_j, the subnet C_2 can be executed at the most n times (the number of times C_1 was executed). Similar to C_1 in C_2 there is a transition for every table $P_j, \dots, P_k$. Whenever C_2 is executed once an array is put in M_2 and p_1. This array would be the array that results after applying the tables $(P_1 \dots P_i)^n$ $(P_j \dots P_k)^m$ (m is the number of times C_2 was executed) to S. Once C_2 starts its execution C_1 cannot be executed again till M_2 is empty as M_2 is an inhibitor input for t_1. After executing C_2 'n' times M_2 can be emptied by firing T 'n' times. Since M_1 is an inhibitor input for T, T cannot be fired until M_1 is empty. In other words M_2 cannot be emptied until C_2 is executed exactly n times. Thus the subnets C_1 and C_2 are executed the same number of times. Hence the sequences $t_1 \dots t_i$ and $t_j \dots t_k$ can be fired exactly the same number of times. This is the effect of a context-free control.

Thus using the concepts of inhibitor arcs we are able to have a context-free control on the firing sequence. Similarly with three subnets and with proper usage of inhibitor inputs we can have a context-sensitive control on the firing sequence. □

Theorem 7. *The families of $(X : Y)AL$, where $X \in \{CF, CS\}$ and $Y \in \{R, CF, CS\}$, can be generated by ATPNS with inhibitor arcs.*

Proof. Let $G = (V, I, P, S)$ be an $(CF : Y)AG$. Then the nonterminal rules be of the form $(A)^n(B)^n$ or $\frac{(A)_n}{(B)_n}$ where A, B are intermediates. L_A and L_B the intermediate languages are regular, context-free or context-sensitive. Similar to the construction given in the proof of Theorem 6 have two subnets C_1 and C_2. The subnet C_1 should generate the intermediate language L_A and the subnet C_2 should generate the intermediate language L_B. With the use inhibitor inputs we can make sure the subnets C_1 and C_2 are executed the same number of times. Thus with inhibitor arcs any $(CF : Y)AL$ can be generated.

For any $(CS : Y)AG$ the nonterminal rules are of the form $(A)^n(B)^n(C)^n$ or $\begin{matrix}(A)_n\\(B)_n\\(C)_n\end{matrix}$, where A, B, C are intermediates. L_A, L_B and L_C the intermediate languages are regular, context-free or context-sensitive. With three subnets and with proper usage of inhibitor inputs we can generate all $(X : Y)$ array languages, where $X \in \{CF, CS\}$ and $Y \in \{R, CF, CS\}$ can be generated by ATPNS with inhibitor arcs. □

5 Conclusion

Array token Petri net structure generates rectangular arrays. This model is able to generate (R)P2DCFL, three of the nine families of array languages and the tabled 0L languages with regular control. Introducing inhibitor arcs to ATPNS the other six families of Array Languages and tabled 0L languages with context-free or context-sensitive control can also be generated. The languages generated by the nine families of array grammars and tabled 0L grammars with the three types of control can all be generated by ATPNS with inhibitor arcs.

References

1. Baker, H.G.: Petri net languages. Computation Structures Group Memo 124, Project MAC. MIT, Cambridge (1972)
2. Giammarresi, D., Restivo, A.: Two-dimensional langauges. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages 3, pp. 215–267. Springer (1997)
3. Hack, M.: Petri net languages. Computation Structures Group Memo 124, Project MAC. MIT (1975)
4. Peterson, J.L.: Petri Net Theory and Modeling of Systems. Prentice Hall, Inc., Englewood Cliffs (1981)
5. Lalitha, D., Rangarajan, K.: Column and row catenation petri net systems. In: Proceedings of the Fifth IEEE International Conference on Bio-Inspired Computing: Theories and Applications, pp. 1382–1387 (2010)
6. Lalitha, D., Rangarajan, K.: An application of array token Petri nets to clustering analysis for syntactic patterns. International Journal of Computer Applications 42(16), 21–25 (2012)
7. Rosenfeld, A., Siromoney, R.: Picture languages - A survey. Languages of Design 1(3), 229–245 (1993)
8. Shirley Gloria, D.K., Immanuel, B., Rangarajan, K.: Parallel context-free string-token Petri nets. International Journal of Pure and Applied Mathematics 59(3), 275–289 (2010)
9. Siromoney, R.: Advances in Array Languages. In: Ehrig, H., Nagl, M., Rosenfeld, A., Rozenberg, G. (eds.) Graph Grammars 1986. LNCS, vol. 291, pp. 549–563. Springer, Heidelberg (1987)
10. Siromoney, R., Siromoney, G.: Extended controlled table L-arrays. Information and Control 35, 119–138 (1977)

11. Siromoney, G., Siromoney, R., Kamala, K.: Picture languages with array rewriting rules. Information and Control 22, 447–470 (1973)
12. Siromoney, G., Siromoney, R., Kamala, K.: Array grammars and kolam. Computer Graphics and Image Processing 3(1), 63–82 (1974)
13. Subramanian, K.G., Rosihan, M., Ali, G.M., Nagar, A.K.: Pure 2D picture grammars and languages. Discrete Applied Mathematics 157(16), 3401–3411 (2009)
14. Wang, P.S.P.: An application of array grammars to clustering analysis for syntactic patterns. Pattern Recognition 17(4), 441–451 (1984)