A Study on Expressiveness of a Class of Array Token Petri Nets

T. Kamaraj, D. Lalitha and D. G. Thomas

Abstract Adjunct Array Token Petri Net structure (AATPNS) to generate rectangular pictures has been defined in Lalitha et al (Indian J. Math. Math. Sci. 8(1):11–19, 2012) 7]. AATPNS with inhibitor arcs generated context free and context sensitive Kolam Array languages and Tabled 0L/1L languages. In this paper we study the expressiveness of this model by comparing with some other interesting array generating grammar devices like Pure 2D context free grammars with regular control, Regional tile rewriting Grammars, Prusa Grammars and also comparing with local languages.

Keywords Array token Petri nets · Adjunction · Pure 2D grammars · Regional tile grammars · Prusa grammars · Local languages

1 Introduction

Since seventies, the study of two dimensional languages generated by Grammars or recognized by Automata have been found in the theory of formal languages with the insight of computations in picture processing and pattern recognition [5, 12, 13]. In the quest of syntactic techniques for generation of digital picture patterns, a number of 2D Grammars have been proposed. Siromoney Matrix grammars [17],

T. Kamaraj (✉) · D. Lalitha
Department of Mathematics, Sathyabama University, Chennai 600119, India
e-mail: Kamaraj_mx@yahoo.co.in

D. Lalitha
e-mail: lalkrish_24@yahoo.co.in

D. G. Thomas
Department of Mathematics, Madras Christian College, Chennai 600059, India
e-mail: dgthomasmcc@yahoo.com

Kolam Array Grammars (KAG) [15, 16], Tabled 0L/1L grammars (T0LG/TILG) [14] are some of the classical formalisms. Pure 2D context free grammars with regular control (RP2DCFG) [18], Prusa grammars (PG) [11], Regional Tile Rewriting grammars [10] are some of the recent and more expressive grammars. Tiling systems [3, 5] is a recognizing device for a ground level class of array languages REC, which involves the projection of the languages belonging to the class of local languages (LOC). Mutual relationship between these new formalisms and also with LOC is analysed in [2].

Recently another picture generating mechanism, Array token Petri Net structure ATPNS [8], has been evolved from string generating Petri nets [1, 6]. Petri Net [9] is one of the formal models used for analyzing systems that are concurrent, distributed and parallel. Tokens are the elements used to simulate the dynamism of the Petri Net systems. The language generated by the Petri net is set of all feasible transition sequence in that net. In ATPNS model, the authors have used arrays as tokens in some of the places as initial configuration called marking and catenation rules as labels of transitions. The language generated is the set of all arrays created at final places of the net. This model along with a control feature called inhibitor arcs generate the same family of languages as generated by KAG, T0LG and P2DCFG with regular control. To increase the generative capacity of this model, adjunction rules are introduced and Adjunct Array Token Petri net systems (AATPNS) [7] is defined. This new model generates Table 1L languages and strictly included ATPNS family of Languages.

Since AATPNS has been only compared with classical formalisms, we try to study the comparison of this model with recent generating devices and also with LOC for the expressiveness with respect to its generating capacity.

This paper is organized in the following manner. In Sect. 2, basic definitions of various array grammars, Petri Nets and notions of Petri nets pertaining to arrays have been recalled. In Sect. 3, we recall the definition of adjunct array token Petri nets, in more generalized form and provide some illustrative examples. In Sect. 4, we compare this model with various classes of picture languages generated by recent grammars and also with class LOC, for the understanding of generative capacity of this model.

2 Preliminaries

The following definitions and notations are mainly from [5, 8, 10, 18].

2.1 Array Grammars

Definition 1 *Let J^{**} denotes the set of all arrays (pictures) over the elements of a finite set J and J^{++} denotes set of all non empty arrays over J. For l, m $\geq$ 0, $J^{(l,\, m)}$*

represents the set of arrays of size (l, m). *An array (picture) language is a subset of* J^{**}. *If* $p \in J^{**}$, *then* $p(i, j)$ *denotes a pixel in the place of ith row and jth column of p.* $|p|_{col}$ *denotes the number of columns of p and* $|p|_{row}$ *denotes the number of rows of p. A* $\varobar$ *B denotes a column catenation of the array A with array B which is defined when A and B have same number of rows. A* $\varobar$ *B denotes row catenation of A with B provided the number of columns of A and B are same. If* $x \in J^{**}$ *then* $(x)^n$ *(resp.* $(x)_m$*) denotes horizontal (resp. vertical) juxtaposition of m copies of x.*

Pure 2D context free grammars (P2DCFG) which make use of only terminal symbols have been recently studied [18]. *Pure 2D context-free grammars, unlike Siromoney matrix grammars* [17], *admit rewriting any row/column of pictures with no priority of columns and rows. Row/column sub-arrays of pictures are rewritten in parallel by equal length strings and by using only terminal symbols, as in a pure string grammar.*

Definition 2 *A pure 2D context-free grammar (P2DCFG) is a 4-tuple* $G = (\Sigma, P_c, P_r, \mathcal{M}_0)$, *where* Σ *is a set of symbols,* $P_c = \{t_{ci}/1 \le i \le m\}$, $P_r = \{t_{rj}/1 \le j \le n\}$.

Each t_{ci} $(1 \le i \le m)$, *called a column table, is a set of context free rules of the form* $a \rightarrow \alpha$, $a \in \Sigma$, $\alpha \in \Sigma^*$ *such that any two rules of the form* $a \rightarrow \alpha$, $b \rightarrow \beta$ *in* t_{ci}, *have* $|\alpha| = |\beta|$ *where* $|\alpha|$ *denotes the length of* α.

Each t_{rj} $(1 \le j \le n)$, *called a row table, is a set of context free rules of the form* $c \rightarrow \gamma^T$, $c \in \Sigma$, $\gamma \in \Sigma^*$ *such that any two rules of the form* $c \rightarrow \gamma^T$, $d \rightarrow \delta^T$ *in* t_{rj}, *have* $|\gamma| = |\delta|$.

$M_0 \subseteq \Sigma^{**} - \{\lambda\}$ *is a finite set of axiom arrays.*

Derivations are defined as follows. For any two arrays M_1, M_2, $M_1 \Rightarrow M_2$ *denotes that* M_2 *is obtained from* M_1 *by either rewriting a column of* M_1 *by rules of a column table* t_{ci} *in* P_c *or a row of* M_1 *by rules of a row table* t_{rj} *in* P_r. $\overset{*}{\Rightarrow}$ *is the reflexive transitive closure of* $\Rightarrow$.

The picture language L(G) generated by G is the set of rectangular picture arrays $\{M/M_0 \overset{*}{\Rightarrow} M \in \Sigma^{**}$, *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 3 *A pure 2D context-free grammar with a regular control (RP2DCFG) is* $G_c = (G, Lab(G), C)$, *where G is a pure 2D context-free grammar, Lab(G) is a set of labels of the tables of G and* $C \subseteq Lab(G)^*$ *is a regular string language. The words of* $Lab(G)^*$ *are called control words of G. Derivations* $M_1 \overset{w}{\Rightarrow} M_2$ *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 from* 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_c *with derivations controlled as described above. (R)P2DCFL denotes the family of all picture array languages generated by pure 2D context-free grammars with a regular control.*

The family P2DCFL is strictly included in family RP2DCFL [18].

Tiling Systems, a recognizing device for the class REC, uses Local languages as projections. A local language is defined by a set of 2×2 arrays (tiles).

Definition 4 *Let J be a finite alphabet, θ be a finite set of tiles over $J \cup \{\#\}$ The local language defined by $L = \{p \in J^{++} | [[\breve{p}]] \subseteq \Theta\}$ where $[[\breve{p}]]$ denotes the set of all tiles contained in p surrounded by a special symbol $\# \notin J$. The family of all local languages is denoted by LOC.*

Prusa Grammar device admits parallel application of rules so that non terminal symbols can be substituted with rectangular subpictures simultaneously. This model has more generative power than context free KAG [11].

Definition 5 *Prusa Grammar (PG) is a tuple (J, N, R, S), where J is the finite set of terminal symbols, disjoint from the set N of non terminal symbols; $S \in N$ is the start symbol; and $R \subseteq N \times (N \cup J)^{++}$ is the set of rules.*

Let G = (J, N, R, S) be a PG. We define a picture language L(G, A) over J for every $A \in N$. The definition is given by the following recursive descriptions:

1. *if $A \rightarrow w$ is in R, and $w \in \Sigma^{++}$, then $w \in L(G, A)$;*
2. *let $A \rightarrow w$ be a production in R, $w = (N \cup \Sigma)^{(m\ ,n)}$, for some m, $n \geq 1$, and $p_{i,\ j}$, with $1 \leq i \leq m$, $1 \leq j \leq n$, be pictures such that:*
   - (a) *if $w(i, j) \in \Sigma$, then $p_{i,\ j} = w(i, j)$;*
   - (b) *if $w(i, j) \in N$, then $p_{i,j} \in L(G, w(i, j))$;*
   - (c) *if $P_k = p_{k,1} \oplus p_{k,2} \oplus \ldots \oplus p_{k,n}$, for any $1 \leq i \leq m$, $1 \leq j \leq n$, $|p_{i,j}|_{col} = |p_{i+1,j}|_{col}$ and $P = P_1 \ominus P_2 \ominus \ldots \ominus P_m$; then $P \in L(G, A)$.*

The set L(G, A) contains exactly the pictures that can be obtained by applying a finite sequence of rules (1) *and* (2). *The language L(G) generated by grammar G is denoted as L(G, S).*

Tile Grammars (TG) [4] *perform an isometric derivation process for which homogeneous subpictures are replaced with isometric pictures of the local language defined by the right part of the rules.*

Definition 6 *A tile grammar (TG) is a tuple (J, N, S, R), where J is the terminal alphabet, N is a set of non terminal symbols, $S \in N$ is the starting symbol, R is a set of rules. Let $A \in N$. There are two kinds of rules:*

1. *fixed size: $A \rightarrow t$, where $t \in \Sigma$;*
2. *variable size: $A \rightarrow \omega$, ω is a set of tiles over $N \cup \{\#\}$.*

At each step of the derivation, an A-homogeneous sub picture is replaced with an isometric picture of the local language defined by the right part α of a rule $A \rightarrow \alpha$, where α admits a strong homogeneous partition. The process terminates when all non terminals have been eliminated from the current picture.

Regional Tile Grammars (RTG) [10] are the Tile Grammars with specified set of tiling.

Definition 7 *A homogeneous partition is regional* (*HR*) *iff distinct* (*not necessarily adjacent*) *subdomains have distinct labels. A picture p is regional if it admits a HR partition. A language is regional if all its pictures are so. A regional tile grammar* (*RTG*) *is a tile grammar* (*see Definition 6*)*, in which every variable size rule* $A \rightarrow \omega$ *is such that* $LOC(\omega)$ *is a regional language.*

2.2 Petri Nets

Definition 8 *Petri Net is one of the mathematical modeling tools for the description of distributed systems involving concurrency and synchronization. It is a weighted directed bipartite graph consisting of two kinds of nodes called places* (*represented by circles*) *and transitions* (*represented by bars*)*. Places represents conditions and transition represents events. The places from which a directed arc runs to a transition are called input places of the transition and the places to which directed arcs run from a transition are called output places. Places in Petri nets 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 the abstract sense, a transition of a Petri net may fire if it is enabled; when there are sufficient tokens in all of its input places.*

Definition 9 *A Petri net structure is a four tuple* $C = \langle Q, T, I, O \rangle$ *where* $Q = \{q_1, q_2, \ldots, q_n\}$ *is a finite set of places,* $n \geq 0$, $T = \{t_1, t_2, \ldots, t_m\}$ *is a finite set of transitions* $m \geq 0$, $Q \cap T = \phi$, $I{:}T \rightarrow Q^{\infty}$ *is the input function from transitions to bags of places and* $O{:}T \rightarrow Q^{\infty}$ *is the output function from transitions to bags of places.*

Definition 10 *An inhibitor arc from a place* q_l *to a transition* t_k *has a small circle in the place of an arrow in regular arcs. This means the transition* t_k *is enabled only if* q_l *has no tokens in it. In other words a transition is enabled only if all its regular arc input places have required number of tokens and all its inhibitor arc* (*if exists*) *input places have zero tokens.*

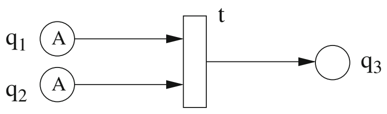
2.3 Array Token Petri Nets

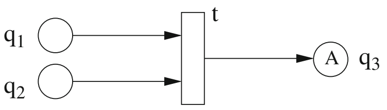
In the array generating Petri Net structure, arrays over an alphabet J are used as tokens in some input places.

Definition 11 *Row* (*resp. column*) *catenation rules in the form of* $A \ominus B$ (*resp.* $A \,\varobar\, B$) *can be associated with a transition t as a label, where A is a* $m \times n$ *array in the input place and B is an array language whose number of columns* (*resp. rows*) *depends on the number of columns* (*resp. rows*) *of A. Three types of transitions can be enabled and fired*

(1) *When all the input places of transition t (without label) having the same array as tokens*

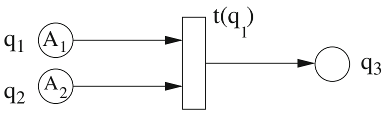
- *Each input place should have at least the required number of tokens (arrays)*
- *Firing t removes arrays from all its input places and moves the array to all its output places*

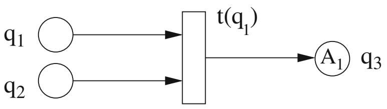




(2) *When all the input places of transition t have the different arrays as tokens*

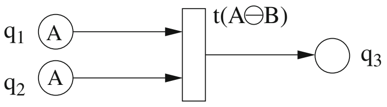
- *The label of t designates one of its input places which has sufficient number of same arrays as tokens*
- *Firing t removes arrays from all its input places and moves the array from the designated input place to all its output places.*

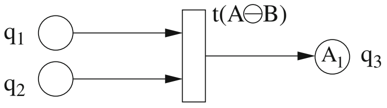




(3) *When all the input places of transition t (with row or column catenation rule as label) have the same array as tokens*

- *Each input place should have at least the required number of tokens (arrays)*
- *Firing t removes arrays from all its input places and creates the catenated array as per the catenation rule, in all its output places*





In all the three types, firing of a transition t is enabled only if all the input places corresponding to inhibitor arcs (if exist) does not have any tokens in it.

Definition 12 *An Array Token Petri net structure (ATPNS) is a five tuple* $N = \langle J, C, M_0, \rho, F\rangle$ *where J is a given alphabet,* $C = \langle Q, T, I, O\rangle$ *is a Petri net structure with tokens as arrays over J,* $M_0 : Q \to J^{**}$*, is the initial marking of the net,* $\rho{:}T \to L$*, a mapping from the set of transitions to set of labels of transitions and* $F \subset Q$*, is a finite set of final places.*

Definition 13 *If P is an ATPNS then the language generated by P is defined as* $L(P) = \{X \in J^{**}/X$ *is in the place q for some q in* $F\}$*. Starting with arrays (tokens) over a given alphabet as initial marking, all possible sequences of transitions are fired. Set of all arrays created in final places of F is called the language generated by Petri Net structure.*

3 Adjunct Array Token Petri Net Structure

In this section, we recall the notions of adjunct array token Petri net structure [7] in generalized form and give some examples.

Definition 14 *Adjunction is a generalization of catenation. In the row catenation* $A \ominus B$*, the array B is joined to A after the last row. But row adjunction can join the array B into array A after any row of A. Similarly column adjunction can join the array B into array A after any column of A. Let A be an* $m \times n$ *array in* J^{**} *called host array;* $B \subset J^{**}$ *be an array language whose members, called adjunct arrays have fixed number of rows. A row adjunct rule (RAR) joins an adjunct array B into a host array A in two ways : By post rule denoted by (A, B,* ar_j*), array B is juxtaposed into Array A after jth row and by pre rule denoted by (A, B,* br_j*), array B is juxtaposed into Array A before jth row. The number of columns of B is same as the number of columns of A. In the similar notion column adjunct rule (CAR) can also be defined in two ways : post rule (A, B,* ac_j*) and pre rule (A, B,* bc_j*) joining B into A, after jth column of A and before jth column of A respectively. It is obvious that a row catenation rule* $A \ominus B$ *in ATPNS is a post RAR rule (A, B,* ar_m*) and column catenation rule* $A \oplus B$ *is a post CAR rule (A, B,* ac_n*). Transitions of a Petri net structure can also be labeled with row or column adjunct rules.*

Definition 15 *An Adjunct Array Token Petri Net Structure (AATPNS) is a five tuple* $N = \langle J, C, M0, \rho, F\rangle$ *where J is a given alphabet,* $C = \langle Q, T, I, O\rangle$ *is a Petri net structure with tokens as arrays over J,* $M_0{:}Q \to J^{**}$*, is the initial marking of the net,* $\rho{:}T \to L$*, a mapping from the set of transitions to set of labels where catenation rules of the labels are either RAR or CAR and* $F \subset Q$*, is a finite set of final places.*

In AATPNS, the types of transitions which can be enabled and fired are similar to that of Definition 11 except the type (3) where labels of transitions may be RAR or CAR rules instead of row or column catenation rules.

When all the input places of a transition t (with RAR or CAR rule as label) have the same array as tokens.

Each input place should have at least the required number of tokens (arrays)

– *Firing t removes arrays from all its input places and creates the array through adjunction as per the RAR or CAR rule, in all its output places In all the three types, firing of a transition t is enabled only if all the input places corresponding to inhibitor arcs (if exists) does not have any tokens in it.*

Definition 16 *If P is an AATPNS then the language generated by P is defined as $L(P) = \{X \in J^{**}/X$ is in the place q for some q in F$\}$. Starting with arrays (tokens) over a given alphabet as initial marking, all possible sequences of transitions are fired. Set of all arrays created in final places F is called the language generated by AATPNS Petri Net structure.*

Example 1 Consider the Adjunct Array token Petri net structure $P_2 = \langle J, C, M_0, \rho, F\rangle$ where $J = \{0, 1\}$, $C = (Q, T, I, O)$, $Q = \{q_1, q_2\}$, $T = \{t_1, t_2\}$, $I(t_1) = \{q_1\}$, $I(t_2) = \{q_2\}$, $O(t_1) = \{q_2\}$, $O(t_2) = \{q_1\}$, M_0 is the initial marking: the array S is in q_1 and there is no array in q_2, $\rho(t_1) = (A, B_1, ac_n)$ and $\rho(t_2) = (A, B_2, ar_m)$ and $F = \{q_1\}$.

Where $S = \begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix}$, and $B_1 = (0)_m$ and $B_2 = (0)^{n-1}1$. Initially t_1 is the only enabled transition. Firing of t_1 adjoins a column of 0's after the last column of array S and puts the derived array in q_2, making t_2 enabled. Firing t_2 adjoins a row of 0's ending with 1 after the last row of the array in q_2 and puts the derived array in q_1. When the transitions t_1, t_2 fire the array that reaches the output place q_1 is shown as

$\begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix} \overset{t_1}{\Rightarrow} \begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{matrix} \overset{t_2}{\Rightarrow} \begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix}$. Firing the sequence $(t_1t_2)^2$ generates the output array as $\begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$. The language generated by Petri net is set of square pictures over {0, 1} with 1's in the main diagonal and other elements are 0's (Fig. 1).

Example 2 Consider the Adjunct Array token Petri net structure $P_1 = \langle J, C, M_0, \rho, F\rangle$ where $J = \{a\}$, $C = (Q, T, I, O)$, $Q = \{q_1, q_2\}$, $T = \{t_1, t_2\}$, $I(t_1) = \{q_1\}$, $I(t_2) = \{q_2\}$, $O(t_1) = \{q_2\}$, $O(t_2) = \{q_1\}$, M_0 is the initial marking: the array S is in q_1 and there is no array in q_2, $\rho(t_1) = (A, B_1, ac_n)$ and $\rho(t_2) = (A, B_2, ar_m)$ and $F = \{q_1\}$.

Where $S = a$, $B_1 = (a)_m$, $B_2 = (a)^n$. On firing the sequence $(t_1t_2)^n$ $n \geq 0$ set of all square pictures over a is generated (Fig. 2).

Example 3 Consider the Adjunct Array token Petri net structure $P_3 = \langle J, C, M_0, \rho, F\rangle$ where $J = \{a\}$, the Petri net structure is $C = (Q, T, I, O)$ with

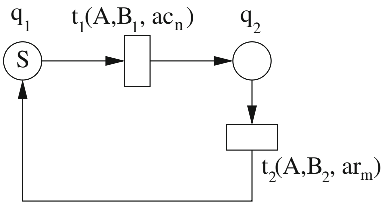


Fig. 1 Adjunct array token Petri net generating unit matrix pictures

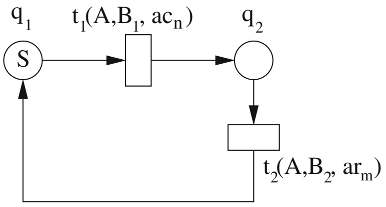


Fig. 2 AATPNS to generate square pictures of a's

$Q = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7\}$, $T = \{t_1, t_2, t_3, t_4, t_5, t_6\}$, $I(t_1) = \{p_1, p_2\}$, $I(t_2) = \{p_3\}$, $I(t_3) = \{p_4\}$, $I(t_4) = \{p_1, p_5\}$, $I(t_5) = \{p_5, p_6\}$, $I(t_6) = \{p_1, p_2\}$, $O(t_1) = \{p_3\}$, $O(t_2) = \{p_4\}$, $O(t_3) = \{p_2, p_5\}$, $O(t_4) = \{p_6\}$, $O(t_5) = \{p_1\}$, $O(t_6) = \{p_7, p_2\}$.

$\rho{:}T \to L$ is defined as follows: $\rho(t_1) = p_2$, $\rho(t_2) = (A, B_1, ac_n)$, $\rho(t_3) = (A, B_2, ar_m)$, $\rho(t_4) = \lambda$, $\rho(t_5) = \lambda$, $\rho(t_6) = \lambda$, $\rho(t_7) = \lambda$, $F = \{p_7\}$. The Petri net graph is given in Fig. 3. The arrays used are $S = \begin{matrix} a & a \\ a & a \end{matrix}$, $B_1 = (aa)_m$, $B_2 = \begin{pmatrix} a \\ a \end{pmatrix}^n$.

To start with only t_1 is enabled. Firing of sequence of transitions $t_1t_2t_3$ results in a square of a's of size 4 × 4 in p_2 and p_5. At this stage both t_6 and t_4 are enabled. Firing the sequence $t_1t_2t_3t_6$ puts a square of size 4 × 4 in p_7. Firing t_4 pushes the array to p_6, emptying p_5. In this position t_5 is enabled. Firing t_5 puts two copies of same array in p_1. Since at this stage there are two tokens in p_1, the sequence $t_1t_2t_3$ has to fire two times to empty p_1. The firing of sequence $t_4t_5(t_1t_2t_3)^2t_6$ puts a square of a's of size 8 × 8 in p_7. The inhibitor input p_1 make sure that a square of size 6 × 6 does not reach p_7. This AATPNS generates the language of squares of a's of size 2^n, $n \geq 1$.

Example 4 The AATPNS $P_4 = \langle J, C, M_0, \rho, F\rangle$ with $J = \{a, b\}$, $F = \{p\}$ given in Fig. 4, where

$$S \in \left\{ \begin{matrix} a & b & b & a \\ b & b & b & b' \end{matrix} \quad \begin{matrix} b & b & b & b \\ b & b & b & b' \end{matrix} \quad \begin{matrix} b & a & a & b \\ a & a & a & a' \end{matrix} \quad \begin{matrix} a & a & a & a \\ a & a & a & a \end{matrix} \right\},$$

$B_1 = (aa)_m$, $B_2 = (a)^n$, $B_3 = (bb)_m$, $B_4 = (b)^n$ generates the language of pictures composed by symmetrical L shaped strings of same character over the alphabet $J = \{a, b\}$.

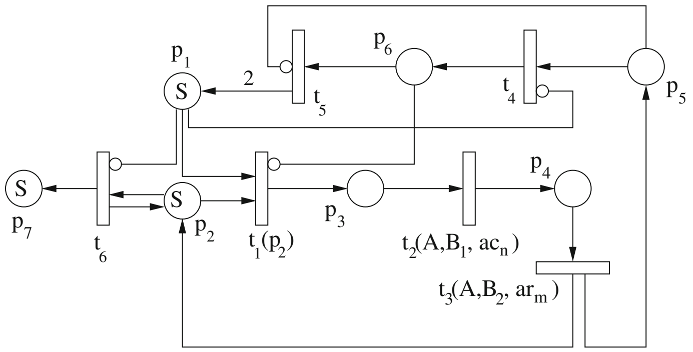


Fig. 3 AATPNS to generate square picture of *a*'s of size 2^n

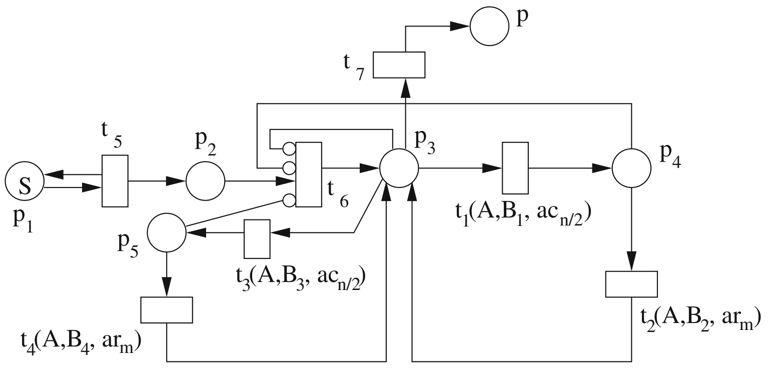


Fig. 4 AATPNS to generate symmetric *L* shaped strings

A typical picture in this language is $\begin{array}{cccccccc} a & b & a & a & a & a & b & a \\ b & b & a & a & a & a & b & b \\ a & a & a & a & a & a & a & a \\ a & a & a & a & a & a & a & a \end{array}$

4 Comparative Results

In the following results $\mathcal{L}(X)$ denotes the family of all languages generated by *X*.

Theorem 1 $\mathcal{L}(RP2DCFG) \subset \mathcal{L}(AATPNS)$

Proof In [8] the authors have proved that $\mathcal{L}(RP2DCFG) \subseteq \mathcal{L}(ATPNS)$. By the result $\mathcal{L}(ATPNS) \subset \mathcal{L}(AATPNS)$ [7], we have $\mathcal{L}(RP2DCFG) \subseteq \mathcal{L}(AATPNS)$. For

Fig. 5 A picture in the language L_{+b}

$$\begin{matrix} a & a & a & b & a \\ a & a & a & b & a \\ b & b & b & b & b \\ a & a & a & b & a \\ a & a & a & b & a \end{matrix}$$

strict inclusion the language generated by AATPNS in Example 3 cannot be generated by any RP2DCFG [2].

Theorem 2 *$\mathcal{L}(AATPNS)$ and $\mathcal{L}(RTG)$ are incomparable but not disjoint.*

Proof The language of squares over a, in Example 2, can be generated by an RTG $G = \langle J, N, S, R\rangle$, where $N = \{S, A, B, A', B', C\}$, $J = \{a\}$ and R consists of variable size rules:

$$S \to \left[\begin{bmatrix} \# & \# & \# & \# & \# \\ \# & S & S & A & \# \\ \# & S & S & A & \# \\ \# & B & B & C & \# \\ \# & \# & \# & \# & \# \end{bmatrix}\right], \quad A \to \left[\begin{bmatrix} \# & \# & \# \\ \# & A & \# \\ \# & A' & \# \\ \# & \# & \# \end{bmatrix}\right],$$

$$B \to \left[\begin{bmatrix} \# & \# & \# & \# \\ \# & B & B' & \# \\ \# & \# & \# & \# \end{bmatrix}\right]$$

Fixed size rules as $S, A, A', B, B' \to a$

Therefore $\mathcal{L}(AATPNS)$ and $\mathcal{L}(RTG)$ are non disjoint.

The language $L_{+\mathrm{b}}$ of pictures consists of a horizontal and a vertical string of b's (not in the border) in the background of a's can be generated by RTG [10]. A typical member of L_{+b} is given in Fig. 5. This language cannot be generated by any AATPNS, as the number of transitions in the net cannot depend on the size of the array. In an $m \times n$ array of a's a column of b's can be adjuncted in $n - 1$ ways and a row of b's can be adjuncted in $m - 1$ ways. To insert both a column of b's and a row of b's the net requires $(m - 1)(n - 1)$ transitions with corresponding adjunction rules. Hence it is not feasible to generate these arrays using AATPNS.

The language in Example 4 cannot be generated by any RTG grammar G [2].

Theorem 3 *$\mathcal{L}(AATPNS)$ and $\mathcal{L}(PG)$ are incomparable but not disjoint.*

Proof The language of squares over a, in Example 2 can also be generated by a Prusa Grammar [11]. Again the language L_{+b} in the proof of Theorem 2 can also be generated by a PG [11]. Since $\mathcal{L}(PG) \subset \mathcal{L}(RTG)$ [10], the language in Example 4 cannot be generated by Prusa grammar also.

Theorem 4 *$\mathcal{L}(AATPNS)$ and LOC are incomparable but not disjoint.*

Proof The language of squares over a, in Example 2 is not a local language [5]. In [2] the authors have proved that the language L_{diag} of pictures containing arbitrary number of diagonals of 1 (no single 1 are admitted at corners) that are separated by at least one diagonal of 0's is in LOC. But L_{diag} cannot be generated by any AATPNS. The only 2×2 array in the language is $\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$. But there are four 3×3 arrays with the property belonging to the language. To generate four 3×3 arrays from the start array we need 8 transitions with different array languages involved in the labels of the transitions. Hence it is impossible to construct a net with finite number of transitions. Finally the language of square pictures over $\{0, 1\}$ with 1's in the main diagonal and other elements are 0's in LOC [5] and it can also be generated by an AATPNS given in Example 1.

5 Conclusion

In this work, we consider Adjunct array token Petri net structure, recently introduced by Lalitha et al. [7]. We compare this model with recent context free array grammars and the class LOC. The future works concern the study of hierarchy of AATPNS with the other existing generating devices. The relationship of AATPNS with context free and context sensitive controlled pure two dimensional grammars is to be investigated.

References

1. Baker, H.G.: Petri Net Languages Computation Structures Group Memo 68, Project MAC, MIT, Cambridge, Massachusetts (1972)
2. Bersani, M.M., Frigeri, A., Cherubini, A.: On some classes of 2D languages and theire relations. In: IWCIA, pp. 222–234, (2011)
3. Cherubini, A., Crespi-Reghizzi, S., Pradella, M., Peitro, P.S.: Picture languages: Tiling systems versus tile rewriting grammars. Theor. Comput. Sci. **356**, 90–103 (2006)
4. Crespi-Reghizzi, S., Pradella, M.: Tile rewriting grammars and picture languages. Theor. Comput. Sci. **340**, 257–272 (2005)
5. Giammarresi, D., Restivo, A.: Two-dimensional langauges. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of formal languages. vol. 3, pp. 215–267. Springer Verlag, (1997)
6. Hack, M.: Petri net languages. Computation Structures Group Memo 124, Project MAC, MIT (1975)
7. Lalitha, D., Rangarajan, K., Thomas, D.G.: Adjunct array images using petri nets. Indian J. Math. Math. Sci. **8**(1), 11–19 (2012)
8. Lalitha, D., Rangarajan, K., Thomas, D.G.: Rectangular arrays and petri nets. In: IWCIA, pp. 166–180, (2012)
9. Peterson, J.L.: Petri Net Theory and Modeling of Systems. Prentice Hall Inc, Englewood Cliffs (1981)

10. Pradella, M., Cherubini, A., Crespi-Reghizzi, S.: A unifying approach to picture grammars. Inf. Comput. **209**, 1246–1267 (2011)
11. Prusa, D.: Two-dimensional Languages. Ph.D. Thesis, (2004)
12. Rosenfeld, A., Siromoney, R.: Picture languages—a survey. Languages of Design **1**(3), 229–245 (1993)
13. Siromoney, R.: Advances in array languages. In: Proceedings of the 3rd international wrokshop on graph grammars and their application to computer science. LNCS, Springer, vol. 291, pp. 549–563 (1987)
14. Siromoney, R., Siromoney, G.: Extended controlled table *L*-arrays. Inf. Control **35**, 119–138 (1977)
15. Siromoney, G., Siromoney, R., Kamala, K.: Array grammars and kolam. Comput. Graph. Image Process. **3**(1), 63–82 (1974)
16. Siromoney, G., Siromoney, R., Kamala, K.: Picture languages with array rewriting rules. Inf. Control **22**, 447–470 (1973)
17. Siromoney, G., Siromoney, R., Krithivasan, K.: Abstract families of matrices and picture languages. Comput. Graph. Image Process. **1**, 284–307 (1972)
18. Subramanian, K.G., Rosihan Ali, M., Geethalakshmi, M., Nagar, A.K.: Pure 2D picture grammars and languages. Discrete Appl. Math. **157**(16), 3401–3411 (2009)