A Comparative Study on Adjunct Array Token Petri Nets with Some Classes of Array Grammars

T. Kamaraj
Department of Mathematics, Sathyabama University, Chennai – 119
kamaraj_mx@yahoo.co.in

D. Lalitha
Department of Mathematics, Sathyabama University, Chennai – 119
lalkrish_24@yahoo.co.in

D.G. Thomas
Department of Mathematics, Madras Christian College, Chennai – 59
dgthomasmcc@yahoo.com

Copyright © 2013 T. Kamaraj, D. Lalitha and D.G. Thomas. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Abstract

Adjunct Array Token Petri Net Structures (AATPNS) are recently introduced rectangular picture generating devices which extended the Array token Petri net Structures. AATPNS generated context free and context sensitive kolam array languages and tabled 0L/1L languages. In this paper we compare AATPNS with some simple and expressive picture grammar models like extended pure 2D context-free grammars, Internal parallel contextual array grammars, indexed Siromoney matrix grammars and CR Image grammars for examining further generating capacity of this model.

Keywords: Petri nets, array token, adjunction, array grammars, picture 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 motivation of picture processing and pattern recognition tasks [7, 8, 11]. With the quest of syntactic techniques on digital picture patterns, many array generating grammar devices have been proposed. Siromoney Matrix Grammars (SMG) [11], Controlled Table L Array Grammars (T0LG / T1LG) [9], Kolam Array Grammars (KAG) [10] are some of the classical generating devices, which used sequential and parallel application of rewriting rules. Pure 2D context free grammars (P2DCFG) [14], Parallel Contextual Array Grammar (PCAG) [12] are some of the parallel rewriting grammars which make use of only terminal symbols as pure string grammars[14].

Recently another picture generating mechanism, Array Token Petri Net Structure (ATPNS) [4] has been evolved from string generating Petri nets [1,2,6]. Petri net [1,6] is one of the formal models used for analyzing system that are concurrent, distributed and parallel. In ATPNS, array tokens are used to simulate the dynamism of the net. To increase the generative power of this model, adjunction rules are introduced and Adjunct array token Petri net structure (AATPNS) [3] is defined. AATPNS generated same family of languages as generated by context-free and context-sensitive KAG, T0LG / T1LG and P2DCFG. To examine further generating power of this model, in this paper, we compare AATPNS with extended forms of pure 2D context free grammars and Siromoney Matrix grammars and also with a class of PCAG.

This paper is organized in the following manner. In section 2, we recall the definition of Adjunct array token Petri nets (AATPNS) in a general form and provide some illustrative examples. In section 3, we compare AATPNS with various array grammars with respect to the generative capacity.

2. Adjunct Array Token Petri Net Structure

In this section, we recall the notions of adjunct array token Petri net structure [3] in generalized form and give some examples.

Let J be a finite alphabet. A rectangular arrangement of elements over an alphabet J is called an array or a picture over J. The set of all arrays over J is denoted by $J^+$ and $J^* = J^+ \cup \lambda$ where $\lambda$ is the empty picture.

Adjunction is a generalization of catenation. In the row catenation $A \circ B$, 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. In the array generating Petri net structure,
arrays over an alphabet \( J \) are used as tokens in some input places.

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 \( j^{th} \) row and by pre rule denoted by \((A, B, br_j)\), array \( B \) is juxtaposed into Array \( A \) before \( j^{th} \) 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 \( j^{th} \) column of \( A \) and before \( j^{th} \) column of \( A \) respectively. It is obvious that a row catenation rule \( A \circ B \) in ATPNS is a post RAR rule \((A, B, ar_m)\) and column catenation rule \( A \Box B \) is a post CAR rule \((A, B, ac_n)\).

**Definition 2.1.**
An Adjunct Array Token Petri Net Structure (AATPNS) is a five tuple \( N = < J, C, M_0, \rho, F > \) where \( J \) is a given alphabet, \( C = < Q, T, I, O > \) is a Petri net structure with tokens as arrays over \( J \), \( M_0 : Q \rightarrow J^{**} \), is the initial marking of the net, \( \rho : T \rightarrow L \), 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 the earlier model [4] except the third type where labels of transitions may be RAR or CAR rules instead of row or column catenation rules.

**Definition 2.2.**
If \( P \) is an AATPNS then the language generated by \( P \) is defined as \( L(P) = \{X \in J^{**} / X \text{ is in the place } q \text{ for some } q \in F\} \). 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 2.1.**
Consider the Adjunct Array token Petri net structure \( P_1 = < J, C, M_0, \rho, F > \) 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 \( F = \{q_1\} \). The Petri net graph is given in Figure 1.
The arrays used in the net are defined as follows: \( S = \begin{array}{cc} a & a \\ a & a \end{array} \) and \( B_1 = (a)_m \) and \( B_2 = (a)^n \). Initially \( t_1 \) is the only enabled transition. Firing of \( t_1 \) adjoins a column of \( a \)'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 \( a \)'s 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
\[
\begin{array}{cccccc}
a & a & a & a & a & a \\
a & a & a & a & a & a \\
a & a & a & a & a & a \\
a & a & a & a & a & a \\
a & a & a & a & a & a \\
a & a & a & a & a & a \\
\end{array}
\]
Firing the sequence \( (t_1t_2)^2 \) generates the output array as
\[
\begin{array}{cccccc}
a & a & a & a & a & a \\
a & a & a & a & a & a \\
a & a & a & a & a & a \\
a & a & a & a & a & a \\
a & a & a & a & a & a \\
a & a & a & a & a & a \\
\end{array}
\]
The language \( L_1 \) generated by Petri net is the set of square pictures over \( \{a\} \).

Example 2.2.
Consider the Adjunct Array token Petri net structure \( P_2 = <J, C, M_0, \rho, F> \), where \( J = \{a\} \), the Petri net structure is \( C = (Q, T, I, O) \) with \( Q = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7\} \), \( T = \{t_1, t_2, t_3, t_4, t_5, t_6\} \), \( I(t_1) = \{p_1, p_2\} \), \( I(t_2) = \{p_3\} \), \( I(t_3) = \{p_4\} \), \( I(t_4) = \{p_1, p_5\} \), \( I(t_5) = \{p_5, p_6\} \), \( I(t_6) = \{p_1, p_2\} \), \( O(t_1) = \{p_3\} \), \( O(t_2) = \{p_4\} \), \( O(t_3) = \{p_2, p_3\} \), \( O(t_4) = \{p_6\} \), \( O(t_5) = \{p_1\} \), \( O(t_6) = \{p_7, p_2\} \). \( \rho : T \to L \) is defined as follows: \( \rho(t_1) = p_2, \rho(t_2) = (A, B_1, acn), \rho(t_3) = (A, B_2, ar_m), \rho(t_4) = \lambda, \rho(t_5) = \lambda, \rho(t_6) = \lambda, \rho(t_7) = \lambda \), \( F = \{p_7\} \). The Petri net graph is given in Figure 2.

![Figure 2. AATPNS with inhibitor arcs](image-url)

The arrays used are \( S = \begin{array}{cc} a & a \\ a & a \end{array} \), \( B_1 = (a)_m \), \( B_2 = (a)^n \). 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 \( 4 \times 4 \) 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 \( 4 \times 4 \) in \( p_7 \). Firing \( t_4 \) pushes the array to \( p_6 \), emptying
Comparative study on adjunct array token Petri nets

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)^2 t_6 puts a square of a's of size 8 \times 8 in p_7. The inhibitor input p_1 make sure that a square of size 6 \times 6 does not reach p_7. This AATPNS generates the language L_2 of square pictures of a's of size \((2^n, 2^n)\), \(n \geq 1\).

**Example 2.3.**

The AATPNS \(P_3 = \langle J, C, M_0, \rho, F \rangle\) with \(J = \{a, b, c\}, F = \{p\}\) given in Figure 4, where \(S = bca\), \(B_1 = b^{n/3}c^{n/3}a^{n/3}\), \(B_2 = a^{n/3}b^{n/3}c^{n/3}\), \(B_3 = c^{n/3}a^{n/3}b^{n/3}\), \(B_4 = m/3m/3m/3\) generates the language,

\[
L_3 = \left\{ \left( a^n b^n c^n \right)_m, \left( b^n c^n a^n \right)_m, \left( c^n a^n b^n \right)_m \right\}.
\]

A typical picture in \(L_3\) is given in Figure 3.

```
a a a a b b b b c c c
a a a b b b c c c
b b b c c c a a a
b b b c c c a a a
b b b c c c a a a
b b b c c c a a a
b b b c c c a a a
b b b c c c a a a
```

**Figure 3.** A picture in \(L_3\)

```
a a a a b b b b c c c
a a a a b b b b c c c
b b b c c c a a a
b b b c c c a a a
b b b c c c a a a
b b b c c c a a a
b b b c c c a a a
b b b c c c a a a
```

**Figure 4.** AATPNS for the language \(L_3\)
3. Comparative Results

In the results we make use of the following expressive array grammars for comparing with AATPNS. (i) Extended pure 2D context-free grammar (EP2DCFG) [14], (ii) Context-free right linear indexed right linear Siromoney matrix grammar ((CF:RIR) SMG) [13], (iii) CR image grammar (CRIG) [5] and (iv) Internal parallel contextual Array Grammars (IPCAG) [12]. The notation \( L(X) \) denotes the family of all languages generated by the array grammar X.

**Theorem 3.1.**
\( L(EP2DCFG) \) and \( L(AATPNS) \) are incomparable but not disjoint.

**Proof.**
Since \( L(P2DCFG) \subset L(EP2DCFG) \) [14, Theorem 1] and \( L(P2DCFG) \subset L(ATPNS) \) [4, Theorem 5] and we have \( L(EP2DCFG) \cap L(AATPNS) \) is a nonempty set. The language \( L = \{ (x^T)^n \square (y^T)^n, n \geq 1 \} \) where \( x \in a\Sigma^*_1 \) and \( y \in b\Sigma^*_1 \), where \( \Sigma_1 = \{a,b\} \), can be generated by the EP2DCFG, \( G = (\Sigma, P_c, P_r, P_t, S) \) where \( \Sigma = VUT \) and \( V = \{\bullet, u, v, e, f\} \), \( T = \{a, b\} \), \( P_c = \{c\} \), \( P_r = \{r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8, r_9\} \), \( S = \{uv, u \bullet v\} \) with \( c : \{\bullet \rightarrow u \bullet v, \bullet \rightarrow uv\} \)

\[
\begin{align*}
& r_1 : \{u \rightarrow a, v \rightarrow b\}, \quad r_2 : \{e \rightarrow a, f \rightarrow a\}, \quad r_3 : \{e \rightarrow a, f \rightarrow b\} \\
& r_4 : \{e \rightarrow b, f \rightarrow a\}, \quad r_5 : \{e \rightarrow b, f \rightarrow b\}, \quad r_6 : \{e \rightarrow a, f \rightarrow b\} \\
& r_7 : \{e \rightarrow a, f \rightarrow a\}, \quad r_8 : \{e \rightarrow b, f \rightarrow a\}, \quad r_9 : \{e \rightarrow b, f \rightarrow b\}
\end{align*}
\]

Since the language generated does not have any specific pattern in the arrangement of elements in each column of any picture, we conclude that it cannot generated by any AATPNS as the net cannot be described with finite number of transitions.

On the other hand the language \( L_2 \) in Example 2.2 cannot be generated by any EP2DCFG since a picture of size \((2^n, 2^n)\) cannot be derived from the picture of size \((2^{n-1}, 2^{n-1})\) because there should exists a procedure to apply rules exactly \(2(n-1)\) times on both rows and columns of the picture. But the number of applications of rules to be applied cannot be a function of the dimension of the picture. Hence the second part of the theorem is proved.

**Theorem 3.2.**
\( L(AATPNS) \) is incomparable with families \( L[(CF : RIR) SMG] \) and \( L[CRIG] \) but not disjoint.

**Proof.**
The language \( L = \{ (x^T)^n \square (y^T)^n, n \geq 1 \} \), where \( x \in a\Sigma^*_1 \) and \( b\Sigma^*_1 \), where
Comparative study on adjunct array token Petri nets

$\Sigma_1 = \{a, b\}$, which is shown to be not generated by AATPNS in the proof of Theorem 3.1, can be generated by a deterministic CR image grammar (CRIG) [5, Example 2]. Since for every deterministic CRIG grammar there exists an equivalent (CF : RIR) SMG [13, Theorem 2.7], $L \in L[(CF : RIR) SMG]$

The language $L_1$ of squares of a’s in Example 2.1 cannot be generated by any (CF : RIR) SMG or CRIG since in both the grammars, the vertical phase rules are in such a way that the number of rows of a picture is independent of number of columns. The language of $m \times n$ arrays ($m \geq 3$, $n \geq 3$) on $\{\cdot, X\}$ describing digitized I patterns (Figure 5).

```
   X X X X X
   . . X . .
   . . X . .
   . . X . .
   . . X . .
   X X X X X
```

Figure 5. Digitized I

can be generated by a CRIG [5, Example 1] and hence by (CF : RIR) SMG. This language can also be generated by an AATPNS, $P_4 = (J, C, M_0, \rho, F)$ where $J = \{X, \cdot\}$, $F = \{p\}$ given in Figure 6.

```
   t_3(A, B_2, a_{r_1})
   S
   p
   t_1(A, B_1, b_{c_1})
   q
   t_2(A, B_2, a_{c_n})
```

Figure 6. AATPNS for the language of I’s

where $S = X X X X X$, $B_1 = (\cdot)_{m-2}$, $B_2 = (\cdot)^{\frac{m}{2}} X (\cdot)^{\frac{m+1}{2}}$.

**Theorem 3.3.**

$L(AATPNS)$ and $L(IPCAG)$ are incomparable but not disjoint.

**Proof.**

The language of $m \times n$ arrays ($m \geq 2$, $n \geq 3$) on $\{\cdot, X\}$ describing digitized T patterns (Figure 7).

```
   X X X X X
   . . X . .
   . . X . .
   . . X . .
   . . X . .
   . . X . .
   . . X . .
```

Figure 7. Digitized T

can be generated by an IPCAG [12, Example 2.1] and also by an AATPNS. The same net structure given in Figure 6 can be used to generate T but replacing $S =
\[
X \ X \ X \quad \text{and} \quad B_1 = \ X \ \big(\cdot\big)_{m-1} \ X \ \big(\cdot\big)_{n-1} \ \big(\cdot\big)_{m-1} \quad \text{So,} \quad \mathcal{L}(\text{AATPNS}) \cap \mathcal{L}(\text{IPCAG}) \neq \emptyset.
\]

For incomparability: The AATPNS language \(L_3\) in Example 2.3 cannot be generated by any IPCAG, which was shown in [12, Corollary 3.1]. The language \(L_{+b}\) of pictures consisting of a horizontal and a vertical string of b’s (not in border) in the background of a’s can be generated by an IPCAG \(G\) whose construction is given as follows:

\[
G = (V, B, C_c, C_r, \phi_c, \phi_r) \quad \text{where} \quad V = \{a, b\}, \quad B = \{M_0\}, \quad M_0 = \begin{bmatrix} a & b & a \\ b & b & a \end{bmatrix},
\]

\[
\]

\[
\phi_c = \begin{bmatrix} \phi_{c_1} & \phi_{c_2} & \phi_{c_3} \\ \phi_{c_4} & \phi_{c_5} & \phi_{c_6} \end{bmatrix}, \quad \phi_r = \begin{bmatrix} \phi_{r_1} & \phi_{r_2} \\ \phi_{r_3} & \phi_{r_4} \end{bmatrix}
\]

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 required \((m-1)(n-1)\) transitions with corresponding adjunction rules. Hence it is not feasible to generate these arrays using AATPNS.

4. Conclusions

In this paper we have considered a variant class of Array token Petri net structure with adjunction rules as labels of transitions and compared it with some of the expressive grammar models: EP2DFCG, CRIG, (CF : RIR)SMG and IPCAG. The non empty intersection of the picture languages generated by this model with other models clearly suggest that AATPNS can generate a wide variety of digitized pictures and patterns. The application of this model in Picture processing tasks and Pattern recognition should be investigated further.
Comparative study on adjunct array token Petri nets

References


Received: October 24, 2013