ADJUNCT ARRAY IMAGES USING PETRI NETS

D. Lalitha, K. Rangarajan & D. G. Thomas

Abstract
Adjunct Array Token Petri Net Structure (AATPNS) with inhibitor arcs has been defined. AATPNS generate regular array languages. By introducing a control on the firing sequence, we show that, AATPNS with inhibitor arcs generate the context free and context sensitive array languages. Comparisons with certain existing array languages have been done.

AMS Classification: 68Rxx.

Keywords: Adjunction, Array token, Catenation, Inhibitor arcs, Petri net.

1. INTRODUCTION
Petri net is an abstract formal model of information flow [1]. 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 model to generate rectangular arrays has already [3,4] been introduced. Here we introduce a variation of the model introduced in [3]. In this model arrays over a given alphabet are used as tokens in the places of the net. Catenation rule is used as labels of transitions. Firing a sequence of transitions starting from a specific initial marking leading to a finite set of terminal markings would concatenate arrays and move the array to the final set of places. To increase the generative capacity we have introduced the concept of adjunction and also a control on the firing sequence. Inhibitor arcs are used to control the firing of transitions in the Petri net. This model is called Adjunct Array Token Petri Net Structure with inhibitor arcs.

2. BASIC DEFINITIONS
In this section we give basic definition of Petri Nets and notations used.

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 \( m \geq 0 \), \( P \cap T = \emptyset \), \( I : T \rightarrow P^c \) is the input function from transitions to bags of places and \( O : T \rightarrow P^c \) 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:**
\( \Sigma^* \) denotes the arrays made up of elements of \( \Sigma \). If \( A \) and \( B \) are two arrays having same number of rows then \( A \Theta B \) is the column wise catenation of \( A \) and \( B \). If two arrays have the same number of columns then \( A \Theta B \) 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 \( x \in \Sigma^* \), \( (x)^n + 1 = (x)^n \Theta x \) and \( (x)^n = (x)^n \). We use \( \oplus \) to denote either \( \ominus \), or \( \oplus \).

**3. ADJUNCT ARRAY TOKEN PETRI NET**
In this section we define adjunct rules, adjunct array Petri net structure, AATPNS with inhibitor arcs and give examples.

Let \( A \) be an array in \( \Sigma^* \) of size \( m \times n \) called the host array. Two types of Adjunct Array Rules are defined.

**Row Adjunct Rule (RAR):** For \( B \) an array language whose number of rows is fixed \( k \) and number of columns is a variable dependent on ‘n’ the number of columns of the host array, the joining of the array \( B \) to \( A \) is done in two ways.

(i) **Post-Rule:** This rule is a triplet of the form \( (A, B, ar) \) where \( 1 \leq j \leq m \). This rule joins the Adjunct Array \( B \) to the array \( A \) after the \( j \)th row.

(ii) **Pre-Rule:** This rule is a triplet of the form \( (A, B, br) \) where \( 1 \leq j \leq m \). This rule joins the Adjunct Array \( B \) to the array \( A \) before the \( j \)th row.

RAR can be used only if the number of columns of the arrays \( A \) and \( B \) coincide. The resultant array is called the derived array.

**Column Adjunct Rule (CAR):** For \( B \) an array language whose number of columns is fixed \( k \) and number of rows is a variable dependent on ‘m’ the number of rows of the host array, the joining of the array \( B \) to \( A \) is done in two ways.
(i) **Post-Rule:** This rule is a triplet of the form \((A, B, ac)\) where \(1 \leq j \leq n\). This rule joins the Adjunct Array \(B\) to the array \(A\) after the \(j^{th}\) column.

(ii) **Pre-Rule:** This rule is a triplet of the form \((A, B, bc)\) where \(1 \leq j \leq n\). This rule joins the Adjunct Array \(B\) to the array \(A\) before the \(j^{th}\) column.

CAR can be used only if the number of rows of the arrays \(A\) and \(B\) coincide. The resultant array is called the derived array.

**Firing Rules:** If a transition of a Petri Net is labeled with an Adjunct Array Rule (RAR/CAR) then all the input places should contain at least the required number of tokens of the same array \(A\) (host array). When the transition fires the host array is joined to the adjunct array \(B\) as per the rule and the derived array is put in the output place.

The following example explains the effect of firing transitions with RAR as label

**The array before firing the transition:**

\[
\begin{align*}
&P_1 \quad (A, B, ar_2) \\
&P_2 \quad (A, A)
\end{align*}
\]

If \(A = \begin{pmatrix} x & y \\ x & y \end{pmatrix}, B = \begin{pmatrix} a^n & b^n \\ a & b \end{pmatrix} \cdot \begin{pmatrix} a^n & b^n \\ a & b \end{pmatrix} \cdot \end{align*}

After firing the transition the array changes its position and size:

\[
\begin{align*}
&P_1 \quad (A, B, ar_2) \\
&P_2 \quad (A, A)
\end{align*}
\]

where \(A_1 = \begin{pmatrix} a & b \\ a & b \end{pmatrix} \cdot \begin{pmatrix} a & b \\ x & y \end{pmatrix} \cdot \end{align*}

\(A_1\), the host array is of size \(3 \times 2\). Since ‘\(n\)’ the number of columns of \(A\) is 2, \(B\) the adjunct array has a column of \(a\)’s and a column of \(b\)’s. When the transition \(t_1\) is fired \(B\) is joined to \(A\) after the second row of \(A\). The derived \(A_1\) is put in place \(p_2\).
The following example explains the effect of firing transitions with CAR as label.

**The array before firing the transition:**

\[
\begin{array}{c}
P_1 \quad A \quad A \\
p_2 \\
t_1 \quad \text{(A, B, bc}_n\text{)} \\
A_1 \quad (A, B, bc}_n\text{)} \\
\end{array}
\]

\[
x \quad y \quad (a \ a)^m_3
\]

If \(A = x \quad y\) and \(B = (b \quad b)\).

\[
x \quad y \quad (a \ a)^m_3
\]

**After firing the transition the array changes its position and size:**

\[
\begin{array}{c}
P_1 \quad A \\
p_2 \\
t_1 \quad \text{(A, B, bc}_n\text{)} \\
A_1 \quad (A, B, bc}_n\text{)} \\
\end{array}
\]

\[
x \quad a \quad a \quad y
\]

where \(A_1 = x \quad b \quad b \quad y\).

\[
x \quad a \quad a \quad y
\]

\(A\), the host array is of size \(3 \times 2\). Since \(m = 3\), \(B\) has a row of \(a\)’s followed by a row of \(b\)’s and another row of \(a\)’s. Since \(n = 2\), \(B\) is joined before the second column of \(A\). \(P_2\) is an inhibitor input and so the transition is enabled only if \(p_2\) does not contain any array. The derived array \(A_1\) is put in place \(p_3\).

**Definition 4:** (Adjunct Array Token Petri Net Structure (AATPNS).)

If \(C = (P, T, I, O)\) is a Petri net structure with arrays over of \(\Sigma^*\) as initial markings, \(M_0 : P \rightarrow \Sigma^*\), at least one of the transitions is labeled by the Adjunct array rules (CAR or RAR) and a finite set of final markings \(F\), then the Petri Net is defined as Adjunct Array Token Petri Net Structure.

**Definition 5:** (Adjunct Array Token Petri Net Structure with inhibitor arcs.)

An Adjunct Array Token Petri Net Structure with at least one inhibitor arc is defined as Adjunct Array Token Petri Net structure with inhibitor arcs.
**Definition 6:** If \( C = (P, T, I, O) \) is an AATPNS with or without inhibitor arcs then the Petri Net language generated by \( C \) is defined as \( L(C) = \{ [a_{ij}] \in \Sigma^{*} / [a_{ij}] \text{ is in } p \text{ and } p \in F \} \).

With an array present in the start place as token, all possible sequence of transitions are fired. The set of all derived arrays which reach the final place is the language generated by the Petri Net. Example 1 is a AATPNS without inhibitor arc and Example 2 is a AATPNS with inhibitor arcs.

**Example 1:**

\[
\Sigma = \{x, \bullet\} \quad B_1 = \left( \begin{array}{c} x \\ x \end{array} \right)^n, \quad B_2 = \left( \begin{array}{c} \bullet \end{array} \right)_{m-1} \quad S = x \quad x \quad x \quad \text{and} \quad F = \{p_1\}
\]

The firing sequence \( t_1 t_2 t_3 \) generates a diamond of size \( 5 \times 5 \)

Thus the firing sequence \( (t_1 t_2 t_3)^n \) generates a diamond of size \( 2n + 3 \). The language generated by the AATPNS in Example 1 can be generated by a tabled 1L array grammar with regular control.
Example 2: $\Sigma$, $B_1$ and $B_2$ are defined exactly like in example 1 and

$S = x \ x \ x \ x \ x$ with $F = \{p_s\}$.

To start with only transition $t_1$ is enabled. So the sequence $t_1t_2t_3t_4$ can be fired. Transition $t_1$ pushes the start array $S$ to $p_3$. Transition $t_2$ joins two rows after the $(m+1)/2$th row. Transition $t_3$ joins a column before the first column and $t_3$ joins a column after the last column. This results in a diamond of size $7 \times 7$ in $p_2$ and $p_6$. At this stage both $t_5$ and $t_6$ are enabled. Hence the firing sequence $t_1t_2t_3t_4t_7$ puts a diamond of size $7 \times 7$ in $p_8$.

$S \rightarrow t_1t_2t_3t_4t_7$

Since $t_5$ is enabled firing $t_5$ pushes this array to $P_7$, emptying $p_6$. Firing $t_6$ puts two arrays in $p_1$. Since there are two tokens in $p_1$ the sequence $t_1t_2t_3t_4$ has to be fired two times to empty $p_1$. Hence the firing sequence $t_5t_6t_1t_2t_3t_4t_7$ puts a diamond of size $11 \times 11$ in $p_8$. In general the firing sequence $t_5^{n-1}t_6^{n-1}(t_1t_2t_3t_4)^{n}t_7$ puts a diamond of size $2^{n+1}+3$ in $p_8$. This language can be generated by a Table 1L array grammar with context sensitive control.
4. COMPARATIVE RESULTS

In this section we recall the definition of Extended Controlled Table L-array Grammar [2] and compare it with AATPNS.

**Definition 7:** An extended, controlled \(<k_l, k_r, k_u, k_d>\) table L-array grammar is a 5-tuple \(G = (V, T, P, C, S, #)\) where \(V\) is a finite nonempty set; \(T \subseteq V\) is the terminal alphabet of \(G\); \(P\) 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 \(P\); and \(S \in V\) 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 \(C = P^*\), 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 \(\gamma\) control; \(X\) may be 0 or 1 and \(\gamma\) may be \(R\), \(CF\) or \(CS\).

**Theorem 1:** Any \((R) TXLAL\) can be generated by AATPNS.

**Proof:** Let \(G = (V, T, P, C, S, #)\) be an ECTL array grammar with regular control. Let us assume that in \(G\) the Array \(M_2\) can be derived from the array \(M_1\) where

\[
M_1 = \begin{bmatrix}
    a_{11} & \cdots & a_{1n-1} & a_{1n} & a_{21} & \cdots & a_{2n-1} & w_1n \\
    \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
    a_{m1} & \cdots & a_{mn-1} & a_{mn} & a_{m1} & \cdots & a_{mn-1} & w_{mn}
\end{bmatrix}
\]

with \(a_{ij} \in V\) and \(w_{in} \in V^*, i = 1, \ldots, m, j = 1, \ldots, n\). Then a right table \(R\) in \(P\) contains a rule of the form \(<a_i, #, a_u, a_d, a_{in}, w_{in}>\), with \(|w_{in}|\) the same for \(i\) in \(\{1, \ldots, m\}\), such that

(i) \(a_i = a_{i-1n}\)
(ii) if \(i > 1\) then \(a_u = a_{i-1n}\) and
(iii) if \(i < m\) then \(a_d = a_{i+1n}\).

This derivation can be achieved in AATPNS as follows. For \(i = 1, \ldots, m\), if \(w_{in} = a_{in} w_i\) where \(w_i \in V^+\) then define \(B\) as the array \(w_1 \ldots w_m\) so that \(M_2 = M_1 \oplus B\). Hence the column concatenation rule \(A o, B\) or equivalently the CAR \((A, B, ac_n)\), where \(A = M_1\) is the host array, can be given as a label to a transition. Firing the transition will put \(M_2\) in the output place. On the other hand if \(w_{in} = w_i a_{in}\), then the
CAR \((A, B, bc_n)\), where \(A = M_1\) is the host array, can be given as a label to a transition. Firing the transition will put \(M_2\) in the output place. Similarly for every left, up, down table we can have transitions with corresponding CAR or RAR labels. If \(C\) is a regular control then \(C\) is of the form \((P_1 \ldots P_n)^x\). For every array \(M \subseteq M(G)\), there exist a positive integer \(k\) such that the productions \(P_1 \ldots P_n\) is applied \(k\) times to derive \(M\) from \(S\). Let \(S\) be a token in place \(p_1\), have a transition \(t_1\) with either CAR or RAR as a label corresponding to \(P_1\), input place \(p_1\) and output place \(p_2\). Have a transition \(t_2\) with either CAR or RAR as a label corresponding to \(P_2\), input place \(p_2\) and output place \(p_3\) and so on. Have a transition \(t_n\) with either CAR or RAR as a label corresponding to \(P_n\), input place \(p_n\) and output place \(p_1\). Firing the sequence of transition \((t_1 t_2 \ldots t_n)^k\) we can generate the array \(M\) from \(S\).

**Theorem 2:** Any (\(T\)) TLAL where \(\gamma\) is either CF or CS can be generated by AATPNS with inhibitor arcs.

**Proof:** Let \(G = (V, T, P, C, S, \#)\) be a table 1L array grammar with context-free control, where \(P\) is a finite set of tables \(\{P_1, P_2, \ldots, P_k\}\), \(C = (P_1 \ldots P)^x(P_j \ldots P_k)^y\) 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 \(p_i\) belong to \(C_i\) with the start array \(S\) as a token. Have a transition \(t_1\) with either CAR or RAR, which corresponds to \(P_1\), as label. \(p_1\) is the input place and \(p_2\) is the output place of \(t_1\). Have a transition \(t_2\) with either CAR or RAR, which corresponds to \(P_2\), as label. \(p_2\) is the input place and \(p_3\) is the output place of \(t_2\). Continuing like this have a transition \(t_i\) with either CAR or RAR, which corresponds to \(P_i\), as label. \(p_i\) is the input place and \(p_1, M_1\) are the output places of \(t_i\). The subnet \(C_i\) can be executed any number of times. The sequence \((t_1 t_2 \ldots t_n)^k\) would put \(n\) different arrays as tokens in \(M_1\). But the place \(p_i\) will have the array which is the array that would result in applying the tables \(P_1 \ldots P_n\) \(n\) times to \(S\). Once \(t_i\) 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 \(C_i\) was executed). Similar to \(C_1\) in \(C_2\) there is a transition for every table \(P_j \ldots P_k\). Whenever \(C_2\) is executed once an array is put in \(M_2\) and \(p_1\). 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 \( t_1 \ldots t_i \) and \( t_j \ldots t_k \) 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.

5. CONCLUSION

The Adjunct Array Token Petri Net Structure has been defined to generate rectangular arrays [3]. This model is able to generate the tabled 1L languages with regular control [2]. We show that by introducing inhibitor arcs to AATPNS, tabled 1L languages with context-free or context-sensitive control can also be generated.

REFERENCES


D. Lalitha
Department of Mathematics,
Sathyabama University,
Chennai-119, India.

K. Rangarajan
Department of MCA,
Bharath University,
Chennai-73, India.

D. G. Thomas
Department of Mathematics,
Madras Christian College,
Tambaram, Chennai-59, India.