

CHAPTER 1

FINITE ARRAY AUTOMATA AND REGULAR ARRAY GRAMMARS

Adrian Atanasiu

*Faculty of Mathematics, Bucharest University,
Str. Academiei 14, sector 1, 70109 Bucharest, Romania
E-mail: aadrian@pcnet.ro*

K. G. Subramanian

*Department of Mathematics, Madras Christian College,
Tambaram, Chennai 600 059, India
E-mail: kgsmani1948@yahoo.com*

K. Rangarajan

*Department of Mathematics, Bharath Institute of Higher Education,
Selaiyur, Chennai 600 059, India*

P. S. P. Wang

*College of Computer Science, Northeastern University,
Boston, MA 02115, USA
E-mail: pwang@ccs.neu.edu*

A recognition device, called Finite Array Automaton, accepting a class of picture arrays, is introduced and it is shown that this device is equivalent to the n -dimensional regular array grammar. Regular (string) languages, called spreading languages, are associated to the corresponding n -dimensional array regular languages in order to deal with certain decision problems. Also the effect of controlling the application of rules of regular array grammars is brought out.

1. Introduction

Picture languages generated by array grammars or accepted by array automata have been studied by researchers and various models have been

proposed in the literature, motivated by problems arising in the framework of syntactic methods of pattern recognition and image processing.^{6,8,9} Freund⁴ has made an extensive and deep study of array grammars in a general setting of n -dimensions. Many different aspects of these grammars such as regulated rewriting,^{2,3} cooperating systems,⁵ contextual features⁴ and so on have been investigated. k -head finite array automata,¹ have also been considered to characterize certain families of array languages.

In this paper, an explicit construction of a recognition device, called a Finite Array Automaton equivalent to an n -dimensional Regular Array Grammar is made. A class of regular string languages, called Spreading languages is associated to the finite array automaton which is useful in certain decision problems. In the case of two dimensions ($n = 2$), the effect of controlling the application of rules of a Regular array grammar is brought out.

2. Preliminaries

For notions of formal languages we refer to Refs. 7 and 10; for basic notions, notations and results about array grammars to Ref. 4.

Let V be a finite and nonempty alphabet. Let Z denote the set of integers and N denote the set of positive integers and let $n \in N$. For $x = (x_1, x_2, \dots, x_n) \in Z^n$, we shall define

$$\|x\| = \sum_{i=1}^n x_i^2.$$

A n -dimensional array A over an alphabet V is a function $A : Z^n \rightarrow V \cup \{\#\}$ with finite support, $\text{supp}(A)$, defined by

$$\text{supp}(A) = \{u \in Z^n \mid A(u) \neq \#\};$$

is called the blank symbol, which is not in V . Usually we write

$$A = \{(u, A(u)) \mid u \in \text{supp}(A)\}.$$

In each location $u \in Z^n$ of the grid an element from $V \cup \{\#\}$ is placed by the function $A : Z^n \rightarrow V \cup \{\#\}$. Moreover, the set

$$\text{supp}(A) = \{u \in Z^n \mid A(u) \neq \#\}$$

is finite and nonempty. We require that for any $u \in \text{supp}(A)$, $A(v) \neq \#$ for at least one v with $\|u - v\| = 1$.

The set of all n -dimensional arrays over V is denoted by V^{*n} . Any subset of V^{*n} is called a n -dimensional array language.

Let $u \in Z^n$. Then the translation $\tau_u : Z^n \longrightarrow Z^n$ is defined by $\tau_u(v) = v + u$ for all $v \in Z^n$; for any array $A \in V^{*n}$ we define $\tau_u(A)$, the corresponding n -dimensional array translated by u , by

$$\tau_u(A(v)) = A(v + u), \quad \forall v \in Z^n.$$

The vector $(0, 0, \dots, 0) \in Z^n$ shall be often denoted by Ω_n .

Usually, arrays are regarded as equivalence classes with respect to linear translations, i.e. only the relative positions of the symbols from $\text{supp}(A)$ are taken into account.

The equivalence class $[A]$ of an array $A \in V^{*n}$ is defined by

$$[A] = \{B \in V^{*n} \mid \exists u \in Z^n, B = \tau_u(A)\}.$$

For any element $u \in \text{supp}(A)$, we define the frame

$$W_u = \{(u, v) \mid v \in Z^n, \|u - v\| = 1\}.$$

If $u = \Omega_n$, then the frame $W_0 = \{(\Omega_n, v) \mid \|v\| = 1\}$ is called the initial frame. Obviously, W_0 has $2n$ elements.

Let us consider the operation of translation $\tau_x(W_u) = \{(u+x, v+x) \mid v \in Z^n\} = W_{u+x}$. Then, any frame can be obtained by the translation of the initial frame: $W_u = \tau_u(W_0)$ (or $W_0 = \tau_{-u}(W_u)$).

A n -dimensional array production over V is a triple $P = (W, A_1, A_2)$ where $W \subset Z^n$ is a finite set and A_1, A_2 are mappings from W to $V \cup \{\#\}$.

In such a writing, all positions in W together with their associated symbols must be listed for representing A_1 and A_2 , by

$$A_i = \{(u, A_i(u)) \mid u \in W\}, \quad 1 \leq i \leq 2.$$

This representation is general, for the infinite set of equivalent n -dimensional array productions of the form $(\tau_u(W), \tau_u(A_1), \tau_u(A_2))$ with $u \in Z^n$. Hence without loss of generality, it can be assumed that $\Omega_n \in W$. Moreover, the set W can be omitted because it can be uniquely reconstructed from the description of the mappings A_1, A_2 . A n -dimensional array production $p = (W, A_1, A_2)$ is regular if:

- (1) $W = \{\Omega_n, u\} \subset Z^n, \|u\| = 1$ and $A_1 = \{(\Omega_n, B), (u, \#)\},$
 $A_2 = \{(\Omega_n, a), (u, C)\}, B, C \in V_N, a \in V_T,$ or
- (2) $W = \{\Omega_n\}, A_1 = \{(\Omega_n, B)\}, A_2 = \{(\Omega_n, b)\},$ where $B \in V_N, b \in V_T.$

If B_1, B_2 are two n -dimensional arrays, we shall write $B_1 \Rightarrow B_2$ if and only if there exist $u \in Z^n$ and a production $p = (W, A_1, A_2)$ such that the restrictions of B_i to $\tau_u(W)$ are $A_i (i = 1, 2)$.

In other words, the array $B_2 \in V^{*n}$ is directly derivable from the array $B_1 \in V^{*n}$ by the n -dimensional production (W, A_1, A_2) if and only if the subarray of B_1 corresponding to A_1 is replaced by A_2 , yielding B_2 .

A n -dimensional array grammar is a quintuple $G = (n, V_N, V_T, P, \{(v_s, S)\}, \#)$, where

- V_N is the alphabet of nonterminal symbols, V_T is the alphabet of terminal symbols, $V_N \cap V_T = \phi$;
- P is a finite non-empty set of n -dimensional array productions over $V_N \cup V_T$;
- $\{(v_s, S)\}$ is the start array ($S \in V_N$ is the start symbol, $v_s \in Z^n$ is the start location).

The array $B_2 \in V^{*n}$ is directly derivable from the array $B_1 \in V^{*n}$ in G , denoted $B_1 \Rightarrow_G B_2$ if and only if there exists a n -dimensional array production $p = (W, A_1, A_2)$ in P such that $B_1 \Rightarrow B_2$. If \Rightarrow^* is the reflexive and transitive closure of \Rightarrow_G , then the array language generated by G is defined by

$$L(G) = \{A \in V_T^{*n} / (v_s, S) \Rightarrow^* A\}.$$

The corresponding n -dimensional array language of equivalence classes with respect to linear translation is $[L(G)] = \{[A] / A \in L(G)\}$.

The n -dimensional array grammar G is called regular if every production in P is regular; in this case $L(G)$ is called a n -dimensional regular array language. The family of n -dimensional regular array languages will be denoted by $L(n, \text{reg})$ and the family of regular array languages of equivalence of classes of arrays will be denoted by $[L(n, \text{reg})]$.

3. Finite Array Automata

Let V be a finite and non-empty alphabet. Let $V^1 = \{a_1 / a \in V\}$ be another alphabet, distinct from V . The set of strings over $V(V^1)$ is denoted as usual by $V^*(V^{1*})$ respectively) with the difference that the identity will be considered $\#$ (blank symbol).

Definition 1: Let A be a n -dimensional array over the alphabet V . A n -Finite Array Automaton (n -FAA in short) is a 7-tuple, $M = (n, Q, V, \delta, q_o, v_0, F)$,

- $n \geq 1$.
- Q and V are finite and nonempty sets of “states” and “input characters” respectively.
- $\delta : (Q \times Z^n) \times V \rightarrow 2^{Q \times Z^n}$ is the transition map, satisfying the invariance property $(p, v) \in \delta((q, u), a) \Leftrightarrow (p, \tau_x(v)) \in \delta(q, \tau_x(u), a), \forall x \in Z^n$.
- $q_0 \in Q$ is the “initial state”.
- $v_0 \in Z^n$ is the start location; if $v_0 = \Omega_n$ this element can be ignored.
- $F \subseteq Q, (F \neq \phi)$ is the set of “final states”.

$(p, v) \in \delta((q, u), a)$ is defined only if the following constraint is fulfilled:

$$p, q \in Q, u, v \in Z^n, \|u - v\| = 1, a \in V.$$

We interpret these restrictions in the following way: if the automaton M is in the state q and finds a symbol $a \in V$ in the location u , then it will pass on to a state p and in another location v in its neighborhood.

Otherwise $\delta((q, u), a) = \phi$.

After such a rule is applied relating to a location u , the element $a \in V$ from this location is replaced by $a^1 \in V^1$ (in order to avoid the possibility of the automaton passing a second time through the location u).

The content of the location v is not important at this moment, but we remark that δ cannot be applied if in the current location u there is $\#$ or an element from V^1 .

Remark 1: $(p, -u) \notin \delta((q, u), a)$ unless $u = \Omega_n$ or $p \in F$ (otherwise this rule cannot be applied).

The property of invariance assures a homogenous behaviour of the transition map δ on the grid Z^n . Therefore it is enough to define the transition map of a n -dimensional finite array automaton only for the initial frame:

$$(p, v) \in \delta((q, u), a) \Leftrightarrow (p, v - u) \in \delta((q, \Omega_n), a).$$

In the following we shall consider, without loss of generality, only the case $v_0 = \Omega_n$.

The transition map δ can be extended recursively to $\delta^\wedge : (Q \times Z^n) \times V^* \rightarrow 2^{Q \times Z^n}$ as follows:

$$\delta^\wedge((q, u), \#) = \{(q, u)\}, \forall q \in Q, u \in Z^n,$$

$$\delta^\wedge((q, u), \alpha a) = \cup_{(p, v) \in \delta^\wedge((q, u), \alpha)} \delta((p, v), a), \forall p, q \in Q, a \in V, \alpha \in V^*.$$

For $\alpha = \#$ we obtain $\delta^\wedge((q, u), \#a) = \delta((q, u), a)$; so, δ^\wedge is a natural extension of δ because the invariance assures that $\#a = a$. In the following we shall denote δ^\wedge also by δ .

Proposition 1: $\delta((q, u), \alpha\beta) = \delta((q, u), \alpha), \beta$.

Proof: Straightforward to prove, by induction on the length of β . Now, we give the definition of the array language accepted by a $n - FAAM = (Q, V, \delta, q_0, V_0, F)$: $L(M)$ contains all the sequences $(v_0, a_0), (v_1, a_1), \dots, (v_n, a_n)$ which satisfy the constraints:

- (1) $\exists q_1, \dots, q_{n+1} \in Q$ with $(q_{i+1}, v_{i+1}) \in \delta((q_i, v_i)a_i), 0 \leq i \leq n$;
- (2) $(v_i, v_{i+1}) \in W_{v_i}, 0 \leq i \leq n$;
- (3) $q_{n+1} \in F$.

Let us denote by $\tau_x(L(M))$ the language accepted by the same automaton but with the start location $v_0 + x$ and let $L_0(M) = \tau_{-v_0}(L(M))$ Then the general language accepted by the automaton $M = (Q, V, \delta, q_0, v_0, F)$ is

$$[L(M)] = \bigcup_{u \in Z^n} \tau_u(L_0(M)).$$

Let $pro : Z^n \times V \rightarrow V$ be the function projection, defined by $pro((u, a)) = a, \forall a \in V, u \in Z^n$. This function can be extended to a morphism; now, another language, which is a projection in a 1-dimensional array of the language $L_0(M)$, can be defined:

$$pro(L(M)) = \{\alpha \in V^* | \exists q \in F, \exists v \in Z^n, (q, v) \in \delta((q_0, v_0), \alpha)\}.$$

It is clear that $pro(L(M))$ is a regular string language (in terms of formal languages). □

Example 1: Let $n = 2, Q = \{q_0, q_1, q_2\}, V = \{a, b, c\}, v_0 = \Omega_n, F = \{q_0\}$ and $\delta((q_0, \Omega_n), a) = \{(q_1, (1, 0))\}, \delta((q_1, \Omega_n), b) = \{(q_2, (0, 1))\}, \delta((q_2, \Omega_n), c) = \{(q_0, (0, 1))\}$. The language accepted by this 2-FAA contains all the arrays having the structure shown below:

$$\begin{array}{c} c \\ a \ b \\ \dots \\ c \\ a \ b \\ c \\ a \ b \end{array}$$

Thus $[L(M)] = \{((i, 2i), a)((i+1, 2i), b)((i+1, 2i+1), c) \cdots ((j, 2j), a), ((j+1, 2j), b), ((j+1, 2j+1), c) | i, j \in Z, i \leq j\}$ and $pro(L(M)) = (abc)^+$.

Example 2: If we take $n = 1$, $Q = \{q_0, q_1\}$, $V = \{x\}$, $v_0 = (0) = \Omega_1$, $F = \{q_0\}$ and $\delta((q_0, \Omega_1), x) = (q_1(1))$, $\delta((q_1, \Omega_1), x) = (q_0, (-1))$, the language accepted by this 1-FAA has only one sequence: $L_0(M) = \{((0), x)((1), x)\}$.

Indeed for the string xx , after the second x is accepted the sequence will have the form x^1x^1 and the current location becomes again Ω_1 where the symbol is now x^1 . Because the current state is the final state q_0 , this string is accepted and the automaton stops.

If $F = \{q_1\}$, then $L_0(M) = \{((0), x)\}$.

Remark 2: The usual Finite Automata accepting strings can be considered as a special case 1-FAA, where the frames used by the transition map are only $(i, i + 1) \in W_i$, $i = 0, 1, 2, \dots$.

Theorem 1: The class of languages accepted by n -FAA is $[L(n, \text{reg})]$.

Proof: Let L be a n -dimensional regular array language. Thus, there is a n -dimensional regular array grammar $G = (n, V_N, V_T, P, \{(v_s, S)\}, \#)$ with $[L(G)] = L$. We shall construct the n -FAAM $= (n, Q, V, \delta, q_0, v_0, F)$ as follows:

- $Q = V_N \cup \{X\}$, $q_0 = S$, $F = \{X\}$ where $X \notin V_N \cup V_T$;
- $V = V_T$;
- $v_0 = v_s$;
- The transition map δ is defined as follows:

For a production $\{A_1 = \{(\Omega_n, B), (v, \#)\}, A_2 = \{(\Omega_n, a), (v, C)\}\}$, the rule $(C, v) \in \delta((B, \Omega_n), a)$ is generated by keeping the same initial frame.

For a production $\{A_1 = \{(\Omega_n, B), A_2 = \{(\Omega_n, b)\}\}$, we consider an arbitrary initial frame $(\Omega_n, b) \in W_0$ we generate the rule $(X, v) \in \delta((B, \Omega_n), b)$.

The equality $[L(M)] = L$ can be proved, by the induction on the number of production rules used in a derivation.

Let M be a n -FAA and $L = [L(M)]$ be the array language accepted. We define a n -dimensional regular array grammar $G = (n, V_N, V_T, P, \{(v_s, S)\}, \#)$ as follows:

- $V_N = Q$, $V_T = V$;
- $S = q_0$; $v_s = v_0$;
- The set P of productions will contain:

$\{A_1 = \{(\Omega_n, q), (v, \#)\}, A_2 = \{(\Omega_n, a), (v, p)\}\}$, for any rule $(p, v) \in \delta((q, \Omega_n), a)$; the frame W is the same for the automaton rule and for the grammar production.

For any $q_f \in F$ we look for the rules $(q_f, v) \in \delta((q, \Omega_n), a)$; each such rule will generate a production $\{A_1 = \{(\Omega_n, q), \} A_2 = \{\Omega_n, a)\}$ where its associated frame is restricted to $W = \{\Omega_n\}$.

The proof of the equality $L = [L(G)]$ is now straightforward. \square

Example 3: For the 2-dimensional finite array automaton defined in Example 1, the regular array grammar which will generate the same language has the rules

$$\begin{aligned} & \{((0, 0), q_0), ((1, 0), \#)\}, \{((0, 0), a), ((1, 0), q_1)\}, \\ & \{((0, 0), q_1), ((0, 1), \#)\}, \{((0, 0), b), ((0, 1), q_2)\}, \\ & \{((0, 0), q_2), ((0, 1), \#)\}, \{((0, 0), c), ((0, 1), q_0)\}, \\ & \{((0, 0), q_2), ((0, 0), c)\}. \end{aligned}$$

3.1. The spreading languages

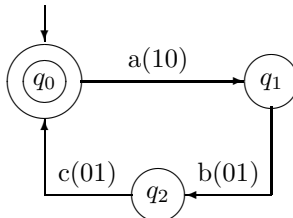
As in the case of regular languages, it is possible to associate an oriented graph Γ to a n -FAA. The construction of Γ is the following:

Let $M = (n, Q, V, \delta, q_0, v_0, F)$ be a n -dimensional finite array automaton.

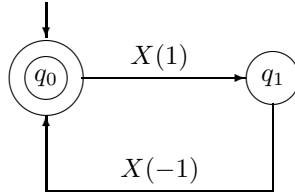
- The nodes of the graph Γ will be labelled by states; q_0 will be the initial node and the nodes labelled by F are final nodes.
- The labels of the edges are the elements of the set $E = \{a_u | a \in V, u \in W_0\}$.

For any rule $(p, u) \in \delta((q, \Omega_n), a)$, an edge from q to p , labelled by a_u is drawn. So, at most $2n$ edges can leave every node of this graph.

Example 4: For the 2-FAA of Example 1, the graph associated is.



Example 5: For the Example 2, the graph associated is



This graph Γ designs a finite automaton and thus a regular language.

Let $\overline{M} = (Q, \overline{V}, \overline{\delta}, q_0, F)$ be the finite automaton defined by the graph Γ , where

- $\overline{V} = \{a_u | a \in V, u \in W_0\}$;
- $p \in \overline{\delta}(q, a_u) \Leftrightarrow (p, u) \in \delta((q, \Omega_n), a)$.

Let $\overline{L} = \overline{L}(M)$ be the regular language accepted by the regular automaton M . This language will be called the spreading language of L .

For every word $w = (\Omega_n, a_0)(v_1, a_1) \cdots (v_k, a_k) \in L(M)$ there is a path in Γ designed by the sequence $\overline{w} = (a_0)_{v'_1}, (a_1)_{v'_2} \cdots (a_{k-1})_{v'_k}, (a_k)_{v'_{k+1}} \in \overline{V}^*$ from the initial state q_0 to a final state q with $(q, v_{k+1}) \in \delta((p, v_k), a_k)$; we have denoted by $v'_i = \tau_{-v_{i-1}}(v_i) (2 \leq i \leq k + 1), v'_{k+1} = v_1$.

Let $\phi : L \rightarrow \overline{L}$ be the mapping defined by $\phi(w) = \overline{w}$. In this way, for every word in a n -dimensional array regular language corresponds a word in a regular language. The language $\phi(L)$ is regular sublanguage of \overline{L} . Sometimes the mapping ϕ is surjective (like for the 2-dimensional array language defined in Example 1), but this property does not hold always; the 1-dimensional regular language defined in Example 2 is finite, but the graph Γ constructed in Example 5 accepts the infinite language $\overline{L} = (xx)^*$. So, in this case $\phi(L) \subset \overline{L}$; But it is possible to find if a given sentence $\overline{w} \in L$ is in $\phi(L)$ or not.

Theorem 2: Let L be a n -dimensional regular array language and \overline{L} its spreading language, with $\phi(L) \subset \overline{L}$. Let $w \in \overline{L}$. We can decide recursively whether $w \in \phi(L)$ or not.

Proof: Let $M = (n, Q, V, \delta, q_0, v_0, F)$ be a n -FAA with $L = L(M)$ and \overline{L} its spreading language. The algorithm will have as input a sentence $w = a_{u_0} a_{u_1} \cdots a_{u_{k-1}} \in \overline{L}, (a_u \in \overline{V}, u \in W_0, 0 \leq i \leq k - 1)$. The steps are:

- (1) $m_0 := \{v_0\}, i = 1$;
- (2) $v_i : -v_{i-1} + u_{i-1}; M_i := M_{i-1} \cup \{v_{i-1}\}$;

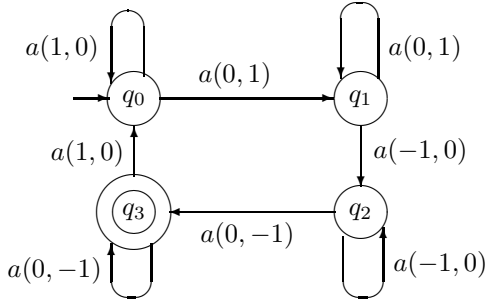
- (3) If $(M \neq M_{i-1})$ and $(i < k)$ then $i := i + 1$, go to 2;
- (4) If $i = k$ then $w \in \phi(L)$, otherwise $w \in \overline{L} \setminus \phi(L)$.

The algorithm tries to find a word $z \in L$ so that $\phi(z) = w$. So it iteratively builds the locations where the symbols of z can be placed. If these locations do not overlap, such a construction is possible from the point of view of finite array automaton M . The formal details are easy to be supplied. \square

Example 6: Let us consider the 2-dimensional finite array automaton $M = (2, Q, V, \delta, q_0, F)$ with $Q = \{q_0, q_1, q_2, q_3\}$, $F = \{q_3\}$, $V = \{a\}$ and the transition map:

$$\begin{aligned} \delta((q_0, \Omega_2), a) &= \{(q_0, (1, 0)), (q_1, (1, 0))\}, \\ \delta((q_1, \Omega_2), a) &= \{(q_1, (0, 1)), (q_2, (-1, 0))\}, \\ \delta((q_2, \Omega_2), a) &= \{(q_2, (-1, 0)), (q_3, (0, -1))\}, \\ \delta((q_3, \Omega_2), a) &= \{(q_3, (0, -1)), (q_0, (1, 0))\}, \end{aligned}$$

We shall construct the graph Γ designed by this automaton:



The spreading language is $L = (a_{(1,0)}^+ a_{(0,1)}^+ a_{(-1,0)}^+ a_{(0,-1)}^+)^+ \in L$.

Let us select for instance the sentence $w = a_{(1,0)} a_{(1,0)} a_{(0,1)} a_{(-1,0)} a_{(0,-1)} a_{(0,-1)} \in \overline{L}$.

Thus $k = 6$ and the algorithm will generate:

- (1) $v_0 = (0, 0)$, $M_0 = \{(0, 0)\}$;
- (2) $v_1 = (0, 0) + (1, 0) = (1, 0)$, $M_1 = \{(0, 0), (1, 0)\}$;
- (3) $v_2 = (1, 0) + (1, 0) = (2, 0)$, $M_2 = \{(0, 0), (1, 0), (2, 0)\}$;
- (4) $v_3 = (2, 0) + (0, 1) = (2, 1)$, $M_3 = \{(0, 0), (1, 0), (2, 0), (2, 1)\}$;
- (5) $v_4 = (2, 1) + (-1, 0) = (1, 1)$, $M_4 = \{(0, 0), (1, 0), (2, 0), (2, 1), (1, 1)\}$;
- (6) $v_5 = (1, 1) + (0, -1) = (1, 0)$, $M_5 = \{(0, 0), (1, 0), (2, 0), (2, 1), (1, 1)\} = M_4$;

Since $i = 5$ and $5 \neq k$, the answer is $w \notin \phi(L)$: if we wish to draw the array of this word in a 2-dimensional grid, the location $(1, 0)$ will appear twice.

But, if we consider the word $w = a_{(1,0)} a_{(1,0)} a_{(0,1)} a_{(-1,0)} a_{(0,-1)} \in L$, it will be accepted, because although $M_5 = M_4$, we have $i = k = 5$. The sentence $z \in L$ with $\phi(z) = w$ is $z = (\Omega_2, a) ((1, 0), a) ((2, 0), a) ((2, 1), a) ((1, 1), a)$ with the array representation shown below:

$$\begin{array}{c} \# a a \\ a a a \end{array}$$

4. Regular Array Languages and Pumping Lemma

Let M be a n -dimensional finite array automaton, $L = L(M)$ be the regular array language accepted by M and \bar{L} its spreading language. Since \bar{L} is a regular language and $\phi(L) \subseteq \bar{L}$, we can use the pumping lemma in order to decide whether or not some n -array languages are regular.

Proposition 2:

- (1) *The 2-dimensional array language of all hollow rectangles is not regular.*
- (2) *The 2-dimensional array language of all hollow squares is not regular.*

Proof: (1) Let L be the 2-dimensional array language of all rectangles. Let us suppose that this is a regular array language and let \bar{L} be its spreading language. Since any rectangle has four sides with opposite sides equal, we have

$$\bar{L} = \{w/w = a_{(1,0)}^n a_{(0,1)}^m a_{(-1,0)}^n a_{(0,-1)}^m, m, n \geq 1\}$$

where $a \in V$ is the symbol used by drawing the rectangles.

But the pumping lemma for regular languages shows that a language of type $\{a^n b^m c^n d^m | m, n \geq 1\}$ is not a regular language⁴; thus \bar{L} is not regular, a contradiction.

(2) If the 2-dimensional language of all squares would be a regular language, the spreading language associated will be

$$\bar{L} = \{w/w = a_{(1,0)}^n a_{(0,1)}^n a_{(-1,0)}^n a_{(0,-1)}^n, n \geq 1\};$$

but a language of type $\{a^n b^n c^n d^n | n \geq 1\}$ is not regular, so \bar{L} cannot be a spreading language, contradiction. □

Remark: It is known¹¹ that hollow rectangles and hollow squares cannot be generated by regular array grammars although solid rectangles and solid

(3): there is no word in L of the form xz , because overlaps will arise in all variants.

5. Control on Regular Array grammars

We examine now the effect of controlling the application of rules of a regular array grammar by prescribed sequences of their applications, constituting a control language. We consider the case $n = 2$ here.

In the two dimensional case the array productions can be depicted in a more simple way as given below:

$$A\# \rightarrow aB, \#A \rightarrow Ba, \begin{matrix} \# & \rightarrow & B \\ A & \rightarrow & a \end{matrix}, \begin{matrix} A & \rightarrow & a \\ & \# & \rightarrow & B \end{matrix}, A \rightarrow a.$$

The concept of control on the application of the rules in a grammar is standard in the string language theory. The rules of a grammar are given labels and the control word is a sequence of such labels. A derivation in the grammar is obtained from the start symbol by applying the productions corresponding to the labels that occur in order from left to right in a given control word. The terminal words obtained in this way constitute the language generated. The control words themselves constitute a language, called the control language. Here we consider regular, context-free or context-sensitive control languages. We denote a 2-dimensional regular array grammar with control by $G = (2, V_N, V_T, P, \{(v_s, S), \#\}, Lab(P), C)$ where $Lab(P)$ is the set of labels of productions in P and $C \subseteq (Lab(P))^*$ is the control language.

As in string language theory regular control does not increase the generative power of regular array grammars but CF and CS controls do increase.

Proposition 3: *Given a 2-dimensional regular array grammar*

$$G = (2, V_N, V_T, P, \{(v_s, S), \#\}, Lab(P), C) \text{ with control } C,$$

the array language generated by G is regular if $C \subseteq (Lab(P))^$ is a regular string language. But if C is context-free or context-sensitive then the array language generated need not be regular.*

Proof: When C is regular generated by rules of the forms $X \rightarrow fY, Z \rightarrow g$ where f, g belong to $Lab(P)$ and X, Y, Z are nonterminals in the regular grammar generating C and if f is the label of a rule in G of the form, say, $A\# \rightarrow aB$, then we form a regular array rule of the form $(A, X)\# \rightarrow$

$a(B, Y)$ (likewise for other types of regular array rules). If g is the label of a rule in G (which will be a terminal rule) of the form $A \rightarrow a$, then we form a regular array rule of the form $(A, Z) \rightarrow a$. It can be easily seen that the 2-dimensional array language generated by the new regular array productions is the same as the 2-dimensional array language of G . In fact the effect of the control word is taken care of by the second component of the new nonterminals formed.

When C is context-free or context-sensitive, that the generative capacity of G is increased can be seen from the following examples.

We give only the rules of the regular array grammar G_1 and the context-free control language C_1 . The rules are

$$f_1 : \#S \rightarrow Sa, \quad f_2 : \begin{matrix} \# \\ S \end{matrix} \rightarrow \begin{matrix} S \\ a \end{matrix}, \quad f_3 : S \rightarrow a$$

and context-free control language

$$C = \{f_1^{n-1} f_2^{n-1} f_3/n \geq 1\}.$$

G_1 generates arrays over $\{a\}$ which describe “token L” with equal arms of all sizes. For instance, for $n = 5$, the control word is $f_1^4 f_2^4 f_3$. Applying rules with label f_1 four times followed by f_2 four times and finally f_3 , we obtain the array as

$$\begin{array}{c} a \\ a \\ a \\ a \\ a \, a \, a \, a \end{array}$$

It can be seen that these arrays describing token L cannot be generated by a regular array grammar as the “arms” of equal sizes cannot be maintained by regular array grammar rules.

When the rules are the following

$$f_1 : S\# \rightarrow aS, \quad f_2 : \begin{matrix} \# \\ S \end{matrix} \rightarrow \begin{matrix} S \\ a \end{matrix}, \quad f_3 : \#S \rightarrow Sa,$$

$$f_4 : \begin{matrix} S \\ \# \end{matrix} \rightarrow \begin{matrix} a \\ S \end{matrix}, \quad f_5 : S \rightarrow a$$

and the context-sensitive control language $C = \{f_1^{m-1} f_2^{n-1} f_3^{m-1} f_4^{n-2} f_5/m, n \geq 1\}$ then the grammar generates arrays over $\{a\}$ which describe hollow rectangles. \square

6. Final Remarks

The recognition device proposed here, namely, the n -dimensional finite array automaton equivalent to the n -dimensional regular array grammar, represents a natural extension of the usual finite automaton and recognizes $[L(n, \text{reg})]$. There are some problems that remain to be solved. For example, each regular language R is spreading a family of n -dimensional finite array languages. What are the properties of this family? It not empty because R is spreading by itself. Is it possible to establish a pumping lemma regarding n -FAA? This will help us to solve some decidability problems (i.e. emptiness, finiteness). It would also be interesting to study the deterministic behavior of n -FAA. For example, we can consider that a n -FAA is non-deterministic if and only if there are $p, q, r \in Q, y \in W_0, a \in V$ with $(p, y), (r, y) \in \delta((q, \Omega_n), a)$ but $p \neq r$. This type of definition for non-deterministic n -FAA can be eliminated in the same way as we do for usual NFA (see [4]). But, What happens if $(p, y), (r, z) \in \delta((q, \Omega_n), a)$ but $y \neq z$? These questions and other problems regarding n -dimensional finite array automata remain to be solved in forthcoming papers.

References

1. H. Fernau, R. Freund and M. Holzer, Character recognition with K-head finite array automata, *SSPR'98, Sydney, Australia*, 1998.
2. H. Fernau, R. Freund and M. Holzer, Regulated array grammars of finite index, Parts I and II: Theoretical investigations in I and Syntactic pattern recognition in II, In "*Grammatical models of multi-agent systems*", Gh. Paun, A. Salomaa (Eds.) Gordon and Breach, Reading, UK, 1998.
3. R. Freund, Control mechanism on $\#$ context-free array grammars, In Gh. Paun (Ed.), *Mathematical aspects of Natural and Formal Languages*, World Scientific, Singapore, 97–137, 1994.
4. R. Freund, Array grammars, *XI Tarragona Seminar on Formal Syntax and Semantics, FS and S, Tech. Report 15/00*, 2000.
5. R. Freund, Array Grammar Systems, *Journal of Automata, Languages and Combinatorics* 5, 13–20, 2000.
6. K. S. Fu, Syntactic Methods in Pattern Recognition, *Academic Press, New York*, 1974.
7. M. A. Harrison, Introduction to Formal Language Theory, *Addison Wesley, Reading Cliffs, MA*, 1978.
8. A. Rosenfeld, Picture Languages, *Academic Press, New York*, 1979.
9. A. Rosenfeld and R. Siromoney, Picture Languages — A Survey, *Languages of design* 1, 229–245, 1999.
10. G. Rozenberg and A. Salomaa (Eds.), Handbook of Formal Languages, *Springer-Verlag*, 1997.

11. Y. Yamamoto, K. Morita and K. Sugata, Context-sensitivity of two-dimensional regular array grammars, *In P. S. P. Wang (Ed.) Array Grammars, Patterns and Recognizers, WSP Series in Computer Science, Vol. 18, 1989.*