Character Recognition with k-Head Finite Array Automata

Henning Fernau[1]*, Rudolf Freund[2], and Markus Holzer[1]

[1] Wilhelm-Schickard-Institut für Informatik, Universität Tübingen
Sand 13, D-72076 Tübingen, Germany
email: {fernau,holzer}@informatik.uni-tuebingen.de
[2] Institut für Computersprachen, Technische Universität Wien
Resselgasse 3, A-1040 Wien, Austria, email: freund@logic.at

Abstract. We introduce the concept of k-head finite two-dimensional array automata and show how this model of two-dimensional array automata can be applied in the field of syntactic character recognition. Moreover, we discuss some of the problems arising with implementing the theoretical concept to obtain an efficient tool for the syntactic analysis of handwritten (uppercase) characters.

1 Introduction

In the field of syntactic character recognition, the theoretical concept of array grammars represents an interesting formal framework that allows for accurate descriptions of two-dimensional patterns like (handwritten, uppercase) characters. Whereas "pure" context-free array grammars as proposed in [19] not yet allow efficient implementations of the array grammar approach for the syntactic clustering analysis of handwritten characters, additional features like the controlled use of productions yield more efficient algorithms as exhibited in [9] within the framework of graph controlled grammars; moreover, the model has to incorporate suitable error measures computed during the analysis of the underlying pattern in order to allow for most accurate analysis results distinguishing between different clusters of characters.

Seen from a theoretical point of view, such additional features increase the generative power of the array model, yet as we operate on a bounded rectangle only, this is of minor importance; on the other hand, a controlled use of array productions reduces the non-determinism of the analyzing device and thus increases efficiency by minimizing the needs to backtrack.

There are some crucial differences between an approach based on neural networks and an approach based on a syntactical model like that proposed in this paper: In a trained neural network, the incorporated features are hardly transparent to the user. In our syntactical approach, we obtain reproducible results, because the characteristics of the letters as well as suitable error measures have to be defined during the design phase of the tool. This also allows us to make experiments

* Supported by Deutsche Forschungsgemeinschaft grant DFG La 618/3-2.

with the parameters weighting different errors as deviations of lines, gaps in the lines, etc. Hence, the more complicated design of all the different grammars for each letter on the other hand allows us to obtain more accurate results.

In [5] prescribed teams of array productions (first investigated from a theoretical point of view in [8]) were proposed as another control mechanism to be used for increasing the efficiency of the analyzing model. With prescibed teams, a bounded number of array productions is applied in parallel to the underlying array. This approach resembles the idea of the cooperation of agents, which is a usual strategy for solving complex problems.

In this paper we elaborate how k-head finite array automata, which were already defined in [7] and, from a theoretical point of view, are closely related to regulated array grammar systems with prescribed teams of finite index as investigated in [6], can be used for the syntactic clustering analysis of handwritten characters.

In the next section, we present the formal framework of (two-dimensional) arrays and array grammars as well as of the new model of k-head finite array automata. In the third section, we describe how (handwritten) characters can be analyzed by using suitable k-head finite array automata and discuss some of the problems arising when going to implement such a theoretical model in such a way that an efficient analysis of given arrays representing handwritten characters becomes possible. A short discussion of the results exhibited in this paper and an outlook to future research topics conclude the paper.

2 The Formal Framework

In this section we introduce the definitions and notations for arrays, (graph controlled) array grammars, and k-head finite array automata (e.g. see [2], [6], [16], [18]). For the basic notions and results of formal language theory addressed in this paper we refer the reader to [3].

2.1 Arrays and array grammars

Let Z denote the set of integers, and let V be an alphabet. Then a *(two-dimensional) array* $\mathcal{A}$ over V is a function $\mathcal{A} : Z^2 \to V \cup \{\#\}$, where $shape(\mathcal{A}) = \{v \in Z^2 \mid \mathcal{A}(v) \neq \#\}$ is finite and $\# \notin V$ is called the *background* or *blank symbol*. We usually shall write $\mathcal{A} = \{(v, \mathcal{A}(v)) \mid v \in shape(\mathcal{A})\}$. The set of all two-dimensional non-empty arrays over V shall be denoted by V^{+2}. Any subset of V^{+2} is called an array language.

Given an array $\mathcal{A} \in V^{+2}$ (for some alphabet V) and a finite pattern α (i.e., a partial function $Z^2 \to V \cup \{\#\}$ with finite domain) of symbols in $V \cup \{\#\}$, we can say that α is a *sub-pattern* of $\mathcal{A}$, if we can place α on $\mathcal{A}$ such that all squares of α marked by symbols in V coincide with the corresponding symbols in $\mathcal{A}$ and each blank symbol $\#$ in α corresponds to a blank symbol $\#$ in $\mathcal{A}$.

An *(isometric) array grammar* is a construct $G = (V_N, V_T, \#, P, (v_0, S))$, where V_N, V_T are disjoint alphabets, $\#$ is a special (blank) symbol, $v_0 \in Z^2$ is the start vector and $S \in N$ is the start symbol, and P is a finite set of rewriting rules of

the form $\alpha \to \beta$, where α, β are finite patterns over $V_N \cup V_T \cup \{\#\}$ satisfying the condition that the shapes of α and β are identical (we say that they are *isometric*).

Thus, for an array grammar $G = (V_N, V_T, \#, P, (v_0, S))$ we can define the relation $\mathcal{A} \Longrightarrow \mathcal{B}$, for $\mathcal{A}, \mathcal{B} \in (V_N \cup V_T)^{+2}$, if there is a rule $\alpha \to \beta \in P$ such that α is a sub-pattern of $\mathcal{A}$ and $\mathcal{B}$ is obtained by replacing α in $\mathcal{A}$ by β (remember that α and β are isometric). The reflexive and transitive closure of $\Longrightarrow$ is denoted by $\Longrightarrow^*$, and the array language generated by G is defined by $L(G) = \{\mathcal{A} \in V_T^{+2} \mid \{(v_0, S)\} \Longrightarrow^* \mathcal{A}\}$. An array production $\alpha \to \beta$ in an array grammar is said to be

1. *monotone* if the non-# symbols in α are not replaced by # in β,
2. *#-context-free* if α consists of exactly one nonterminal and some occurrences of blank symbols #,
3. *context-free* if it is #-context-free and monotone.

An array grammar is said to be of type MON, $\#\text{-}CF$, or CF, respectively, if every array production in P is monotone, #-context-free, or context-free, respectively. The same notation is used for the corresponding (families of) array languages.

2.2 Control mechanisms

In the following, we give the necessary definitions of graph controlled array grammars and languages. Concerning these control mechanisms as well as many other interesting results about regulated rewriting in the theory of string languages, the reader is referred to [3].

A *two-dimensional graph controlled grammar* [9] is a construct $G = (V_N, V_T, \#, (P, L_{in}, L_{fin}), (v_0, S))$; V_N and V_T are disjoint alphabets of non-terminal and terminal symbols, respectively; v_0 is the start vector, $S \in V_N$ is the start symbol; P is a finite set of productions p of the form $(l(p) : \pi(l(p)), \sigma(l(p)), \varphi(l(p)))$, where $l(p) \in Lab(G)$, $Lab(G)$ being a set of labels associated (in a one-to-one manner) to the rules p in P, $\pi(l(p))$ is an array production over $V_N \cup V_T$, $\sigma(l(p)) \subseteq Lab(G)$ is the *success field* of the rule p, and $\varphi(l(p))$ is the *failure field* of the rule p; $L_{in} \subseteq Lab(G)$ is the set of initial labels, and $L_{fin} \subseteq Lab(G)$ is the set of final labels. For $p \in P$ and $v, w \in (V_N \cup V_T)^{+2}$ we define $(v, l(p)) \Longrightarrow_G (w, k)$ if and only if *either* $\pi(l(p))$ is applicable to v, the result of the application of the production $\pi(l(p))$ to v is w, and $k \in \sigma(l(p))$, *or* $\pi(l(p))$ is not applicable to v, $w = v$, and $k \in \varphi(l(p))$. Let $w_0(v_0) = S$ and $w_0(v) = \#$ for $v \in Z^2 \setminus v_0$ define the start array w_0. The array language generated by G is

$$L(G) = \{w \in V_T^{+2} \mid (w_0, l_0) \Longrightarrow_G (w_1, l_1) \Longrightarrow_G \ldots (w_k, l_k), k \geq 0,$$
$$w_j \in (V_N \cup V_T)^{+2}, l_j \in Lab(G) \text{ for } 0 \leq j \leq k, w_k = w, l_0 \in L_{in}, l_k \in L_{fin}\}.$$

If the failure fields $\varphi(l(p))$ are empty for all $p \in P$, then G is called a *graph controlled array grammar without appearance checking*.

A graph controlled array grammar is said to be of type X if every array production appearing in this grammar is of the corresponding type X, too; for every $X \in \{\#\text{-}CF, CF\}$, by $GC_{ac}(X)$ and $GC(X)$ we denote the array languages described by graph controlled array grammars and graph controlled array grammars without appearance checking, respectively, containing only productions of type X.

2.3 The finite index restriction

Usually, the number of non-terminal symbols occurring in the sentential forms of a derivation is not bounded, yet a natural measure for the complexity of the evolving terminal object. Even in some applications as character recognition we can restrict ourselves to a quite low bound of non-terminal symbols occurring in the sentential forms of a derivation as, for example, exhibited in [5]. Hence, we introduce the following definitions:

The *index* of a derivation D of a terminal object w in an array grammar G is defined as the maximal number of non-terminal symbols occurring in an intermediate sentential form. A grammar G is said to be of index k if every word $w \in L(G)$ has a derivation D of index at most k. This definition also makes sense for graph controlled array grammars with or without appearance checking etc.

For a given array language L and the class of graph controlled array grammars of type X, we say that L has index k with respect to this class, abbreviated as $L \in GC_{ac}^{[k]}(X)$, if there exists a graph controlled grammar G of index k and type X such that $L = L(G)$. Furthermore, let $GC_{ac}^{[fin]}(X) = \bigcup_{k \geq 1} GC_{ac}^{[k]}(X)$ denote the class of graph controlled finite index languages of type X. In those cases where we do not allow appearance checking, we simply omit the subscript ac.

2.4 k-head finite array automata

We are now going to define k-head finite array automata, which we will use as the formal basis of the tool for syntactic character recognition described in the following section. In the string case, multi-head finite automata first were described in [14]. For various results on two-dimensional automata, the reader is referred to [1], [10], [11], and [12]. A (2-dimensional) *k-head finite array automaton of type X*, $X \in \{\#\text{-}CF,\ CF\}$ is a construct $M = (V_N, V_T, \#, (P, \mathcal{R}, F), (v_0, S))$, such that (1) $(V_N, V_T, \#, P, (v_0, S))$ is a grammar of type X; (2) $\mathcal{R}$ is a set of subsets (called *teams*) of P such that each set (team) in $\mathcal{R}$ contains at most k array productions and for each set (team) $R \in \mathcal{R}$ it is true that each non-terminal symbol occurs at most once within the left-hand side of a rule in R; (3) $F \subseteq P$ so that $R = \{p_1, \ldots, p_m\}$ can be applied in the so-called appearance checking mode in such a way that any of the p_i appearing in F can be skipped, if p_i is not applicable to the sentential form after having team R chosen to be applied. If F is empty, then we call M a *finite k-head array automaton without appearance checking.*

In general, the configurations of M are objects from the set

$$\{(a,a),(a,X),(\#,Y) \mid a \in V_T, X \in V_N \cup \{\#\}, Y \in V_N\}^{+2},$$

where the first component always contains the input array and the second component keeps track of the array scanned so far by M. The work of the automaton M on a given array from V_T^{+2} now is defined as follows: Given an input array $\mathcal{A} \in V_T^{+2}$, the initial sentential array form (configuration) is

$$C_0(\mathcal{A}) = \{(v, (\mathcal{A}(v), \#)) \mid v \in Z^2 \setminus \{v_0\}\} \cup \{(v_0, (\mathcal{A}(v_0), S))\}.$$

More precisely, a configuration C of such an automaton may be split into (1) a set of already verified (read) positions containing symbols (a,a) with $a \in V_T$, (2) a set of positions containing symbols $(a,\#)$ which remain to be verified, (3) a set of at most k symbols of the form (b,X) with $b \in V_T \cup \{\#\}$, $X \in V_N$. Comparing our automaton model with traditional ones, one sees that the current state of the automaton M is represented by the set of non-terminal symbols Y occurring in the pairs (b,Y) of the current array.

The idea of applying a team in $\mathcal{R}$ is as follows: (1) By applying the team all the non-terminal symbols appearing in the current array are derived in parallel. (2) At each position where we already find a terminal symbol in the underlying array, this terminal symbol must coincide with the corresponding symbol at this position in the originally given array. We would like to mention that the chosen domains of the array productions might overlap. As in [6] we demand that at those positions where a non-blank symbol will result the domains of the chosen subarrays must not overlap, whereas we allow the sensing for a blank symbol at a special position from different positions. Furthermore, the application of a team is only allowed if all non-terminal symbols occurring in the sentential form are affected by the team, at least in the appearance checking mode. This guarantees that in a derivation yielding a terminal object the number of non-terminal symbols occurring in the intermediate sentential forms can never exceed the maximal number of context-free productions occurring in any of the teams in $\mathcal{R}$.

Formally, the derivation relation $\Longrightarrow_M$ for M is defined as the union over all team relations $\vdash_R$ for $R \in \mathcal{R}$. Let $R = \{p_1, \ldots, p_n\}$. For a configuration C, let $C(2)$ denote the projection to the second component of the letters. Having two configurations C, C', we write $C \vdash_R C'$ if and only if (1) every variable occurring in $C(2)$ occurs only once; (2) every variable occurring in $C'(2)$ occurs only once; (3) every $p_i \in R$ either is applicable to $C(2)$ or belongs to F; (4) let $R(C)$ denote the set of applicable rules of R; then, the set of variables occurring in left-hand sides of rules in $R(C)$ coincides with the set of variables occurring in C; (5) the domains of the subarrays to be replaced by the applicable rules of R must not overlap at positions where according to the right-hand sides of these rules non-blank symbols will result.

The array language accepted by M therefore is defined by

$$L(M) = \left\{ \mathcal{A} \mid \mathcal{A} \in V_T^{+2}, C_0(\mathcal{A}) \Longrightarrow_M^* \{(v, (\mathcal{A}(v), \mathcal{A}(v))) \mid v \in \mathit{shape}(\mathcal{A})\} \right\}.$$

Observe that due to our definitions, a head of the automaton, which is represented by one of the at most k (different) variables appearing in the current array, reads out the symbol in the first component just when leaving this position with putting there the exactly same terminal symbol into the second component. As the terminal symbol at a specific position is already uniquely determined by the first position we could also put only a specific marker symbol into the second components just to mark these positions as non-reachable by any head of the automaton anymore.

The family of all array languages accepted by k-head finite array automata of type X, $X \in \{\#\text{-}CF, CF\}$, is denoted by $k\text{-}FA_{ac}(X)$. In those cases where we do not allow appearance checking we simply omit the subscript ac.

2.5 A concrete example

Let us now give an example of a two-dimensional 4-head finite array automaton that accepts the cluster of ideal letters "H" of arbitrary size:

$$M = (\{S, L, R, D_L, U_L, D_R, U_R\}, \{a\}, \#, (P, \mathcal{R}, \emptyset), \{v_0, S\});$$

$$P = \Big\{ S\,\# \to L\,R,\ \#\,L \to L\,a,\ R\,\# \to a\,R,$$

$$\begin{matrix} \# \\ L \end{matrix} \to \begin{matrix} U_L \\ a \end{matrix}, \begin{matrix} R \\ \# \end{matrix} \to \begin{matrix} \# \\ a \end{matrix}, \begin{matrix} \# \\ U_L \end{matrix} \to \begin{matrix} U_L \\ a \end{matrix}, \begin{matrix} \# \\ U_R \end{matrix} \to \begin{matrix} U_R \\ a \end{matrix},$$

$$\begin{matrix} \# \\ D_L \end{matrix} \to \begin{matrix} D_L \\ a \end{matrix}, \begin{matrix} \# \\ D_R \end{matrix} \to \begin{matrix} D_R \\ a \end{matrix}, U_L \to a, U_R \to a, D_L \to a, D_R \to a \Big\};$$

$$\mathcal{R} = \Big\{ \{ S\,\# \to L\,R \}, \{ \#\,L \to L\,a,\ R\,\# \to a\,R \},$$

$$\left\{ \begin{matrix} \# \\ L \end{matrix} \to \begin{matrix} U_L \\ a \end{matrix}, \begin{matrix} R \\ \# \end{matrix} \to \begin{matrix} \# \\ a \end{matrix} \right\},$$

$$\left\{ \begin{matrix} \# \\ U_L \end{matrix} \to \begin{matrix} U_L \\ a \end{matrix}, \begin{matrix} \# \\ U_R \end{matrix} \to \begin{matrix} U_R \\ a \end{matrix}, \begin{matrix} D_L \\ \# \end{matrix} \to \begin{matrix} a \\ D_L \end{matrix}, \begin{matrix} D_R \\ \# \end{matrix} \to \begin{matrix} a \\ D_R \end{matrix} \right\},$$

$$\{U_L \to a, U_R \to a, D_L \to a, D_R \to a\} \Big\}.$$

A typical computation of M is the following one:

$$\begin{matrix} (a,\#) & & (a,\#) \\ (a,\#) & & (a,\#) \\ & (a,S)\ (a,\#) & \\ (a,\#) & & (a,\#) \\ (a,\#) & & (a,\#) \end{matrix} \Longrightarrow_M \begin{matrix} (a,\#) & & (a,\#) \\ (a,\#) & & (a,\#) \\ & (a,L)\ (a,R) & \\ (a,\#) & & (a,\#) \\ (a,\#) & & (a,\#) \end{matrix} \Longrightarrow_M$$

$$\begin{matrix} (a,\#) & & (a,\#) \\ (a,U_L) & & (a,U_R) \\ & (a,a)\ (a,a) & \\ (a,D_L) & & (a,D_R) \\ (a,\#) & & (a,\#) \end{matrix} \Longrightarrow_M \begin{matrix} (a,U_L) & & (a,U_R) \\ (a,a) & & (a,a) \\ & (a,a)\ (a,a) & \\ (a,a) & & (a,a) \\ (a,D_L) & & (a,D_R) \end{matrix} \Longrightarrow_M$$

$$\begin{matrix} (a,a) & & (a,a) \\ (a,a) & & (a,a) \\ & (a,a)\ (a,a) & \\ (a,a) & & (a,a) \\ (a,a) & & (a,a) \end{matrix}$$

2.6 Some theoretical results

The computational power of k-head finite array automata was studied in [7], where the following theoretical results were proved: For $X \in \{\#\text{-}CF,\ CF\}$, we have

1. $\bigcup_{k\geq 1} k\text{-}FA\,(X) = GC^{[fin]}\,(X)$ as well as
2. $\bigcup_{k\geq 1} k\text{-}FA_{ac}\,(X) = GC_{ac}^{[fin]}\,(X)$.

As an important implication of these theoretical results for the practical implementation discussed in the following section we would like to mention that adding control graphs to k-head finite array automata (in order to get more control on the inherent non-determinism with respect to the choice of a suitable team) theoretically does not increase the power of the model.

3 Implementation Features

In this section we discuss some of the main problems we have to take care of when implementing a tool based on the theoretical framework described in the preceding section.

3.1 Data acquisition and preprocessing

Handwritten characters were acquired from hundreds of persons on specific forms and then scanned in order to obtain digital pixel images. A reference to the database obtained in that way can be found in [4].

The scanned characters first were normalized to fill out a 320×400 grid in order to get comparable patterns. Then noisy pixels were eliminated. After noise elimination, the resulting arrays on the 320×400 grid were mapped on a 20×25 grid. These arrays on the 20×25 grid then were subjected to a thinning algorithm (e.g., see [17]) which finally yielded unitary skeletons of the digitized characters.

3.2 Syntactic analysis by k-head finite array automata

The unitary skeletons of the digitized characters now are the input for the syntactic analyzing tool based on the formal model of k-head finite array automata. For each letter in the alphabet, such an automaton has to be designed in a suitable way. For example, compare the 4-head finite array automaton for the letter "H" described in the preceding section.

An important feature of the parsing sequence depicted in the example given in Subsection 2.3 is the determinism of the given derivation, i.e., for each array in $L\,(M)$ there is exactly one parsing derivation. For arrays not in $L\,(M)$, the crucial moment is the change from the horizontal line to the vertical lines. Yet in this special case of M, the possibility of a non-deterministic choice in the underlying pattern immediately implies that this pattern cannot belong to $L(M)$. Yet for practical implementations where we also want to recognize non-ideal patterns in

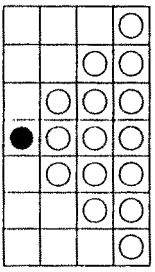

Fig. 1. Lookahead region (the current head position is indicated by a filled circle, the positions of the lookahead region by circles).

a decent way, this is one of the most important problems we have to deal with. One possible solution is depicted in Figure 1, i.e., looking ahead to larger areas for possible continuations of the analysis instead of searching for the most suitable subpatterns by backtracking.

3.3 Error measurement

One of the most important features of an efficient tool is to use suitable error measures which allow us to obtain reasonable clusters for the different letters in the alphabet. The main features constituting the distance between a given array obtained from a realistic pattern and the ideal cluster are the number of gaps, the deviation of lines, and the number of unused pixels.

For example, in order to cover deviations of a horizontal line, a head of the automaton going to the right not only has to look at the next position to the right, but also at the positions in the diagonals to the right as depicted in Figure 2.

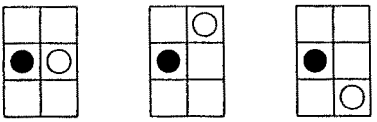

Fig. 2. Deviation of a horizontal line (the current head position is indicated by a filled circle, the positions of possible continuations of the line to the right by non-filled circles).

Moreover, sometimes the border line between two different letters is quite "fluent" (see Figure 3). For example, the array represented by the filled circles will still be recognized as an "H" by a lot of people, whereas when adding the pixels indicated by the non-filled circles most people will agree in recognizing this array as an "A", because the upper endings of the vertical lines now are close enough to each other; yet the question remains how to determine exact values for the distance of these endings as well as for the deviations of the vertical lines in order to separate the cluster of arrays representing the symbol "A" from the cluster of arrays representing the symbol "H".

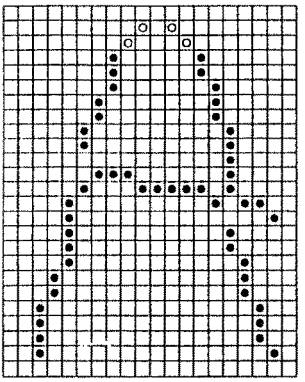

Fig. 3. On the border between "H" and "A".

The main features of a given character that may contribute to an error measure are the deviations from the lines building up an ideal letter and the remaining pixels not covered by the syntactic analysis. Yet also more elaborated features as the distances of end points of lines (compare the discussion above concerning the letters "A" and "H" may increase the error and thus help to distinguish between two clusters representing different letters.

4 Summary and Future Research

We have shown that k-head finite array automata represent a new promising theoretical approach for the efficient syntactic analysis of characters. In order to increase the efficiency of the implemented tool, the analysis of a given pattern by the k-head finite array automata defined for different characters should be carried out in parallel, which would also allow us to avoid to continue the analysis by k-head finite array automata having computed an error greater than the error computed by a k-head finite array automaton having already finished the analysis of the given pattern. Moreover, we would like to point out that in this paper we only have presented part of a hybrid system also containing a quick pre-analysis part and a neural network part. So far we have considered two variants of such a quick pre-analysis: The first one checks for specific crossing points with vertical and horizontal lines, whereas the second one is based on a neural network using the density of pixels in the rows and columns, respectively. A careful use of such a pre-analysis is necessary in order not to pay for the increase of efficiency with the loss of accuracy (by eliminating the "correct answer", i.e. due to the bad quality of the underlying letter the right k-head finite array automaton will not even be started). On the other hand, a neural network based on more complex features represents a second useful tool for the correct clustering of the given characters and can be used in parallel with the syntactic tool based on the k-head finite array

automata. The improvement of each single component of the hybrid system as well as of the collaboration of the components remains for future research.

References

1. M. Blum and C. Hewitt, Automata on a two-dimensional tape. In: *IEEE Symposium on Switching and Automata Theory* (1967), pp. 155–160.
2. C. R. Cook and P. S.-P. Wang, A Chomsky hierarchy of isotonic array grammars and languages, *Computer Graphics and Image Processing* **8** (1978), pp. 144–152.
3. J. Dassow and Gh. Păun, *Regulated Rewriting in Formal Language Theory* (Springer, Berlin, 1989).
4. W. Dittmann, Character recognition by array systems of bounded index, *Diploma thesis,* Techn. Univ. Wien, Austria, 1997.
5. H. Fernau and R. Freund, Bounded parallelism in array grammars used for character recognition. In: P. Perner, P. S.-P. Wang, and A. Rosenfeld (eds.), *Proceedings SSPR'96,* LNCS 1121 (Springer, Berlin, 1996), pp. 40–49.
6. H. Fernau, R. Freund, and M. Holzer, Regulated array grammars of finite index, Part I. Theoretical investigations, *to appear in:* [13].
7. H. Fernau, R. Freund, and M. Holzer, Regulated array grammars of finite index, Part II: Syntactic pattern recognition, *to appear in:* [13].
8. R. Freund, Array grammar systems with prescribed teams of array productions. In: J. Dassow, Gh. Păun and A. Salomaa (eds.), *Developments in Language Theory II* (Gordon and Breach, London, 1996), pp. 220–229.
9. R. Freund, Syntactic recognition of handwritten characters by programmed array grammars with attribute vectors, Seventh International Conference on Image Analysis and Processing, Bari, Italy, in: *Progress in Image Analysis and Processing III* (ed. S. Impedovo, World Scientific Publ., Singapore, 1993), pp. 357 – 364.
10. O. Ibarra and R. T. Melson, Some results concerning automata on two-dimensional tapes, *International Journal of Computer Mathematics, Series A* **4** (1974), pp. 269–279.
11. K. Inoue and I. Takanami, A survey of two-dimensional automata theory. In: J. Dassow and J. Kelemen (eds.), Machines, Languages, and Complexity, *IMYCS'88,* LNCS 381 (Springer, Berlin, 1988), pp. 72–91.
12. A. Nakamura, Parallel Σ-erasing array acceptors, *Computer Graphics and Image Processing* **14** (1980), pp. 80–86.
13. Gh. Păun and A. Salomaa (eds.), *Grammatical Models of Multi-Agent Systems* (Gordon and Breach, Reading, UK, 1998).
14. A. L. Rosenberg, On multihead finite automata, *IBM J. Res. Develop.* **10** (1966), pp. 388–394.
15. A. Rosenfeld, Some notes on finite-state picture languages, *Information and Control* **31** (1976), pp. 177–184.
16. A. Rosenfeld, *Picture Languages* (Academic Press, Reading, MA, 1979).
17. J. H. Sossa, An improved parallel algorithm for thinning digital patterns, *Pattern Recognition Letters* **10** (1989), pp. 77-80.
18. P. S.-P. Wang, Some new results on isotonic array grammars, *Information Processing Letters* **10** (1980), pp. 129–131.
19. P. S.-P. Wang, An application of array grammars to clustering analysis for syntactic patterns, Pattern Recognition **17**, 4 (1984), pp. 441 – 451.