TR-79-002

Two-Demensional Pattern Generation and Classification Using Array Grammars

by

Patrick Shen-Pei Wang

Institute of Information Science
Academia Sinica
Taipei, Taiwan 115
R. O. C.

Department of Computer Science
University of Oregon
Eugene, Oregon 97403
U. S. A.

Array grammars and languages, hierarchy, rookwise connectedness, shearing effect, two-dimensional patterns, recognition.

Two-Dimensional Pattern Generation and Classification Using Array Grammars

Patrick Sehn-pei Wang

Department of Computer Science
University of Oregon
Eugene, Oregon 97403
U. S. A.

and

Institute of Information Science
Academia Sinica
Taipei, Taiwan 115
R. O. C.

Array grammars and languages, hierarchy, rookwise connectedness, shearing effect, two-dimensional patterns, recognition.

1. Introduction

In recent years there appear several research papers and technical reports on two-dimensional languages using array grammars [1,3,4,,5,6,8, 10,11,12]. Because of its significant application in data processing of two-dimensional patterns this research topic has become more and more attracting. The purpose of this paper is to point out some mistakes in [12] and introduce a new hierarchy of array languages. I adapt the definitions and terminologies as emoloyed in [1,12].

2. Definitions and Results

Dfn 2.1.. An array grammar (AG) is a quintuple $G = (V_N, V_T, P, S, \#)$ where $V_N \neq \emptyset$ non-terminal set, $V_T \neq \emptyset$, $V_T \cap V_N = \emptyset$ terminal set, $\# \notin V_N \cup V_T$ blank symbol, $S \in V_N$ start symbol and P is a set of production (or rewriting or generating) rules. Each member of P is of the form $\alpha \rightarrow \beta$ and can be explained as follows : let J be a finite connected subset of I^2 where I is the set of integers. Let $i \in I^2$. Then α and β are mappings from J into $V_T \cup V_N \cup \{\#\} = V \ni$ if $\alpha(i) = a \in V_T$ then $\beta(i) = a$, i.e.

terminals are never rewitten.

An array A is a mapping : $I^2 \to V$. A production rule $\alpha \to \beta$ is applicable to A if a translation τ of the domain J of $\alpha \ni A \mid \tau J = \alpha$. We say A directly produces A' (or A' is directly derivable from A), written as $A \Rightarrow A'$, if for $\alpha \to \beta$ applicable to $A \mid \tau J = \alpha$, $A' \mid \tau J = \beta$ and $A' \mid (I^2 - \tau J) = A \mid (I^2 - \tau J)$. Let $\overset{*}{\Rightarrow}$ be the transitive closure of $\Rightarrow$. Then for $A \overset{*}{\Rightarrow} B$, B is said to be derivable from A and is called a sentential form.

An initial array A_S is a mapping $I \to \{\#, S\} \ni \{i \mid A_S(i) = S\}$ is a singleton. A terminal array A_T is a mapping form I^2 into $V_T \cup \{\#\}$. The array language generated by an AG is $L(G) = \left\{ B \mid A_S \overset{*}{\Rightarrow} B,\ B \text{ terminal} \right\}$. B is also called a sentence of the grammar.

In order to avoid a shearing effect [10], in each rule $\alpha \to \beta$, α and β are isotonic, i.e. geometrically identical. An AG is called monotonic if it can never erase, i.e. if for each rule $\alpha \to \beta$, $\beta(i) = \#$ then $\alpha(i) = \#$.

Dfn 2.2. Define the weight of an array W(A) to be the number of non # in the array. We say an AG is quasi-monotonic if for every rule $\alpha \to \beta$, $W(\beta) \geqslant W(\alpha)$.

Dfn. 2.3. An array language is said to be recursive if there exists an algorithm for recognizing the language. An array language is said to be recursively enumerable if there exists an effective procedure to list all and only all members of the array.

Now the author wants to show that in [12] Theorem 2.2 (The class of quasi-monotonic array languages ($\mathcal{L}(G_{qm})$) is recursive) and Theorem 2.3

(There exists a recursive array language L which is not quasi-monotonic) are both wrong.

This is due to the fact that there is no requirement in the paper that terminal arrays on sentential forms need be connected. Therefore the non-# symbols can be separated by an arbitrary number of #'s and the number of arrays of weight j is infinite. This capability can be used to simulate a finite automaton with a fixed number of counters. For instance, a counter can be simulated by a row of the array of the form

a # # # # # b

where the number of #'s between a and b represents the value of the counter. The array grammar can increment the counter, decrement the counter, and test for zero. It is well known [2] that finite automaton with two counters can simulate Turing machine and hence reocgnize all recursively enumerable sets. Therefore Theorem 2.2 and Theorem 2.3 are both wrong and should be modified as follows :

Theorem 2.2 of [12] : The class of quasi-monotonic array language ($\mathcal{L}$(Gqm)) is not recursive but identical to the class of isotonic array languages ($\mathcal{L}(G_I)$).

Theorem 2.3 of [12] : There exists an array language L which is recursive but not monotonic.

From the above, Theorem 2.6 of [12] should also be modified as follows :

Theorem 2.6 of [12] : $\mathcal{L}(G_m) \subsetneq \mathcal{L}(\text{recursive}) \subsetneq \mathcal{L}(G_{qm}) = \mathcal{L}(G_I)$
$= \mathcal{L}$ (recursively enumerable)

3. A New Hierarchy

In this section the author intends to find a new hierarchy of array grammars and languages.

Dfn 3.1 An array is connected (rookwise-connected) if there is a path of non-# symbols between every pair of non-#symbols.

Dfn 3.2 Let $G = (V_N, V_T, P, S, \#)$ be an isotonic array grammar.

(1) G is of Type 0 (isotonic, IAG) if there are not restrictions on P,

(2) G is of Type 1 (monotonic, MAG) if both sides of each rule are connected and the image of each left-side symbol is in $V_N \cup V_T$; that is, #'s cannot be created.

(3) G is of Type 2 (context-free, CFAG) if it is monotonic and the left side contains exactly one nonterminal symbol in a field of #'s ,

(4) G is of Type 3 (regular, RAG) if every rewriting rule is of the form

$$\#A \longrightarrow Ba\ ,\quad A\# \longrightarrow aB\ ,\quad \begin{matrix}\#\\A\end{matrix} \longrightarrow \begin{matrix}B\\a\end{matrix}\ ,\quad \begin{matrix}A\\\#\end{matrix} \longrightarrow \begin{matrix}a\\B\end{matrix}\ ,\quad \text{or } A \longrightarrow a$$

where $A, B \in V_N$ and $a \in V_T$.

Let IAL, MAL , CFAL and RAL denote the isotonic, monotonic, context-free, and regular array languages, respectively.

Milgram and Rosenfeld [5] defined Type 0 and Type 1 grammars and defined array acceptors for the corresponding languages. They also showed that the family of MALs ($\mathcal{L}$ (MAL)) is propely contained in the family of IALs ($\mathcal{L}$(IAL)). In fact $\mathcal{L}$(IAL)= $\mathcal{L}(G_I)$= $\mathcal{L}$(recursively enumerable). Rosenfeld [6] also showed that $\mathcal{L}(G_m)$ = $\mathcal{L}$ (MAL). In [9] Rosenfeld and Milgram gave several definitions of RAG's and finite state array acceptors, but were unsuccessful in obtaining a natural characterization of each as in the string case. They did not define CFAG's or any class of array grammars between monotonic and regular. However, later in [1] Cook and Wang did define Type 2 (CFAG) and found a Chomsky hierarchy, that is $\mathscr{L}(\text{RAL}) \subsetneq \mathscr{L}(\text{CFAL}) \subsetneq \mathscr{L}(\text{MAL}) \subsetneq \mathscr{L}(\text{IAL})$. In this section, I will introduce a new type of array grammars called "extended regular array grammar" (ERAG) and complete the Chomsky hierarchy of isotonic array grammars.

Dfn 3.3 An array grammar G is of Type 3' (extented regular, ERAG) if every rewriting rule is of the form of either Type 3 or

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

Let ERAL denote the extended regular array languages.

Theorem 3.1 The family of Type 3' languages ($\mathscr{L}$(ERAL)) properly contains the family of Type 3 languages ($\mathscr{L}$(RAL)).

Proof : The language $\left\{\begin{matrix}\# & a & \#\\ a & a & a\\ \# & a & \#\end{matrix}\right\}$ is not generated by any Type 3 grammar (Cook and Wang [1]) but is generated by the following Type 3' grammar :

$$G = (\{S, A, B, C, D\}, \{a\}, P, S, \#)$$

where $P = \left\{ S\# \rightarrow Aa\ ,\ \begin{matrix}\#\\A\end{matrix} \rightarrow \begin{matrix}a\\B\end{matrix}\ ,\ \#B \rightarrow aC\ ,\ \begin{matrix}C\\\#\end{matrix} \rightarrow \begin{matrix}D\\a\end{matrix}\ ,\ D \rightarrow a \right\}$

Q. E. D.

Theorem 3.2 $\mathscr{L}$ (ERAL) is properly contained in $\mathscr{L}$ (CFAL).

Proof : The language $\left\{\begin{matrix}\# & \# & a & \# & \#\\ \# & \# & a & \# & \#\\ a & a & a & a & a\\ \# & \# & a & \# & \#\\ \# & \# & a & \# & \#\end{matrix}\right\}$ is not generated by any ERAG, but is generated by the following CFAG : $G = (\{S\}, \{a\}, \left\{\begin{matrix}\# & \# & \# & \# & \#\\ \# & \# & \# & \# & \#\\ \# & \# & S & \# & \#\\ \# & \# & \# & \# & \#\\ \# & \# & \# & \# & \#\end{matrix}\right.$

$$\longrightarrow \left.\begin{matrix}\# & \# & a & \# & \#\\ \# & \# & a & \# & \#\\ a & a & a & a & a\\ \# & \# & a & \# & \#\\ \# & \# & a & \# & \#\end{matrix}\right\}, S, \#).$$

Q. E. D.

We thus complete a Chomsky hierarchy of isotonic array languages, that is

Theorem 3.3 $\mathscr{L}(\mathrm{RAL}) \subsetneq \mathscr{L}(\mathrm{ERAL}) \subsetneq \mathscr{L}(\mathrm{CFAL}) \subsetneq \mathscr{L}(\mathrm{MAL}) \subsetneq \mathscr{L}(\mathrm{IAL})$

Proof : From Theorem 3.1, Theorem 3.2 and Cook and Wang [1] . Q.E.D.

4. Discussion and Conclusions.

We have just completed a new Chomsky hierarchy of isotonic array languages by introducing a new type of array grammars between context free and regular. In fact, the following observations are quite interesting because they contradict to the aspects in one-dimensional string case :

(1) ERAG's are more powerful than RAG's (Theorem 3.1), whereas in 1-d case, the grammars of the forms of Type 3 is equivalent to the grammars of the forms of Type 3' [2].

(2) The family of ERAL's does not contain all finite arrays (as witnessed in Theorem 3.2), whereas in 1-d case, the family of regular languages contains all finite string languages [2] .

These results reconfirm the claim made by Rosenfeld [7] that finite state two-dimensional languages are considerably more complicated and much less well-behaved than their one-dimensional counterparts. It is the author's hope and belief that more interesting aspects will be found and that more interesting types of array languages as well as their recognition counterparts will be established.

References :

[1] C. R. Cook and P. S. P. Wang, A Chomsky Hierarchy of Isotonic Array Grammars and Languages, J of Computer Graphics and Image Processing, Vol 8 (1978) 144 - 152.

[2] J. E. Hopcroft and J. D. Ullman, Formal Languages and Their Relation to Automata (Addison-Wesley, Reading, Mass; 1969).

[3] R. A. Kirch, Computer Interpretation of English Text and Picture Patterns, IEEE Trans. Computers EC-14 (1964) 363 - 376.

[4] A. Mercer and A. Rosenfeld, An Array Grammar Programming System, Comm. ACM 16 (1973) 299 - 305.

[5] D. L. Milgram and A. Rosenfeld, Array Automata And Array Grammars, Proc. IFIP Congre-s 71, booklet TA - 2, North-Holland, Amsterdam (1971) 166 - 173.

[6] A. Rosenfeld, Array Grammar Normal Forms, Information and Control 23, (1973) 173 - 182.

[7] A. Rosenfeld, Some Notes on Finite-State Picture Languages, Information and Control 31, (1976) 177 - 184.

[8] A. Rosenfeld, Pebble, Pushdown, and Parallel-Sequential Picture Acceptors, (TR-633, Computer Sci. Dept, U. of Marylang, February, 1978).

[9] A. Rosenfeld and D. L. Milgram, Array Automata And Array Grammars, 2, (TR-171, Computer Sci. Center, U. of Maryland, September 1971).

[10] P. S. P. Wang, Sequential/Parallel Matrix Array Languages, Journal of Cybernetics, vol 5.4, (1975) 19 - 36.

[11] P. S. P. Wang, Recognition of Two-Dimensional Patterns, Proc. of ACM'77, Seattle, Wa (1977) 484 - 489.

[12] P. S. P. Wang and W. I. Grosky, Recursiveness of Monotonic Array Grammars and A Hierarchy of Array Languages, Information Processing Leggers, Vol 5.1 (1976), 24 - 26, and Vol 5.3 (1976) 90.