CIS-TR-80-1

Syntactic Structures of Parallel Isometric Array Patterns

by

Patrick Shen-pei Wang

Department of Computer and Information Science
University of Oregon
Eugene, Oregon 97403

Abstract

The relationship between parallel isometric array languages and sequential isometric array languages is examined. Their hierarchical structures are investigated and a hierarchy is completed by introducing parallel context-free array languages (PCFAL), derivation bounded array languages (DBAL), linear array languages (LAL) and extended regular array languages (ERAL). It is interesting to find that some fundamental aspects that hold in 1-dimensional string languages do not hold in their 2-dimensional counterparts.

Some parsing techniques are also explored. It is shown that while parallel parsing grammars may be much simpler to write and parallel processing usually takes less time than sequential ones, the nature of parallel parsing is very complicated.

Finally, several future research topics concerning with parallel isometric array languages including their complexities, hierarchical structures and application to pattern recognition are discussed.

Index Terms - parallel isometric languages, parallel context-free array languages, hierarchy, generation, parsing, automaton, pattern recognition, array grammars.

1. Introduction

Recently the study of 2-dimensional grammars and languages has become more and more important and interesting since it has significant applications in the data processing of 2-dimensional patterns[12, 17, 23]. That few studies have been done in this field is surprising considering numerous and extensive studies in the area of 1-dimensional or string grammars and languages. This is probably due to the fact that a reasonable grammar for 2-dimensional language is very difficult to obtain by natural extension of the grammars for 1-dimensional languages. For instance, in the string case when a 1-dimensional rule, say $A \to BCD$, is applied to a sentential form $\alpha A \beta$ we get $\alpha BCD\beta$. This can be done by either pushing α to the left or pushing β to the right. When the application of a rule is extended to 2-dimensional case we are immediately in trouble. For example, if we apply the rule $A \to \begin{matrix}B\\CD\end{matrix}$ to the sentential form

```
XXX
YAY  we may obtain
ZZZ
                                 X
                  XXX           XBX         XXX
                  YBY    or     YCDY   or   YBY   or ...
                  ZCDZ          ZZZ         ZCD
                   Z                         ZZ
```

The shearing effect [23] occurs and what we really want might be totally distorted. Therefore, to promote research on 2-dimensional languages and patterns we need to find an appropriate general mathematical model of the grammar for such languages and patterns.

In 1964 Kirsch first proposed a model called "array grammar"[10], which falls into this category. A similar formal system is established and grammars for languages consisting of classes of polygons are exhibited by Dacey [3,4]. There also has been work on the relation of array grammars to other topics. In [14,16,17]Rosenfeld has examined some relationships between array and string grammars. Automata whose tapes are 2-dimensional are studied in Blum and Hewitt

[1], Fischer [5] and Wang [24] and the relation of such automata to array grammars is discussed in Smith [22], Cook and Wang [2] and Rosenfeld [17].

However, the fundamental properties of array grammars such as hierarchical structures and parsing techniques remain largely unknown. Thus, to promote research on 2-dimensional languages and patterns using array grammars we need to further investigate and understand more thoroughly the fundamental properties of array grammars. This is one of the motivations of this research paper.

The other is the parallelism. In Mercer and Rosenfeld [11] a parallel array grammar programming system is designed and it is found that parallel derivation takes less time and that parallel array grammars are usually simpler to write than sequential ones. Furthermore, parallel technique is especially natural for array grammars since local picture operations are often applied to digital pictures in parallel as pointed out in Rosenfeld and Pfaltz [18]. However, very little is known about the basic aspects and phenomena of parallel array grammars except that in[14] Rosenfeld has shown that any parallel monotonic array language is a monotonic array language and vice versa. For instance, the relationship between parallel context-free array languages and context-free array languages as introduced by Cook and Wang [2] remains unknown.

The main purpose of this research paper is therefore two-fold:

(1) to provide some basic backgrounds that will lead to further investigation of the fundamental properties of array grammars and languages, and

(2) to study the parallel techniques employed for array language generation and parsing.

The author also intends to solve some open questions mentioned in this section as the subgoal.

In section 2 some definitions and notations are revisited. In section 3, some fundamental aspects of array languages are investigated. It is interesting to find that some properties that hold in 1-dimensional string languages do not hold in their 2-dimensional counterparts. Section 4 explores some parsing techniques for array languages and it is observed that parallel parsing, while advantageous, is extremely complicated in nature. Finally, some future research topics are discussed in section 5.

2. Array Grammars and Languages

Definition 2.1 An array grammar is a quintuple $G = (V_N, V_T, P, S, \#)$ where

V_N is a nonempty, finite set of non-terminals,

V_T is a nonempty, finite set of terminals, $V_T \cap V_N = \emptyset$,

$S \in V_N$ is the start symbol, $\# \notin V_N \cup V_T$ is the blank symbol, and

P is a set of production(or rewriting, 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 $V_T \cup V_N \cup \{\#\} = V$ such that if $\alpha(i) = a \in V_T$ then $\beta(i) = a$, i.e. terminals are never rewritten.

An array A is a mapping : $I^2 \Rightarrow V$. A production rule $\alpha \rightarrow \beta$ is applicable to A if $\exists$ a translation τ of the domain J of α such that $A \mid \tau J = \alpha$. We say A directly produces A' (or A' is directly derivable from A), written as $A \Rightarrow A'$, if for $\alpha \rightarrow \beta$ applicable to A, $A' \mid \tau J = \beta$ and $A' \mid (I^2 - \tau J) = A \mid (I^2 - \tau J)$. Let $\overset{*}{\Rightarrow}$ be the transitive and reflexive 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 $I^2 \rightarrow \{\#, S\}$ such that $\{ i \mid A_S(i) = S \}$ is a singleton. A terminal array A_T is a mapping from I^2 into $V_T \cup \{\#\}$. The array language generated by an array grammar G is $L(G) = \{ B \mid A_S \overset{*}{\Rightarrow} B$, B a terminal array $\}$.

$By connected we mean "rookwise-connected", i.e., points (i,j) and (i',j') are connected iff $|i - i'| + |j - j'| \leq 1$. A subset K of I^2 is connected iff, for any two points p and q in K, there is a sequence of points $p_1, p_2, \cdots p_n$ in K with $p_1 = p$, $p_n = q$ and p_i connected to p_{i+1}, $1 \leq i \leq n-1$.

In order to avoid the shearing effect [23], in each rule $\alpha \to \beta$, α and β are isometric, i.e. geometrically identical. An array grammar is called monotonic if it can never erase, i.e. if for each rule $\alpha \to \beta$, $\beta(i) = \#$ then $\alpha(i) = \#$.

Definition 2.2 Let $G = (V_N, V_T, P, S, \#)$ be an isometric array grammar. Then G is of

(1) Type 0 (isometric, IAG) if there is no restriction on P, or

(2) Type 1 (monotonic, MAG) if G is monotonic and both sides of each rule are connected, or

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

(4) Type 3 (regular, RAG) if every rewriting rule is in one of the following forms:

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

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

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

A possible modification to the definition of a derivation in an array grammar is parallel rule application: i.e. all instances of the rule's antecedent are simultaneously replaced by the consequent (rather than just one instance). Let the language generated by an array grammar G in parallel be denoted by $L_p(G)$. Let PIAL, PMAL, PCFAL and PRAL denote the parallel languages generated by IAG, MAG, CFAG and RAG respectively.

3. Parallel Context-Free Array Languages

We first observe that in string case, given a context free grammar G, the language generated by G sequentially (denoted by L(G)) always covers the language generated by G in parallel (denoted by $L_p(G)$), i.e. $\forall$ CFG G $(L(G) \supseteq L_p(G))$. This is true because if a sentential form, say, α contains more than one occurrence of the same nonterminal, say, A, then what we can obtain by applying a rule, say, $A \to \beta$, to all instances of A simultaneously (in parallel) can also be obtained by applying the same rule $A \rightarrow \beta$ to all A's one by one (sequentially). Therefore, whatever a parallel derivation can obtain can also be obtained sequentially. However, it is not so in array languages.

Example 3.1 There exists a CFAG G such that both L(G) and $L_p(G)$ are not empty and $L(G) \cap L_p(G) = \emptyset$.

Let $G = (\{S,A\}, \{a\}, P, S, \#)$ where

$$P \text{ is : } \left\{ \begin{array}{rl} \begin{array}{r} \# \\ S\# \\ \# \end{array} \rightarrow \begin{array}{c} A \\ aa \\ A \end{array} , & \begin{array}{r} \# \\ \# \\ \# \\ A\#\# \end{array} \rightarrow \begin{array}{r} a \\ a \\ a \\ aaa \end{array} , \quad \begin{array}{r} \# \\ \# \\ \# \\ A\#\#\#\# \end{array} \rightarrow \begin{array}{r} a \\ a \\ a \\ aaaaa \end{array} \end{array} \right\}$$

It can be seen that $L(G) = \left\{ \begin{array}{r} a \\ a \\ aa \\ aaaa \\ aa\;\;\;a \\ aaaaa \end{array} \right\}$ and $L_p(G) = \left\{ \begin{array}{r} a \\ a \\ aa \\ aaaaa \\ aa\;\;\;a\; \\ aaaaa\; \end{array} , \quad \begin{array}{r} a \\ a \\ aa \\ aaa \\ aaa\; \\ aaa\; \end{array} \right\}$.

Therefore $L(G) \cap L_p(G) = \emptyset$. Please notice the difference between parallel derivations [Definition 2.2, page 4] and sequential derivations [2,14,15,17] for isometric array grammars. Furthermore, we can see the following:

Lemma 3.1 (A Small Pumping Lemma) Let $L_U = \left\{ n\left\{ \begin{array}{ccc} a & & a \\ \vdots & & \vdots \\ \underbrace{a \dots a}_{m} \end{array} \right\} n \;\middle|\; m,n \geq 2 \right\}$,i.e. all equal-armed U's. If a CFAG G can generate all the sentences of L_U, then it shall also generate

$$L_{U'} = \left\{ n\left\{ \begin{array}{ccc} a & & a \\ \vdots & & \vdots \\ \underbrace{a \dots a}_{m} \end{array} \right\} p \;\middle|\; p \neq n,\ n,p \geq k \text{ for some } k \geq 2, m \geq 2 \right\}.$$

Proof: Let $G = (V_N, V_T, P, S, \#)$ be a CFAG that can generate all members of L_U. Consider the figure below: (Figure 3.1)

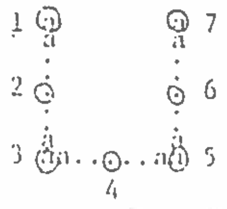
Figure 3.1 An equal-armed "U"

In order to generate this pattern there are 7 possible positions to start with as shown above. Notice that because of its symmetry we need only consider points 1,2,3 and 4.

Assume starting from point 1, then there must be rules which should enable derivations downward, then horizontal towards right, then upwards. Since there is essentially no control over how far it will derive downward and upward, it must also generate all members of L_U. Similar arguments hold for situations starting from points 2,3 and 4.

Q.E.D.

Lemma 3.2 There is a PCFAL which is not CFAL.

Proof: We claim that L_U is such a language. Let $G = (\{S,A,B\}, \{a\}, P, S, \#)$, where $P = \left\{ S\# \rightarrow AA,\ A\# \rightarrow aA,\ \overset{\#}{A} \rightarrow \overset{B}{a},\ \overset{\#}{B} \rightarrow \overset{B}{a},\ B \rightarrow a \right\}$. Clearly $L_p(G) = L_U$ and from Lemma 3.1 L_U is not a CFAL.

Q.E.D.

Lemma 3.3 There is a CFAL which is not PCFAL.

Proof: Let $G = \left(\{S,A\}, \{a\}, \left\{ S\#\# \rightarrow SaS,\ \underset{S}{\overset{\#}{\#}} \rightarrow \underset{S}{\overset{S}{a}},\ S \rightarrow a \right\}, S, \# \right)$. It can be seen that L(G) is a maze-like language, denoted as L_m. For instance, the following pattern is in the language : (Figure 3.2)

```
a         a
a         a
a aaa     a a a
a a       a a a
aaa aaa a aaa a a
a   a   a a   a a
a aaaaa a a a aaa
a a     a a a a
a a aaa a aaa a
a a a   a a   a
aaaaaaaaaaaaaaa
```
$\in L_m$

Figure 3.2 A maze-like pattern

We claim L_m is a CFAL but not a PCFAL. A detailed proof is rather tedious. Here we just outline the basic principles behind.

To generate this pattern, suppose we start from the lower leftmost corner. Realizing its horizontal and vertical patterns, we see that there must be rules of the forms (1) $\underset{S}{\overset{\#}{\#}} \rightarrow \underset{A}{\overset{B}{a}}$ and (2) $S\#\# \rightarrow CaD$. Since these patterns can essentially be repeated to the right and upward as far as one wants , these rules must be recursive, i.e. B = S and D = S.

Furthermore, all generated A's should have the capability to derive the same pattern as by rule (2) and all generated C's should have the capability to derive the same pattern as by rule (1). Therefore, there must be rules of the forms (3) $A\#\# \rightarrow EaF$ and (4) $\underset{C}{\overset{\#}{\#}} \rightarrow \underset{G}{\overset{H}{a}}$. Similarly, because these patterns can essentially be repeated to as far as one wants , these rules must be recursive, i.e. F = A and H = C.

Continuing this argument we obtain the following situations. In order to generate L_m in parallel, (i) we need an infinite number of non-terminals and rules, or (ii) if the number of non-terminals and rules is finite, then we can not avoid derivations whose sentential forms involve more than one occurrence of the same non-terminals.

Either of the above situations leads us to conclude that there is no any CFAG G such that $L_p(G) = L_m$.

Similar argument is held if we assume starting derivation from any other locations of the pattern.

Q.E.D.

Let $\mathcal{L}$(PCFAL) denote the family of PCFAL's, $\mathcal{L}$(CFAL) denote the family of CFAL's and so forth. From Lemmas 3.2 and 3.3 we immediately see that

Theorem 3.1 $\mathcal{L}(\text{PCFAL}) \overset{\not\supseteq}{\not\subseteq} \mathcal{L}(\text{CFAL})$.

It would be interesting to ask what $\mathcal{L}(\text{PCFAL}) \cap \mathcal{L}(\text{CFAL})$ is. Evidently it is not empty. Let's define a CFAL L(G) to be derivation bounded (DBAL) if for each sentence in L(G), its derivation involves no more than k occurrences of the same non-terminal in any sentential form. Then following the similar argument by Siromoney and Krithivasan [20] we see that

Theorem 3.2 $\mathcal{L}(\text{PCFAL}) \cap \mathcal{L}(\text{CFAL}) = \mathcal{L}(\text{DBAL})$.

Another interesting question would be : is there any MAL which is not a PCFAL or CFAL? To answer this question we first need a lamma.

Lemma 3.4 (Another small pumping lemma) Let $L = \left\{ \#^{\infty} a^n b^n \#^{\infty} \mid n \geq 1 \right\}$. A CFAG G that can derive all sentences in L should also generate all sentences in $\left\{ \#^{\infty} a^m b^n \#^{\infty} \mid m,n \geq k,\ m \neq n \text{ for some constant } k \geq 1 \right\}$.

Proof: The argument here is similar to that of Lemma 3.1. Let $G=(V_N, V_T, P, S, \#)$ be a CFAG such that $L_p(G)=L$. Consider the following pattern: (Figure 3.3)

$$\#^{\infty}\ \underset{\substack{\uparrow\\1}}{a}\overbrace{a\ldots\underset{\substack{\uparrow\\2}}{a}\ldots a}^{n}\underset{\substack{\uparrow\\3}}{a}\overbrace{b b\ldots \underset{\substack{\uparrow\\4}}{b}\ldots b}^{n}\underset{\substack{\uparrow\\5}}{b}\ \#^{\infty}$$

Figure 3.3 $\#^{\infty} a^n b^n \#^{\infty}$

In order to generate this pattern there are 5 possible places to start with as shown above. Notice that because of its symmetry, we need only consider locations 1,2 and 3.

Suppose the derivation starts from location 1. There must be rules of the form $S\# \rightarrow aS$, $S\# \rightarrow aS_1$, $S_1\# \rightarrow bS_1$ and $S_1 \rightarrow b$. Since both strings of "a" and strings of "b" can be extended arbitrarily long, there is essentially no control over how far each string will be extended (unless using context-sensitive rules). Notice here that the parallelism does not give us any special privileges for a one-dimensional array, and the effect of parallel derivations makes no difference to sequential ones. Therefore it should also be able to derive all members of $\{ \#^{\infty} a^m b^n \#^{\infty} | m,n \geq k,\ m \neq n$ for some constant $k \geq 1\}$.

Similarly for derivations starting from locations 2 and 3.

Q.E.D.

Lemma 3.5 There is an MAL which is not a PCFAL or CFAL.

Proof: We claim that $L = \{ \#^{\infty} a^n b^n \#^{\infty} \mid n \geq 1 \}$ is such a language. This can be seen directly from Cook and Wang [2] that L is an MAL but not a CFAL and from Lemma 3.4 that L is not a PCFAL.

Q.E.D.

Theorem 3.3 $\mathcal{L}$(PCFAL) $\cup$ $\mathcal{L}$(CFAL) $\subsetneq$ $\mathcal{L}$(MAL).

Proof: This can be seen from Lemma 3.5 and the fact that PCFAL is a special case of PMAL and CFAL is a special case of MAL and from Rosenfeld [15] that $\mathcal{L}$(PMAL) = $\mathcal{L}$(MAL).

Q.E.D.

Now I am going to introduce two new array languages.

Definition 3.1 An array grammar G is called linear (LAG) if every rule is of type 2 and the right hand side contains at most one nonterminal in a filed of $V_T \cup \{\#\}$.

Definition 3.2 An array grammar G is called extended regular (ERAG) if every rule is of the form of eith type 3 or $\#A \rightarrow aB$, $A\# \rightarrow Ba$, $\begin{matrix}\# \\ A\end{matrix} \rightarrow \begin{matrix}a \\ B\end{matrix}$, or $\begin{matrix}A \\ \#\end{matrix} \rightarrow \begin{matrix}B \\ a\end{matrix}$.

Let LAL and ERAL denote the linear array language and extended regular array language respectively.

Lemma 3.6 There is an ERAL which is not an RAL.

Proof: The language $\left\{\begin{matrix} \# & a & \# \\ a & a & a \\ \# & a & \# \end{matrix}\right\}$ is not generated by any RAG (Cook and Wang[2]) but is generated by the following ERAG: $G = (\{S,A,B,C,D\}, \{a\}, P, S, \#)$ where $P = \left\{ S\# \to Aa,\ \begin{matrix}\#\\A\end{matrix} \to \begin{matrix}a\\B\end{matrix},\ \#B \to aC,\ \begin{matrix}C\\\#\end{matrix} \to \begin{matrix}D\\a\end{matrix},\ D \to a \right\}$. Q.E.D.

Lemma 3.7 There is an LAL which is not an ERAL.

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 LAG : $G=\left(\{S\},\{a\},\left\{\begin{matrix} \# & \# & \# & \# & \# \\ \# & \# & \# & \# & \# \\ \# & \# & S & \# & \# \\ \# & \# & \# & \# & \# \\ \# & \# & \# & \# & \# \end{matrix} \to \begin{matrix} \# & \# & a & \# & \# \\ \# & \# & a & \# & \# \\ a & a & a & a & a \\ \# & \# & a & \# & \# \\ \# & \# & a & \# & \# \end{matrix}\right\}, S,\#\right)$. Q.E.D.

Lemma 3.8 There is a DBAL which is not an LAL.

Proof: We claim that $L_T = \left\{ \begin{array}{c} \overbrace{aa\ldots a}^{n}a\overbrace{a\ldots aa}^{m} \\ \left.\begin{matrix} a \\ \vdots \\ a \\ a \end{matrix}\right\} p \end{array} \;\middle|\; m,n,p \geq 1 \right\}$ is such a language.

Suppose L_T is an LAL. In order to generate it, there are essentially 7 possible locations to start with as shown in Figure 3.4:

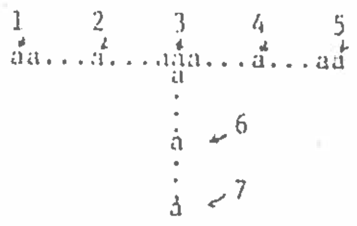

Figure 3.4 a member of L_T

Because of its symmetry we need only consider locations 1,2,3,6 and 7. Suppose the derivation starts from location 1. Then there must be a rule of the form $S\# \to aS$ since n could be as large as one wants. Once this derivation reaches location 3, it faces a decision as for whether to continue derivation rightward or downward. If it continues to derive downward, then it can only extend a finite number of "right" arms, i.e. $m \leq k$ for some constant k since there can be at most one nonterminal during derivation. Similarly, if it continues derivation rightward, then it can only extend a finite number of "downward arms", i.e. $p \leq k$ for some constant k. This shows that it can not generate L_T.

Similar arguments hold if derivations start from other locations 2,3,6 or 7. Therefore L_T is not an LAL. However, it is a DBAL generated by the following DBAG

$G_T = (\{S,A,B\}, \{a\}, P, S, \#)$, where $P = \left\{ \begin{matrix}\# & S & \# \\ & \# & \end{matrix} \rightarrow \begin{matrix} A & a & A \\ & B & \end{matrix},\ A\# \rightarrow aA,\ \#A \rightarrow Aa,\ \begin{matrix} B \\ \# \end{matrix} \rightarrow \begin{matrix} a \\ B \end{matrix},\ A \rightarrow a,\ B \rightarrow a \right\}$.

Q.E.D.

From Rosenfeld[15,17], Cook and Wang[2] and the fact that every RAG (ERAG, LAG) involves no sentential forms with more than one nonterminal, we conclude the following hierarchy for array languages:

Theorem 3.4

$$\begin{array}{ccccccccccccc} \mathcal{L}(\text{IAL}) & \supsetneq & \mathcal{L}(\text{MAL}) & \supsetneq & \mathcal{L}(\text{CFAL}) & \supsetneq & \mathcal{L}(\text{DBAL}) & \supsetneq & \mathcal{L}(\text{LAL}) & \supsetneq & \mathcal{L}(\text{ERAL}) & \supsetneq & \mathcal{L}(\text{RAL}) \\ = & & = & & \genfrac{}{}{0pt}{}{\not\supset}{\not\subset} & & = & & = & & = & & = \\ \mathcal{L}(\text{PIAL}) & & \mathcal{L}(\text{PMAL}) & & \mathcal{L}(\text{PCFAL}) & & \mathcal{L}(\text{CFAL}) \cap \mathcal{L}(\text{PCFAL}) & & \mathcal{L}(\text{PLAL}) & & \mathcal{L}(\text{PERAL}) & & \mathcal{L}(\text{PRAL}) \end{array}$$

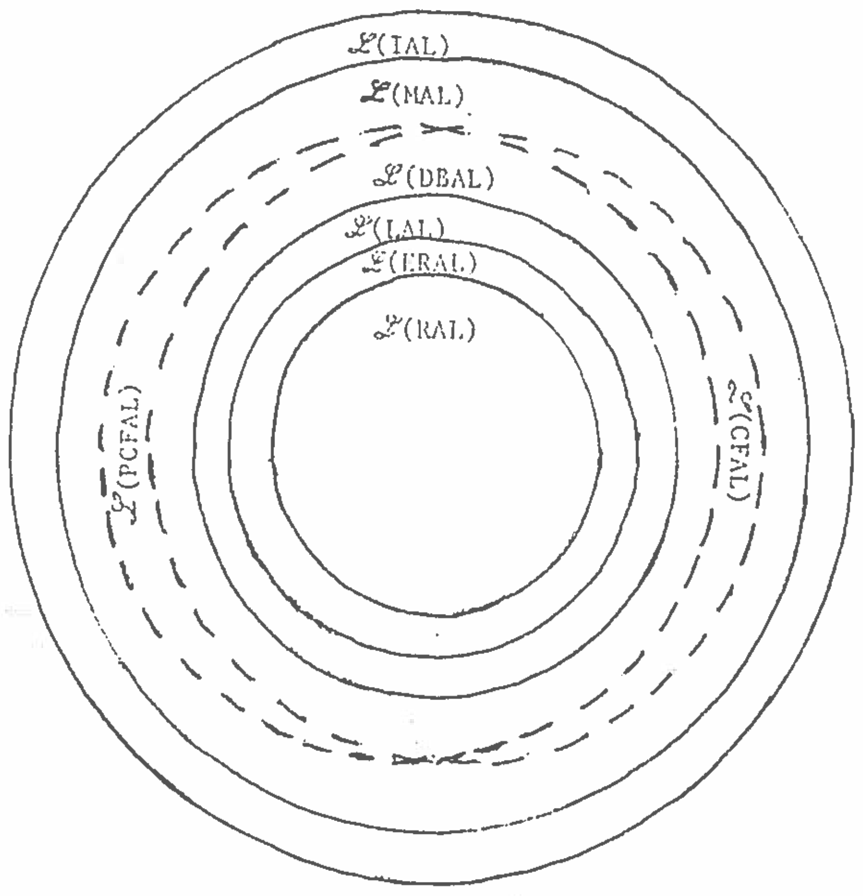

Figure 3.5 a hierarchy of isometric array languages

4. Complexities of Generation and Parsing

This section discusses the recognition and generation of PCFAL's. In

[14]it has been shown that given an MAG G, L(G), $L_p(G)$, the language parsed sequentially (denoted as $L^p(G)$) and the language parsed in parallel (denoted as $L_p^p(G)$) may not be the same; and in Mercer and Rosenfeld[11] several examples of parsing grammars are given. But nothing has been mentioned concerning with how a parsing grammar could be obtained from a given array grammar. This open question will be further investigated in part for PCFAL's.

We first observe that while L_U introduced in Lemma 3.1 is a PCFAL, another very similar language L_u defined in Example 4.1 is not a PCFAL.

Example 4.1 Let $L_u = \left\{ n \left\{ \begin{matrix} a & & c \\ \vdots & & \vdots \\ a & & c \\ \multicolumn{3}{c}{\underbrace{b \ldots b}_{m}} \end{matrix} \right\} n \;\middle|\; m,n \geq 2 \right\}$.

From a similar reason as Lemma 3.1 we can see that if a PCFAG G can derive all members of L_u, then it should also be able to derive all members of

$$L_{u'} = \left\{ n \left\{ \begin{matrix} a & & c \\ \vdots & & \vdots \\ a & & c \\ \multicolumn{3}{c}{\underbrace{b \ldots b}_{m}} \end{matrix} \right\} p \;\middle|\; m \geq 2,\ n \neq p,\ n,p \geq k \text{ for some } k \geq 2 \right\}.$$

Therefore L_u couldn't be a PCFAL.

This shows that the nature of a PCFAL not only depends on its pattern (shape) but also depends on its vocabulary(content). This is different from 1-dimensional string case,where a^nb^n and a^na^n, $n \geq 1$ both are in the same hierarchical category, i.e. context free.

To parse a PCFAL is even more complicated. The key point is that, given a 2-dimensional input array, we usually don't know where to start the parsing process with. We will demonstrate this complexity by using two possible settings for array parsers, one is "parsing grammar", the other is "automaton".

Let's define a parsing grammar to be a quintuple $G = (V_N, V_T, P, S, \#)$, where each component has exactly the same meaning as an array grammar except that all the rules in P have the form $\alpha \leftarrow \beta$. The language parsed (recognized, accepted) by G is defined as $L^p(G) = \left\{ \mathcal{O}\!\iota \,\middle|\, \mathcal{O}\!\iota \overset{*}{\underset{G}{\Rightarrow}} A_S,\ \mathcal{O}\!\iota \text{ connected terminal array} \right\}$, or $L_p^p(G)$ if parsed in parallel. For convenience, let's use the notation $L^p(G)$ to denote the language parsed by the parsing grammar obtained from reversing all the arrows of the rules of the generating grammar G. Similarly for $L_p^p(G)$.

Evidently, given a CFAG G, the language generated and the language parsed in parallel may not be the same, i.e. $L_p(G) \neq L_p^p(G)$ in general. This can be seen from the following example.

<u>Example 4.2</u> Let G_U be the CFAG stated in <u>Lemma 3.2</u> such that $L_p(G_U)=L_U$. Evidently $L_p^p(G_U) \neq L_U$. In fact $L_p^p(G_U) = \emptyset$. For instance, to parse an "U" with height n and width m, we immediately obtain an "U" of B's and then can not proceed any further as shown in the following figure: (<u>Figure 4.1</u>)

$$n\left\{\begin{matrix} a & & a \\ a & & a \\ \vdots & & \vdots \\ a & \cdots & a \end{matrix}\right\}n \quad \underset{\text{apply rule } B \leftarrow a}{\Longrightarrow} \quad n\left\{\begin{matrix} B & & B \\ B & & B \\ \vdots & & \vdots \\ B & \cdots & B \end{matrix}\right\}n$$

$$\underbrace{\qquad\quad}_{m} \qquad\qquad\qquad\qquad \underbrace{\qquad\quad}_{m}$$

<u>Figure 4.1</u>

The above difficulty can be overcome by properly taking care of the "boundary condition", i.e. the boundary environment of sentential arrays during derivation process. This usually will garantee the parsing process will start from the appropriate locations and then the subsequent parsing will continue as a reverse derivation process. Take L_U for instance. Realizing that the "boundary condition" of L_U is that when the terminal rule $B \rightarrow a$ is applied, all B's terminate and are replaced by a's in such an environment that each ending "a" has "#" as its west, north and east neighbor, one can design a parsing grammar by appropriately modifying rule $B \rightarrow a$ of G_U as in the following example.

<u>Example 4.3</u> Let $G = (\{S,A,B\}, \{a\}\ P,S,\#)$ be a parsing grammar, where

$$P = \left\{ (1)\,S\# \leftarrow AA,\ (2)\,A\# \leftarrow aA,\ (3)\,\begin{matrix}\#\\A\end{matrix} \leftarrow \begin{matrix}B\\a\end{matrix},\ (4)\,\begin{matrix}\#\\B\end{matrix} \leftarrow \begin{matrix}B\\a\end{matrix},\ (5)\,\begin{matrix}\#\\\#B\#\end{matrix} \leftarrow \begin{matrix}\#\\\#a\#\end{matrix} \right\}.$$

It can be seen that $L_p^p(G)=L_U$. Notice that rule (5) takes care of the boundary condition.

Since L_U is not a CFAL, to find a grammar $\bar{G}$ to generate L_U sequentially, $\bar{G}$ must be at least context-sensitive, i.e. monotonic.

Example 4.4 Let $\bar{G}$ = ($\{S,A,B,D\}$, $\{a\}$, P, S , #), where

$$P = \left\{ \begin{array}{l} (1)\ \begin{array}{cc} S & \# \\ \# & \end{array} \to \begin{array}{cc} A & S \\ A & \end{array},\ (2)\ \begin{array}{cc} S & \# \\ A & \end{array} \to \begin{array}{cc} A & B \\ D & \end{array},\ (3)\ \begin{array}{c} \# \\ B \end{array} \to \begin{array}{c} B \\ D \end{array},\ (4)\ AD \to DA,\ (5)\ \begin{array}{c} D \\ A \end{array} \to \begin{array}{c} A \\ D \end{array}, \\ (6)\ \#AD \to AA,\ \ (7)\ D \to A,\ \ (8)\ \begin{array}{c} \# \\ D \\ A \end{array} \to \begin{array}{c} A \\ A \\ A \end{array},\ (9)\ A \to a,\ (10)\ B \to a \end{array} \right\}.$$

It can be seen that $L(\bar{G}) = L^P(\bar{G}) = L_U$. However, we observe that in order to parse an input array (U) with height n and width m, it takes m + m.n + n(n+1) steps if $\bar{G}$ is used while it takes only m + n steps if G is used (in parallel). Furthermore $\bar{G}$ uses more non-terminals and rules, and its structure is more complicated than that of G's. Let G_U denote the grammar in Lemma 3.2 that generates L_U. The comparison between parallel and sequential parsing can be summarized below: Table 4.1

	G_U	G(parsing)	$\bar{G}$	$\bar{G}$(parsing)
no. of nonterminals	3	3	4	4
no. of rules	5	5	10	10
structure of rules	CFAG	simpler	MAG	more complicated
time complexity	O(m+n)	O(m+n)	O(m.n)	O(m.n)

Table 4.1 Comparison of sequential and parallel parsing

This idea works for a rather large class of PCFAL's. The following illustrates how it works for some electrical networks.

Example 4.5 Let G = ($\{S,S',+'\}$, $\{R,C,+\}$, P , S , #) be a CFAG, where

$$P:\quad (1)\ \begin{array}{c} \# \\ \# \\ S \end{array} \to \begin{array}{c} S' \\ R \\ + \end{array} \qquad\qquad (2)\ \begin{array}{c} \# \\ \# \\ S' \end{array} \to \begin{array}{c} S' \\ R \\ S' \end{array}$$

$$(3)\ \begin{array}{ccc} S' & \# & \# \\ & & \# \end{array} \to \begin{array}{ccc} + & C & +' \\ & & R \end{array} \qquad (4)\ \begin{array}{ccc} +' & \# & \# \\ & & \# \end{array} \to \begin{array}{ccc} + & C & +' \\ & & R \end{array} \qquad (5)\ +' \to +$$

If we interpret the primitives (terminals) as R= ⌇ , C= ⊣⊢ , and + = node, then $L_p(G)$ is the set of all grid networks with capacitors on the horizontal lines and resistors on the vertical lines. A derivation is shown as follows: (Figure 4.2)

```
                               S'        +C+'          +C+C+C+C+C+'
                               R         R R           R R R R R R
                               S'        +C+'          +C+C+C+C+C+'
                               R         R R           R R R R R R
                               S'        +C+'          +C+C+C+C+C+'
                               R         R R       4   R R R R R R
#  (1)   S'   (2)(2)(2)        S'  (3)   +C+'   (4)    +C+C+C+C+C+'
#  ==>   R    =========>       R   ==>   R R    ==>    R R R R R R
S        +                     +         +             +

                                                       +C+C+C+C+C+
                                                       R R R R R R
                                                       +C+C+C+C+C+
                                                       R R R R R R
                                                (5)    +C+C+C+C+C+
                                                ===>   R R R R R R
                                                       +C+C+C+C+C+ ∈ L (G)
                                                       R R R R R R    p
                                                       +
```

Figure 4.2 A derivation of $L_p(G)$

which can be visualized as in Figure 4.3.

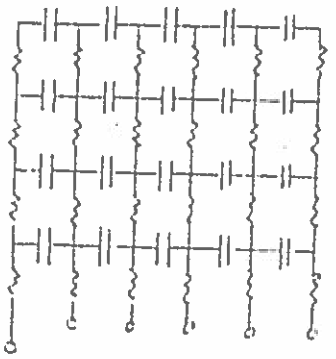

Figure 4.3 A network as a member of $L_p(G)$

In order to recognize this network, we shall modify rules (1), (2), (5) and reverse rules (3) and (4) to obtain a parsing grammar G' as follows:

$G' = (\{S, S', +'\}, \{R, C, +\}, P', S, \#)$, where P' is : (1) $\begin{array}{c}\#\\\#\\\#\\S\end{array} \leftarrow \begin{array}{c}\#\\S'\\R\\+\end{array}$ (2) $\begin{array}{c}\#\\\#\\\#\\S'\end{array} \leftarrow \begin{array}{c}\#\\S'\\R\\S'\end{array}$

(3) $\begin{array}{r}S'\#\#\\\#\end{array} \leftarrow \begin{array}{c}+C+'\\R\end{array}$ (4) $\begin{array}{r}+'\#\#\\\#\end{array} \leftarrow \begin{array}{c}+C+'\\R\end{array}$ (5) $+'\# \leftarrow +\#$

Notice that while rule (5) of P' takes care of the boundary condition concerning with the terminal rule, rules (1) and (2) take care of the boundary conditions involving non-terminals also. It can be seen that $L_p^p(G')=L_p(G)$.

However, while parallel parsing grammars usually are simpler to write and take less time than sequential ones, the above idea to obtain them does not work for some PCFAL's. This can be demonstrated in the following examples.

Example 4.6 Let $G_1 = (\{S\}, \{a\}, \{S\#\# \Rightarrow SS\#, \begin{array}{c}\#\\\#\\S\end{array} \rightarrow \begin{array}{c}\#\\S\\S\end{array}, S \rightarrow a\}, S, \#)$. It can be seen that $L_p(G_1)$ is the set of all rectangles. Clearly $L_p^p(G_1) \neq L_p(G_1)$. In fact $L_p^p(G_1)$ also contains all "L"-shaped patterns. For instance, $\begin{array}{l}a\\aaaa\end{array} \in L_p^p(G_1)$. It turns out that the "boundary condition" technique does not work for G_1. However, it works for an equivalent grammar G_2 in the following example.

Example 4.7 Let $G_2 = (\{S, A\}, \{a\}, \{(1) S\#\# \Rightarrow SS\#, (2) \begin{array}{c}\#\\S\end{array} \rightarrow \begin{array}{c}A\\a\end{array}, (3) \begin{array}{c}\#\\A\end{array} \rightarrow \begin{array}{c}A\\a\end{array}, (4) A \rightarrow a\},$ $S, \#)$. Clearly $L_p(G_2) = L_p(G_1)$. Realizing the boundary condition, we can modify rule (4) and construct a parsing grammar G_3 as follows:

$G_3 = (\{S, A\}, \{a\}, \{(1) S\#\# \Leftarrow SS\#, (2) \begin{array}{c}\#\\S\end{array} \Leftarrow \begin{array}{c}A\\a\end{array}, (3) \begin{array}{c}\#\\A\end{array} \Leftarrow \begin{array}{c}A\\a\end{array}, (4) \begin{array}{c}\#\\A\end{array} \Leftarrow \begin{array}{c}\#\\a\end{array}\}, S, \#)$.

It can be seen that $L_p^p(G_3) = L_p(G_1)$.

However, one is not always so lucky to find an equivalent grammar for which the "boundary condition" technique applies. This can be illustrated through the following example adapted from Mercer and Rosenfeld [11].

Example 4.8 Let $G_4 = (\{S\}, \{a\}, \{(1) \begin{array}{ll}\#&\\S&\#\end{array} \rightarrow \begin{array}{ll}S&\\a&S\end{array}, (2) S \rightarrow a\},. S, \#)$. It can be seen that $L_p(G_4)$ is the set of all isosceles right triangles composed of a's in a field of #'s. Clearly $L_p^p(G_4) \neq L_p(G_4)$. In fact $L_p^p(G_4) = \{a\}$.

Notice that even the boundary condition is taken care of by modifying rule (2) to $\begin{matrix}\# & \\ S & \#\end{matrix} \leftarrow \begin{matrix}\# & \\ a & \#\end{matrix}$, it is still not satisfactory.

<u>Example 4.9</u> Let $G_5 = (\{S\}, \{a\}, \{ (1) \begin{matrix}\# & \\ S & \#\end{matrix} \leftarrow \begin{matrix}S & \\ a & S\end{matrix}, \ (2) \begin{matrix}\# & \\ S & \#\end{matrix} \leftarrow \begin{matrix}\# & \\ a & \#\end{matrix} \}, S, \#)$.

It can be seen that $L_p^p(G_5) \neq L_p(G_4)$. In fact, all squares with the north east corner cut also belong to the language. For instance, see <u>Figure 4.4</u>:

$$\begin{array}{l}aaa\\aaaa\\aaaa\\aaaa\end{array} \overset{(2)}{\Longrightarrow} \begin{array}{l}aaS\\aaaS\\aaaa\\aaaa\end{array} \overset{(1)}{\Longrightarrow} \begin{array}{l}aa\\aaS\\aaaa\\aaaa\end{array} \overset{(2)}{\Longrightarrow} \begin{array}{l}aS\\aaS\\aaaS\\aaaa\end{array} \overset{(1)}{\Longrightarrow} \begin{array}{l}a\\aS\\aaS\\aaaa\end{array} \overset{(2)}{\Longrightarrow} \begin{array}{l}S\\aS\\aaS\\aaaS\end{array} \overset{(1)^3}{\Longrightarrow} S \in L_p^p(G_5).$$

<u>Figure 4.4</u>

It is the author's conjecture that there is no equivalent CFAG G such that $L_p(G)=L_p(G_4)$ and the "boundary condition" technique can be applied to G to obtain a parsing grammar which captures $L_p(G_4)$.

This drawback motivates us to try another model, namely "automaton".

Let's define a parallel array automaton to be a 6-tuple $M=(Q,\Sigma,\Gamma,\delta,q_o,F)$ where Q is the set of states, Σ is the set of input symbols, Γ is the set of transition symbols, $q_o \in Q$ is the starting state, $F \subseteq Q$ is the set of final states, and δ is the state transition function in the form $\delta(q,\beta) = \{ (q',\alpha_1),(q'',\alpha_2),\ldots \}$. The interpretation of δ is in the usual sense. Let $\mathcal{A}$ and $\mathcal{B}$ be two connected arrays over $\Sigma \cup \Gamma$, then $(q,\mathcal{A}) \vdash (q',\mathcal{B})$ if there exists $(q',\alpha) \in \delta(q,\beta)$ and β is applicable to $\mathcal{A}$ (in parallel) and $\mathcal{B}$ is obtained by replacing all β in $\mathcal{A}$ (in parallel). Let $\overset{*}{\vdash}$ to be the transitive and reflexive closure of $\vdash$. Then the language accepted by M can be defined as follows: $L^p(M)=\{\mathcal{A} \mid (q_o,\mathcal{A}) \overset{*}{\vdash} (q_f,A_S), q_f \in F,$ $\mathcal{A}$ is a connected terminal array and A_S is an initial array defined in section $2\}$.

Essentially, a parallel array automaton (PAA) can be considered as a parallel automaton which works as a parsing array grammar with internal state. This idea works for a rather large class of PCFAL's including those which are not applicable for the "boundary condition" technique, such as the one in <u>Example 4.9</u>. This can be illustrated in the following example.

<u>Example 4.10</u> Let $M_4 = (\{q_o,q_1\}, \{a\}, \{S\}, \delta, q_o, \{q_1\})$ where δ is defined as

$\delta(q_o, \genfrac{}{}{0pt}{}{\#}{a\ \#}) = \left\{(q_1, \genfrac{}{}{0pt}{}{\#}{S\ \#})\right\}$, $\delta(q_1, \genfrac{}{}{0pt}{}{S}{a\ S}) = \left\{(q_1, \genfrac{}{}{0pt}{}{\#}{S\ \#})\right\}$. It can be easily verified that $L^p(M_4) = L_p(G_4)$.

While the model of PAA seems appropriate for PCFAL recognizers, it does have two drawbacks: (1)Some PAA is too powerful; it can recognize a non-PCFAL such as $\left\{\#^\infty a^n\ b^n\ \#^\infty \mid n \geq 1\right\}$, and (2)There exists a PCFAL which can not be recognized by any PAA. This can be described through the following examples.

Example 4.11 Let $\bar{M} = (\{q_o, q_1\}, \{a,b\}, \{A,B,S\}, \delta, q_o, \{q_1\})$ where δ is defined as follows:
$\delta(q_o, \#a) = \{(q_1, \#A)\}$, $\delta(q_1, b\#) = \{(q_o, B\#)\}$,
$\delta(q_o, Aa) = \{(q_1, \#A)\}$, $\delta(q_1, bB) = \{(q_o, B\#)\}$,
$\delta(q_o, AB) = \{(q_1, S\#)\}$

It can be seen that $L^p(\bar{M}) = \left\{\#^\infty a^n\ b^n\ \#^\omega \mid n \geq 1\right\}$, which is not a PCFAL. Notice that even the range of δ is restricted to exactly one nonterminal in a field of #'s (which corresponds to the α of a PCFAG rule $\alpha \to \beta$), $\bar{M}$ is still too powerful.

Example 4.12 Let $G_6 = (\{S,A,B,C,D\}, \{a\}, P, S, \#)$ be a CFAG, where
$P = \left\{S\# \to aS,\ \genfrac{}{}{0pt}{}{\#}{S} \to \genfrac{}{}{0pt}{}{A}{a},\ \genfrac{}{}{0pt}{}{\#}{A} \to \genfrac{}{}{0pt}{}{A}{a},\ A\# \to aB,\ B\# \to aB,\ \genfrac{}{}{0pt}{}{B}{\#} \to \genfrac{}{}{0pt}{}{a}{C},\ \genfrac{}{}{0pt}{}{C}{\#} \to \genfrac{}{}{0pt}{}{a}{C},\ C\# \to aS,\ \#C \to Da,\right.$
$\left.\#D \to Da,\ \genfrac{}{}{0pt}{}{\#}{D} \to \genfrac{}{}{0pt}{}{A}{a},\ S \to a\right\}$.

It can bee seen that $L_p(G_6)$ is the set of all "rugged saw-tooth"-shaped patterns, some of them have closed perimeters. For instance, the following two members are in $L_p(G_6)$: (Figure 4.5)

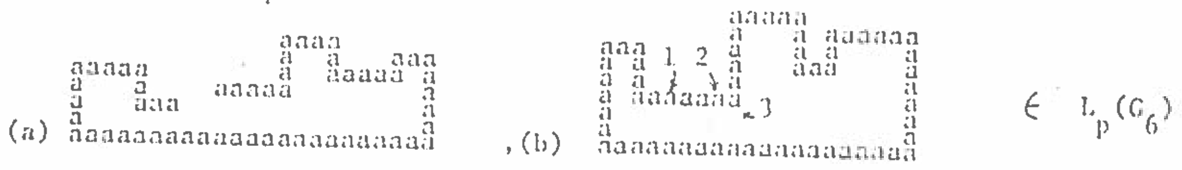

Figure 4.5 Some members of $L_p(G_6)$

To construct a PAA to recognize $L_p(G_6)$ will fail because it will always miss those which have closed perimiters. Take Figure 4.5 (b) for instance. To form a closed perimeter, the rule $S \to a$ should end up at location 1, 2, or 3 as indicated in Figure 4,5 (b). Considering these boundary conditions, a would be PAA should include the following δ transitions: (1) $\delta(q,aaa)=\{(q',Saa)\}$, (2) $\delta(q,a\overset{a}{a})=\{(q',S\overset{a}{a})\}$, and (3) $\delta(q,\ a\ \overset{a}{a})=\{(q',\ a\ \overset{a}{S})\}$. However, applying any of these transitions to a closed perimeter <u>in parallel</u> will unavoidably divide it into several separate regions. For instance, if (3) is applied (in parallel) and then $\delta(q',aS)=\{(q',\ S\#)\}$ is applied (in parallel) to <u>Figure 4.5</u>(b) we will obtain the following: (<u>Figure 4.6</u>)

```
        aaaaa
        a   a aaaaaaa
aaa     a   a a      a
a a     a   aS       a
a a     a            a
a aaaaaS             a
a                    a
a                    a
aaaaaaaaaaaaaaaaaaS  a
```

<u>Figure 4.6</u>

This means that the parsing will never reach A_S and therefore $L_p(G_6)$ can not be recognized by any PAA. Notice that here we even relax the restriction of the range α of δ to be exactly one nonterminal in a field of #'s.

Interesting enough, a very very similar (almost identical) pattern of $L_p(G_6)$ is readily recognized by a PAA.

<u>Example 4.13</u> Let G_6' be a CFAG identical to G_6 except that the rule $S \rightarrow a$ is replaced by $S \rightarrow b$ and $V_T=\{a,b\}$ instead of $\{a\}$. It is easy to see that $L_p(G_6')$ is almost identical to $L_p(G_6)$ except that the terminating symbol is "b" rather than "a". However, because of this unique, outstanding "b", one can easily construct a PAA M_6' to recognize $L_p(G_6')$ as follows.

<u>Example 4.14</u> Let $M_6'=(\{q_o\},\{a,b\},\{S,A,B,C,D\},\delta,q_o,\{q_o\})$ where

$$\delta(q_o,b)=\{(q_o,S)\},\quad \delta(q_o,\overset{A}{a})=\{(q_o,\overset{\#}{D})\},$$

$$\delta(q_o,Da)=\{(q_o,\#C),(q_o,\#D)\},\qquad \delta(q_o,\overset{a}{C})=\{(q_o,\overset{B}{\#}),\ (q_o,\overset{C}{\#})\},$$

$$\delta(q_o,\overset{A}{a})=\{(q_o,\overset{\#}{A}),\ (q_o,\overset{\#}{S})\},\qquad \delta(q_o,aB)=\{(q_o\ B\#),\ (q_o,A\#)\},$$

$$\delta(q_o,aS)=\{(q_o,\ C\#),\ (q_o,\ S\#)\}.$$

It can be seen that $L^P(M_6')=L_p(G_6')$.

This demonstrates that in dealing with 2-dimensional isometric patterns, the nature of parallel parsing is much more complicated and less well behaved than that of parallel generation, which in turn is more complicated and less well behaved than that of its 1-dimensional counterparts. Also, both parallel parsing and generation are more complicated than sequential ones. In fact, it can be seen that for every CFAG G, $L(G) = L^{p}(G)$. Furthermore, the hierarchical structure of a parallel isometric array language not only depends on its pattern (shape) but also depends on its vocabulary (content). So is the complexity of parallel parsing.

5. Discussions and Conclusions

We have just established a hierarchy for isometric array languages by showing that $\mathcal{L}$(PCFAL) and $\mathcal{L}$(CFAL) are incomparable and that $\mathcal{L}$(PCFAL) $\cap$ $\mathcal{L}$(CFAL) $=$ $\mathcal{L}$(DRAL). This result is very similar to that of 1-dimensional string languages as shown by Siromoney and Krithivasan [20] and Skyum [21]. However, the behavior of the CFAG introduced in Example 3.1 contrasts that of 1-dimensional context-free languages. In fact, for every context-free string grammar G, $L(G) \cap L_p(G) = \emptyset$ is true if and only if both $L_p(G)$ and $L(G)$ are empty. Furthermore, Lemma 3.6 ($\mathcal{L}$(ERAL) $\overset{?}{\neq}$ $\mathcal{L}$(RAL)) and Lemma 3.7 ($\left\{\begin{matrix} \# & \# & a & \# & \# \\ \# & \# & a & \# & \# \\ a & a & a & a & a \\ \# & \# & a & \# & \# \\ \# & \# & a & \# & \# \end{matrix}\right\} \notin \mathcal{L}$(ERAL)) contradict the basic aspect of 1-dimensional languages, where grammars of the form of ERAG are equivalent to the grammars of the form of RAG and the class of regular languages contains all finite languages. This is interesting since it coincides with the claim made by Rosenfeld [16] that 2-dimensional languages are more complicated and less well behaved than their 1-dimensional counterparts.

Some parsing techniques are also investigated, and it is shown that while a sequential parsing grammar for a CFAL L(G) can be easily obtained by reversing all the arrows of G, a parallel parsing grammar usually is simpler to write and takes less time than sequential ones. However, the nature of parallel parsing is very complicated. Some examples of non-CF PCFAL and non-PCF CFAL are demonstrated. However, it is an open question as for whether the language of isoceles right triangles (shown in Example 4.8) is a CFAL. The author's conjecture is that it is not, though.

For the future research, the following problems and research areas are among those the author would like to explore or see explored:

(1) To find a more appropriate model for recognizing the class of PCFAL's. Although the "parsing grammar" and "automaton" discussed in section 4 fail to serve as such a model, they do reveal that parallel recognizing algorithms have better time complexities than sequential ones. For practical considerations, a "deterministic" model of parallel recognizers is more desirable.

(2) To find an algorithm for "grammatical inference ". In section 4, the idea of obtaining a parsing grammar from an array grammar G is based upon the assumption that the grammar G is already known. We would very much like to know how a generating grammar G could be obtained algorithmically from the given array language L(G) or L_p(G). Similar work has been done for tree grammars in[6,7].

(3) To find a more general "pumping lemma". The two "small pumping lemmas" introduced in Section 3, while helpful for some purpose such as the proof of the hierarchy in Theorem 3.4, are somewhat too specific. It would be interesting to know if there is a "pumping lemma" which characterizes the necessary and sufficient conditions for CFALs (or PCFAL's). If such a pumping lemma exists, and once it is found, it will help solve a lot of problems such as to prove or disprove the author's conjecture mentioned in the second paragraph of this section.

(4) To find more fundamental properties of array languages such as closure properties. A deeper and more thorough understanding of these fundamental properties for array languages, while still largely unknown, are quite desirable. For instance, if a certain type of array languages is closed under homomorphisms, the study of a seemingly rather complicated array pattern can largely be sim-

plified if it is a homomorphic image of a comparatively simple ones.

(5) To find the practical roles of PCFAL's and CFAL's. Since one-dimensional context-free languages play a very important role in high-level programming language design and compiler construction, we would very much like to know what role PCFAL's and CFAL's play in practical world such as picture processing , computer graphics and two-dimensional pattern recognition.

(6) To explore further the nature of parallel complexities. This may include an expansion of parallel definition. For instance, the definition of parallel application introduced in section 2 is that all instances of a rule's antecedent are simultaneously replaced by the consequent. What if there are more than one rule applicable in a sentential form and all instances of these rules' antecedents are replaced by the consequents? This may even speed up further and parallel patterns under this definition may result in different hierarchical structures.

(7) To compare isometric array grammars with other models as surveyed in [12]and [23]. Investigate their advantages and disadvantages. Hopefully this will help stimulate an even more appropriate and ideal model for 2-dimensional pattern processing.

(8) To investigate solvability problems. It is well known that the emptiness problem of finite state array automata (FSAA) is unsolvable [13]. Similar work also has been done for 1-dimensional languages [8,9,19]. How about PCFAL's and CFAL's? Is it decidable whether two PCFAG's(CFAG's) are equivalent? Is it decidable whether a CFAG(PCFAG) is empty? infinite? or ambiguous? Notice that "ambiguity" here can be defined similarly as in 1-dimensional languages, i.e. a CFAG G is ambiguous if for some $\mathfrak{A} \in L(G)$(or $L_p(G)$ respectively) there are two distinct derivations.

Bibliography

[1] M. Blum and C. Hewitt, "Automata on 2-dimensional tape", Proc. of 8th IEEE Symp. 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, vol. 8 (1978), pp 144-152.

[3] M.F. Dacey, "The syntax of a triangle and some other figures", Pattern Recognition, 2 (1970), pp 11-31.

[4] M.F. Dacey, "POLY: A two dimensional language for a class of polygons", Pattern Recognition, 3 (1971), pp 197-208.

[5] M.J. Fischer, "Two characterizations of the context-sensitive languages", Proc. of 10th IEEE Symp. on Switching and Automata Theory, (1969), pp 149-156.

[6] K.S. Fu, Syntactic Methods in Pattern Recognition, Academic Press, New York (1974).

[7] K.S. Fu and T.L. Booth, "Grammatical inference: introduction and survey - Part I and II", IEEE Transactions on Systems, Man, and Cybernetics, vol SML-5 No. I and X (1975), pp 95-111, 409-423.

[8] M.A. Harrison, Introduction to Formal Language Theory, Addison-Wesley, Reading, Mass. (1978).

[9] J.E. Hopcroft and J.D. Ullman, Introduction to Automata Theory, Languages and Computation, Addison-Wesley, Reading, Mass. (1979).

[10] R.A. Kirsch, "Computer interpretation of English text and picture patterns", IEEE Trans. EC-13, (1964), pp 363-376.

[11] A. Mercer and A. Rosenfeld, "An array grammar programming system", C ACM, 16, (1973), pp 299-305.

[12] W.F. Miller and A.C. Shaw, "Linguistic methods in picture processing - a survey", Proc. AFIPS FJCC, 33, Pt 1, AFIP Press, Montvale, N.J., (1968), pp 279-290.

[13] J. Mylopoulos, "On the application of formal language and automata theory to pattern recognition," Pattern Recognition, 4, (1972), pp 37-51.

[14] A. Rosenfeld, "Isotonic grammars, parallel grammars, and picture grammars", Machine Intelligence VI, American Elsevier, New York, (1971), pp 281-294.

[15] A. Rosenfeld, "Array grammar normal forms", Information and Control, 23, (1973), pp 173-182.

[16] A. Rosenfeld, "Some notes on finite-state picture languages", Information and Control, 31, (1976), pp 177-184.

[17] A. Rosenfeld, Picture Languages, Academic Press, New York, (1979).

[18] A. Rosenfeld and J.L. Pfaltz, "Sequential operations in digital picture processing", J ACM, 13, (1966), pp 471-494.

[19] A. Salomaa, Formal languages, Academic Press, New York, (1973).

[20] R. Siromoney and K. Krithivasan, "Parallel context-free languages", Information and Control, 24, (1974), pp 155-162.

[21] S. Skyum, "Parallel context-free languages", Information and Control, 26, (1974), pp 280-285.

[22] A.R. Smith III, "Two-dimensional formal languages and recognition by cellular automata", 12th IEEE Symp. on Switching and Automata Theory, (1971), pp 144-152.

[23] P.S.P. Wang, "Sequential/parallel matrix array languages", Journal of Cybernetics, vol. 5.4, (1975), pp 19-36.

[24] P.S.P. Wang, "Recognition of two-dimensional patterns", Proc. of ACM '77, Seattle, WA (1977), pp 484-489.