A new system called sequential/parallel matrix grammars for two-dimensional pattern processing is introduced and studied. Miscellaneous language operations such as union, catenation (row and column), Kleene's closure (row and column) and substitutions are investigated. The equivalence of sequential/parallel matrix languages and finite-turn repetitive checking automata is established. Hierarchies for both languages and machines are also found. We also give regular-like expressions and array grammars that describe sets of matrices or rectangular arrays with fixed proportions. Finally, several related future research topics are mentioned.

The primary advantage of our model over others is that it provides a compromise between purely sequential methods, which take too much time for large arrays, and purely parallel methods, which usually take too much hardware for large arrays.
On Sequential/Parallel Matrix Array Languages

by

Patrick Shen-Pei Wang

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Completed May 2, 1978

Commencement June 1978
APPROVED:

Redacted for Privacy
Professor of Department of Computer Science
in charge of major

Redacted for Privacy
Head of Department of Computer Science

Redacted for Privacy
Dean of Graduate School

Date thesis is presented May 2, 1978

Typed by Patrick Shen-Pei Wang for Patrick Shen-Pei Wang
To My Parents
ACKNOWLEDGEMENT

I want to show, first of all, my deepest gratitude to my parents for their love and everything they give me, without which I could never have today.

I am indebted to my advisor Dr. Curtis Cook for his invaluable inspiration, constant encouragement and constructive criticisms, without which this thesis could never be done. Appreciations are also due to Dr. Harry Goheen, Dr. Paul Cull, Dr. James Herzog, Dr. James Hall, Dr. William Bregar and Dr. Steve Hedetniemi for their kind help and assistance.

Finally, but not least, I want to thank my wife, Yuan-fang, for her patience and being considerate with the promise that I will never do this again.
# TABLE OF CONTENTS

I. Introduction ......................................................... 1

II. Historical Background ............................................. 7

III. Matrix Grammars and Hierarchy $H_1$ ......................... 8
     3.1 Definitions, Notations and Examples ....................... 8
     3.2 Hierarchy $H_1$ .................................................. 14
     3.3 Closure Properties ............................................. 14

IV. Finite-Turn Repetitive Checking Automata ................... 24
     4.1 Finite-Turn Repetitive Checking Automata ................. 24
     4.2 The Equivalence of $(3:3) \text{kTRCA}$ and $(3:3) \text{ML}$ ..... 29
     4.3 Recognition of other Sequential/Parallel Matrix Languages ...... 36

V. Array Grammars and Languages .................................. 40
     5.1 Definitions and Notations ..................................... 40
     5.2 An Example: Digitized English Letter "R" with Fixed Proportions 43

VI. Discussions and Conclusions .................................... 47

Bibliography .......................................................... 60
**LIST OF ILLUSTRATIONS**

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1.1</td>
<td>Some Members of L(G)</td>
<td>12</td>
</tr>
<tr>
<td>3.1.2</td>
<td>L(G) under homomorphisms h, H and H'</td>
<td>13</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Construction of a (3:3)kTRCA</td>
<td>27</td>
</tr>
<tr>
<td>4.2.1</td>
<td>The Proof of Theorem 4.2.2</td>
<td>35</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Construction of a (2:2)kTRCA</td>
<td>37</td>
</tr>
<tr>
<td>5.2.1</td>
<td>The first four members of L(G)</td>
<td>45</td>
</tr>
<tr>
<td>5.2.2</td>
<td>The derivation process of $M_3$</td>
<td>46</td>
</tr>
<tr>
<td>6.1</td>
<td>The set of all digitized English letter &quot;L&quot; with different proportions, which falls in RML</td>
<td>54</td>
</tr>
<tr>
<td>6.2</td>
<td>The set of all digitized English letter &quot;I&quot; with different proportions, which falls in CFML</td>
<td>55</td>
</tr>
<tr>
<td>6.3</td>
<td>A language which falls in CSML</td>
<td>56</td>
</tr>
<tr>
<td>6.4</td>
<td>The relation between iML and $(i:R)ML, i=0,1,2,3$</td>
<td>57</td>
</tr>
</tbody>
</table>
CHAPTER I

INTRODUCTION

Recently the study of 2-dimensional grammars and languages has become a more important and interesting subject because of its significant applications in the data processing of 2-dimensional patterns [34]. 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 languages is very difficult to obtain by natural extension of the grammars for 1-dimensional languages. For instance, in the string case a 1-dimensional rule, say $A \rightarrow BCD$, applied to a sentential form $\alpha A \beta$ produces $\alpha BCD \beta$. This can be interpreted as either $\alpha$ being pushed left or $\beta$ pushed right to leave room for the insertion of BCD in place of $A$. When the application of a rule is extended to the 2-d case we encounter the shearing problem. For example, if we apply the rule $A \rightarrow B\quad_{CD}$ to the sentential form $YAY$ do we obtain $XXA$ or $ZCDZ$ or $\ldots$ ?

The shearing problem as above is concerned with the relations between the symbols in the sentential form before and after application of the rule. Thus, to promote research on 2-dimensional languages we need find an appropriate general mathematical model of the grammar that resolves the shearing problem as described above. This is one of the motivations for this thesis. The other is that in 2-dimensional pattern pro-
cessing, purely sequential methods usually take too much time for large arrays and purely parallel methods require too much hardware for their constructions for large arrays. Therefore for practical purposes some "mixed" types of methods serving as a compromise between purely serial and purely parallel ones become necessary.

Several types of shearing effect-free grammars have been proposed. Rosenfeld's isotonic array grammar \([19,20]\) overcomes the shearing effect by defining the left and right sides of a rule to have the same shape. However, his generating process is either purely parallel or purely sequential. Siromoney's matrix grammar\([25\) is sequential/parallel but the generative power is quite limited. In this paper we introduce a model which is shearing effect-free and more powerful than Siromoney's matrix grammar.

Chapter II gives us some historical survey of the current research on 2-dimensional pattern processing, including various methods of 2-dimensional pattern generating grammars and picture descriptive languages. In chapter III, matrix language is defined and a Chomsky hierarchy \(H_1\) corresponding to extensions of 1-dimension is found. Miscellaneous operations such as union, row\(\text{column}\) catenation, row\(\text{column}\) Kleene's closure and homomorphisms of these languages are investigated. In chapter IV, the construction showing the equivalence of machines and matrix languages is given. Array grammars and regular-like expressions that describe matrices with fixed proportions are given in Chapter V. It is seen in an example that the generative capability of array grammar is more powerful than that of matrix grammar.

Finally, some discussions and future research topics such as
formal properties of matrix array languages and sequential/parallel complexities are given in Chapter VI.
CHAPTER II

HISTORICAL BACKGROUND

In dealing with 2-dimensional languages, array grammars and picture grammars are widely used and shown to be very useful\[5,30,18\]. There are several ways to generalize phrase-structure grammars whose rules allow the replacement of a subarray of a picture with another subarray. The relation between pictures and 2-dimensional arrays is that a 2-d array is a digitized representation of a picture. For instance, a triangle can be represented by a 2-dimensional array:

\[
\begin{array}{c}
\text{X} \\
\text{AC} \\
\text{AIC} \\
\text{YBBZ}
\end{array}
\]

The earliest example of an array grammar was described by Kirsch[12] for generating a class of labeled 45° right triangles. A similar formal system is developed by Dacey[6] who gave grammars for languages consisting of classes of polygons. The syntactic structure of these languages was analyzed and it was shown that a mathematical group summarized the structure holding between languages constructed for polygons related to proper and improper rotations. Yodokawa and Honda[34] constructed an appropriate mathematical model of 2-dimensional pattern grammars on the set of 2-dimensional words.

Some array grammar normal forms are given by Rosenfeld[20] and it is shown that several possible ways of defining an array grammar are in fact all equivalent. Isotonic grammars, parallel grammars and picture grammars are investigated and it is shown in Rosenfeld[19] that any parallel language is a sequential language and vice versa. In Milgram and Rosenfeld[4]
the equivalence relation between array grammars and automata with 2-dimensional tapes was found. The relation between array grammars and another parallel processing device, tessellation automata, i.e. a finite array of identical finite-state-machines called cells, each connected to its neighbors in a highly regular fashion, has been intensively studied by Smith [26,27,28] and by Wang and Grosky [33].

Most early work concerning with 2-dimensional pictures such as Freeman's chain code (1961) [7], Ledeley's syntax-controlled picture processing system (1965) [13], Miller and Shaw's Picture Descriptive Languages (1969) [15], Rosenfeld and Pfaltz web grammars (1969) [17] and Uhr and Gordon's method for describing disconnected graphs (1970) [29] attempted to reduce 2-dimensional pictures to strings.

Other picture descriptive methods such as Chang's 2-dimensional expressions, Narasimhan's labeling schemata, Rosenfeld and Pfaltz's digital picture processing methods and tree grammars can be found in [1,2,16,22,8,30,31,32].

In 1972, Siromoney et al. [25] proposed a grammar called a matrix grammar. Briefly speaking, their matrix grammar is an ordered two-tuple G = (G_1, G_2) where G_1 = (V_1, I_1, P_1, S) is a grammar and G_2 = \{ G_{2i} \}_{i=1}^{k} where G_{2i} = (V_{2i}, I_2, P_{2i}, S_1), i=1, ..., k are right linear grammars, V_{2i} \cap V_{2j} = \emptyset, i \neq j. A matrix is obtained by first applying G_1 horizontally, sequentially to get a string in I_1^*, then applying G_{2i} vertically, simultaneously.

The language generated by a matrix grammar is defined to be the set of all terminal matrices (over I_2) derived from horizontal and vertical rules. For example, let G = (G_1, G_2) be a matrix grammar, where
\[ G_1 = \left\{ \{ S \}, \{ S_1, S_2 \}, \{ S \rightarrow S_1S_1, S \rightarrow S_2 \} , S \right\}, \quad G_2 = \left\{ G_{21} \right\} \cup \{ G_{22} \} \]
\[ G_{22} = \left\{ \{ S_2 \}, \{ x \}, \{ S_2 \rightarrow xS_2, S_2 \rightarrow x \} , S_2 \right\}, \]
\[ G_{21} = \left\{ \{ S_1, A \}, \{ x \}, \{ S_1 \rightarrow xA, A \rightarrow A, A \rightarrow x \} , S_1 \right\}. \]

Then \( L = \left\{ S_1^n S_2 S_1^m \mid n \geq 1 \right\} \),
\[ L(G_{21}) = \left\{ x(\cdot)^m x \mid m \geq 1 \right\} \]
\[ L(G_{22}) = \left\{ x^m \mid m \geq 1 \right\}. \]
\[ L(G) = \left\{ \text{all digitized English letter "I" with variable proportions} \right\}. \]

For instance:
\[
S \Rightarrow S_1S_1 \Rightarrow^* S_1S_1S_2S_1S_1
\]

\[
\begin{array}{cccccccc}
X & X & X & X & X \\
\downarrow^* \\
X & X & X & X & X \\
\vdots & X & \vdots \\
\vdots & X & \vdots \\
X & X & X & X \\
\end{array}
\]

\( \in L(G) \)

Their method is useful in describing a wide variety of picture classes. However, the generative power of their matrix grammar is still quite limited. For example it can not generate the following patterns:

\[
\left\{ \left( \begin{array}{c} B_m \\ R_n \end{array} \right)^p \left( \begin{array}{c} R_m \\ E_n \end{array} \right)^p \mid p, m, n \geq 1 \right\},
\]

\[
\begin{array}{cc}
P & P \\
B & R \\
R & B \\
\end{array}
\]
Notice that even if the restriction of the vertical derivations to be right linear is relaxed, it still can not generate such languages. This is not because of the forms of the rules, but is due to the ways in which the rules are applied. The method introduced in CHAPTER III overcomes this disadvantage.

Milgram and Rosenfeld[14] introduced array automata which recognize array languages. They showed that grammars that rewrite arrays are equivalent to Turing machines having array "tapes", and that "monotonic" array grammars (in which arrays never shrink during derivations) are equivalent to "array-bounded machines", which are extensions of linear bounded automata.

In 1972, Selkow[24] introduced a "mixed" type of machine which is essentially a one-dimensional (e.g. horizontal) array of automata moving as a unit, in the vertical direction. Rosenfeld and Milgram[21] applied this concept to array languages and found several array automata hierarchies.
CHAPTER III

MATRIX GRAMMARS AND HIERARCHY H₁

3.1 Definitions, Notations and Examples

A matrix over an alphabet I is an mxn rectangular array of symbols from I(m,nAo). The set of all matrices over I is denoted by I** and I**={I} where I is the empty matrix.

Definition 3.1.1 Let \( X = \begin{pmatrix} a_{11} \ldots a_{1n} \\ \vdots \\ a_{m1} \ldots a_{mn} \end{pmatrix} \) and \( Y = \begin{pmatrix} b_{11} \ldots b_{1n'} \\ \vdots \\ b_{m1} \ldots b_{mn'} \end{pmatrix} \)

then the column concatenation \( \Theta \) is defined only when \( m=m' \) and is given by

\[
X \Theta Y = \begin{pmatrix} a_{11} \ldots a_{1n} b_{11} \ldots b_{1n'} \\ \vdots \\ a_{m1} \ldots a_{mn} b_{m1} \ldots b_{mn'} \end{pmatrix}
\]

and the row concatenation \( \Theta \) is defined only when \( n=n' \) and is given by

\[
X \epsilon Y = \begin{pmatrix} a_{11} \ldots a_{1n} \\ \vdots \\ a_{m1} \ldots a_{mn} \\ b_{11} \ldots b_{1n'} \\ \vdots \\ b_{m1} \ldots b_{mn'} \end{pmatrix}
\]

Definition 3.1.2 Let \( M \) and \( M' \) be two sets of matrices, the column concatenation is defined as \( M \Theta M' = \{ X \Theta Y | X \in M, Y \in M' \} \) and the row concatenation is defined as \( M \epsilon M' = \{ X \epsilon Y | X \in M, Y \in M' \} \).

Notation. \( I^* \) denotes the set of all horizontal sequences over \( I \) and \( I^+ = I^* - \{ \varepsilon \} \), where \( \varepsilon \) is the empty word of \( I^* \). \( I_* \) is the set of all vertical sequences of symbols over \( I \) and \( I_+ = I_* - \{ \varepsilon \} \), where \( \varepsilon \) is the empty word of \( I_* \). \( x \in I^+ \), \( i=1,2, \ldots \).
Kleene's closure can be defined recursively as follows:

**Definition 3.1.3** Let $M$ be a set of matrices and $M^1 = M, M^2 = M \cdot M, \ldots,$ $M^{i+1} = M^i \cdot M$, then $M^* = \bigcup_{i=1}^{\infty} \{M^i\}$, \((\text{column } +)\) and $M^* = M^{+} \cup \{\Lambda\}$\((\text{column } *)\).

Similarly, $M_1 = M, M_2 = M \cdot M, \ldots, M_{i+1} = M_i \cdot M$, then $M_{+} = \bigcup_{i=1}^{\infty} \{M_i\}$, \((\text{row } +)\), and $M_* = M_{+} \cup \{\Lambda\}$ \((\text{row } *)\).

**Definition 3.1.4** Let $X = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}$ then the transpose of $X$, denoted by $\mathcal{T}(X)$ is defined as $\mathcal{T}(X) = \begin{bmatrix} a_{11} & \cdots & a_{m1} \\ \vdots & \ddots & \vdots \\ a_{1n} & \cdots & a_{mn} \end{bmatrix}$.

the quarter turn of $X$\((\text{in the clockwise direction})\) is defined as $$Q(X) = \begin{bmatrix} a_{m1} & \cdots & a_{11} \\ \vdots & \ddots & \vdots \\ a_{mn} & \cdots & a_{mn} \end{bmatrix},$$

the reflection about the right-most vertical axis is defined as $$\overline{X} = \begin{bmatrix} a_{in} & \cdots & a_{11} \\ \vdots & \ddots & \vdots \\ a_{mn} & \cdots & a_{m1} \end{bmatrix},$$

the reflection about the base is $$X^* = \begin{bmatrix} a_{m1} & \cdots & a_{mn} \\ \vdots & \ddots & \vdots \\ a_{11} & \cdots & a_{1n} \end{bmatrix},$$

and a half turn is $$X^\frac{1}{2} = \begin{bmatrix} a_{mn} & \cdots & a_{m1} \\ \vdots & \ddots & \vdots \\ a_{in} & \cdots & a_{11} \end{bmatrix},$$

If $M$ is a set of matrices from $I^{++}$ then $0(M) = \{0(X) \mid X \in M\}$, $0 = \mathcal{T}, Q, \overline{\cdot}, \sim$ or $\cong$.

**Definition 3.1.5** A mapping $H: I^{++} \rightarrow I^{++}$ is called a homomorphism if $H(X \cdot Y) = H(X) \cdot H(Y)$ and $H(X \circ Y) = H(X) \circ H(Y)$. It is easily seen that a homomor-
phism is defined only when \( H(a) = r \times s \) array of terminals from \( I' \), \( r, s \) the same for all \( a \in I \). If \( M \) is a set of matrices then \( H(M) = \{ H(X) | X \in M \} \).

Definition 3.1.6 A matrix grammar (MG) is a triple \( G = (G_1, G_2, M) \), where

\[ G_1 = (V_1, I_1, P_1, S) \] is a phrase structure grammar (PSG or type 0), context-sensitive grammar (CSG or type 1), context-free grammar (CFG or type 2), regular grammar (RG or type 3), where

- \( V_1 \) is a finite nonempty set of horizontal nonterminals,
- \( I_1 \) is a finite nonempty set of intermediates, \( I_1 = \{ S_1, \ldots, S_k \} \),
- \( P_1 \) is a finite nonempty set of type 0(1,2,3) production rules called horizontal rules,
- \( S \in V_1 \) is the start symbol, and

\[ G_2 = \bigcup_{i=1}^{k} G_{2i} \] where \( G_{2i} = (V_{2i}, I_{2i}, P_{2i}, S_i) \) are \( k \) type \( 0,2,3 \) grammars with \( I_{2i} \) a finite nonempty set of terminals, \( V_{2i} \) a finite set of vertical nonterminals, \( S_i \) the start symbol, and \( P_{2i} \) a finite nonempty set of production rules, \( V_{2i} \cap V_{2j} = \emptyset \), for \( i \neq j \).

\[ M = \{ n_i | i = 1, \ldots, k \} \] where each \( n_i \) is a \( k \times 1 \) matrix consisting of \( k \) rules from the \( k \) \( G_{2i} \)'s. In each matrix, the length of the both right hand and left hand sides of rules are identical.

The derivations are obtained by first applying the horizontal rules of \( G_1 \) until every symbol becomes an intermediate and then applying the vertical productions of a matrix in \( M \). In the vertical derivation, whenever a matrix is applied, all of the applicable rules in the matrix should be applied simultaneously. If no such a matrix is found, the derivation halts.

Definition 3.1.7 An MG \( G = (G_1, G_2, M) \) is called a type \((i:j)\)MG if \( G_1 \) is a type \( i \) and \( G_2 \) is a type \( j \) grammar for \( 0 \leq i, j \leq 3 \). We use also
(CF:CS) for type(1:2)MG and so on. Let $\alpha$ and $\beta$ be in $(V_1, U_1)^*$. By $\alpha \rightarrow \beta$ we mean after $P_1$ is applied $\rightarrow$ we get $\beta$. Let $A, B$ in $(V_2, U_2)^{++}$. By $A \downarrow B$ we mean after a column matrix is applied to $A$ we get $B$. Let $\Rightarrow$ and $\Rightarrow^*$ be the transitive closure of $\Rightarrow$ and $\Rightarrow^*$ respectively. The language generated by $G$ is defined to be

$$L(G) = \{(a_{ij})_{i=1}^{m} \in \mathbb{Z}, j=1, \ldots, n \mid S \xrightarrow{\psi}^* G_1(a_{ij}), \psi \in I_1^*, a_{ij} \in I_2 \}$$

Notice that under our definition, a matrix grammar can only generate rectangular arrays.

**Notation.** If $L=L(G_1), L_1=L(G_2), i=1, \ldots, k$ we write

$$L(C)=\langle L \rangle : : (\psi L_1, \ldots, \psi L_k)$$

where $\psi$ means the transpose of a string (defined in Definition 3.1.4).

**Example 3.1.1** Let $G=(G_1, G_2, M)$ be a (CF:CF)MG, where $G_1=$ \{\{S, S_o\}, \{S_1, S_2, S_3, S_4, S_5\}, P, S\}, $P=$ \{\{S \rightarrow S_1 S_2 S_5, S_1 \rightarrow S_2 S_3 S_4, S_2 \rightarrow S_3 \}

$G_2 = G_{21} U G_{22} U G_{23} U G_{24} U G_{25}$ where

$G_{21} = (\{S_1\}, I_2, P_{21}, S_1)$

$G_{22} = (\{S_2, S_2o\}, I_2, P_{22}, S_2)$

$G_{23} = (\{S_3, S_3o\}, I_2, P_{23}, S_3)$

$G_{24} = (\{S_4, S_4o\}, I_2, P_{24}, S_4)$

$G_{25} = (\{S_5, S_5o\}, I_2, P_{25}, S_5)$

$M = \{m_1, m_2, m_3\}$ and $P_{21}$'s are listed below:

(Please see next page.)

$: By this we mean a rule of $P_1$ is applied.
P21 = \{ (S_1 \rightarrow XS_1 X), (S_1 \rightarrow XS_1 X), (S_1 \rightarrow X) \}

P22 = \{ (S_2 \rightarrow XS_2 X), (S_2 \rightarrow XS_2 X), (S_2 \rightarrow X) \}

P23 = \{ (S_3 \rightarrow XS_3 X), (S_3 \rightarrow XS_3 X), (S_3 \rightarrow X) \}

P24 = \{ (S_4 \rightarrow XS_4 X), (S_4 \rightarrow XS_4 X), (S_4 \rightarrow X) \}

P25 = \{ (S_5 \rightarrow XS_5 X), (S_5 \rightarrow XS_5 X), (S_5 \rightarrow X) \}

It can be seen that \( L(G) \) is the set of all digitized "R" with all sizes and all proportions. Some members of \( L(G) \) are shown in Fig 3.1.1.

Example 3.1.2 Let's define homomorphisms \( h, H \) and \( H' \) in Example 3.1.1 such that

\[
\begin{align*}
h(\cdot) &= \cdot & H(\cdot) &= \cdot & H'(\cdot) &= \cdot \\
h(x) &= \times & H(x) &= \times & H'(x) &= \times
\end{align*}
\]

Then \( L, h(L), H(L) \) and \( H'(L) \) are given in Figure 3.1.2. They denote respectively the original pattern, its elongation along y direction, its topological transformation and its magnification.

Definition 3.1.3 A substitution \( f \) is a mapping of a finite set \( \Sigma \) onto subsets of \( \Delta^{**} \) for some finite set \( \Delta \).

This definition could be extended to matrices as follows:

\[
f(M) = \{ M' \mid M' \text{ is a rectangular matrix made by replacing each symbol}\}
\]
Figure 3.1.2 A member of \( L(G) \) under \( h, H \) and \( H' \).

(a) original pattern
(b) elongation
(c) topological transformation and (d) magnification.
of $M$ by $f(a)$.  

Notice that, $f$ is defined only when $f(a) = r \times s$ array of symbols, $r$ and $s$ are the same for all symbols in each substitution. Notice also that homomorphism defined in Section 3.1 is a special case of substitution. Also, a substitution $f$ (homomorphism $h$) on $\Sigma$ is said to be $\Lambda$-free if for each $a$ in $\Sigma$, the range of $f$ (and $h$) does not contain $\Lambda$.

Now, using the technique employed in the first half of this section, together with Theorems 9.7, 9.9 and Corollary 9.3 of [11] we can see that

Theorem 3.1.1 The class of $(i:j)ML$ is closed under substitution and homomorphism, $i=0,2,3$; $j=0,2,3$.

Theorem 3.1.2 The class of $(1:1)ML$, $(i,1)ML$ is closed under $\epsilon$-free substitution and $\epsilon$-free homomorphism, $i=0,1,2,3$.

3.2 Hierarchy $H_1$

Since type $i$ language properly contains type $j$ language, $0 \leq i \leq j \leq 3$, Theorem 3.2.1 There exists a hierarchy $H_1$ of matrix languages:

$$
H_1: RL \subset (R:R)ML \subset (R:CF)ML \subset (R:CS)ML \subset (R:PS)ML \\
CFL \subset (CF:R)ML \subset (CF:CF)ML \subset (CF:CS)ML \subset (CF:PS)ML \\
CSL \subset (CS:R)ML \subset (CS:CF)ML \subset (CS:CS)ML \subset (CS:PS)ML \\
PSL \subset (PS:R)ML \subset (PS:CF)ML \subset (PS:CS)ML \subset (PS:PS)ML
$$

Proof: Obvious. Q.E.D.

3.3 Closure Properties

Now we want to investigate closure properties of $ML$ under $U, \Theta, \Theta, \text{row}$ (column)Kleene's closure and several simple operations such as $\cap$, $\cup$ and $Q$.

Proposition 3.3.1 If a class of arrays $M$ is described by an $(L):\cap L_1$,
, ..., \tau L_k), the class of its

(1) \tilde{N} is given by (L)::(\tau L_1, ..., \tau L_k),
(2) \tilde{M} is given by (L)::(\tau L_1, ..., \tau L_k),
(3) \tilde{T} is given by (L)::(\tau L_1, ..., \tau L_k),
(4) \sim (\tilde{N}) is given by (L)::(L_1, ..., L_k) and
(5) Q(\tilde{N}) is given by (L)::(L_1, ..., L_k).

Proof: The proof of (1) is given below, all others are similarly got, hence omitted.

It is well known that if \(G_1\) is a type \(i\) grammar then we can find a

\(G_1'\) such that \(L(G_1') = L(G_1)^R\), which in our notation is

\(\tilde{L}(G_1)\).

Q.E.D.

As in the string case, we can define a regular array in
the 2-d case as the smallest class of arrays containing all finite
matrices and closed under union, catenation (row and column) and Kleene's
closure (row and column). Firstly we see that

Theorem 3.3.1 The class of (i:j)\((L)::(\tau L_1, ..., \tau L_k)\), i.e. \((i:j)_X\), is
closed under union and contains all finite matrices, \(0 \leq i, j \leq 3\).

Proof: The proof is very similar to Theorems 9.1 and 3.7 of [11]. Q.E.D.

Theorem 3.3.2 The class of (i:j)_M is closed under column catenation and
column Kleene's closure, \(0 \leq i, j \leq 3\).

Proof: Let \(G = (G_1, G_2, M)\) and \(G' = (G_1', G_2', M')\) be two \((i:j)_X\). According to
Theorems 3.8, 3.9 and 9.1 of [11], type \(i, i = 0, 1, 2, 3\) string languages are
closed under product and Kleene's closure. We can find a type \(i \in \mathbb{G}_1^*\) such
that \(L(G^*) = L(G_1)L(G')\) and a \(\tilde{G}_1\) such that \(L(\tilde{G}) = (L(G_1))^*\). This takes care
of horizontal strings. The vertical grammars can be obtained by combining
G_2 and G'_2. M^* can be easily obtained by combining M and M'. Therefore we get a (i:j)G^*=(G_1^*,G_2^*,M^*) such that L(G^*)= L(G)\#L(G') and a (i:j)G such that L(G) = (L(G_1))^*.

Q.E.D.

So far the matrix languages are mainly defined by first applying sequential horizontal rules from G_1 and then applying vertical rules from G_2. Notationally, given a matrix grammar G = (G_1,G_2,M), the language generated by G is L(G) = (L)::(\tau L_1,\ldots, \tau L_k) where L = L(G_1), L_1 = L(G_{21}) i=1,\ldots,k and G_2 = \bigcup_{i=1}^k G_{2i}, \tau \text{ is the transpose.}

We are now going to explore the possibilities that allow first using G_1 vertically, sequentially, then applying G_2 horizontally. Notationally, such language can be represented as L_\tau(G) = (\tau L)::(L_1,\ldots,L_k).

We see that

**Theorem 3.3.3** The class of (i:i)L_\tau(G) is equivalent to the class of (i:i) L(G), i=0,1,2,3.

**Proof:** We'll show for the case (3:3)ML. Other cases are proved similarly.

Let G = (G_1,G_1,M) be an (3:3)MG, where G_1=(V_1,I_1,P_1,S), where

P_1=\{a \vDash a_i r_i \mid a_i \in I_1, l_i r_i \text{ are nonterminals with } r_i \text{ possibly } \epsilon\},

V_1=\{v_1, \ldots, v_h\},

I_1=\{s_1, \ldots, s_k\} \text{ and }

G_2 = \bigcup_{i=1}^k G_{2i}, \text{ where}

G_{21} = (V_{21}, I_{21}, P_{21}, S_1), \bigcup_{i=1}^k I_{2i} = I_2

P_{21}=\{a_i r_{i j} \vDash a_{i j} r_{i j} \mid 1 \leq i \leq k, 1 \leq j \leq p, p \text{ is the number of matrices in } M, l_{i j} r_{i j} s_t \in V_{21} \text{ with } r_{i j} \text{ possibly empty, } a_{i j} \in I_{21}\},

M=\{m_1, \ldots, m_p\}.

We can construct another (3:3)MG G' = (G'_1,G'_2,M') where

V'_1 = \bigcup_{i=1}^k V_{2i}, I'_1 = \bigcup_{i=1}^k I_{2i}.
$S'_0 = (s_1, \ldots, s_k)$ and

$P'_1 = \left\{ \left( l_{1j}, \ldots, l_{kj} \right) \rightarrow (a_{1j}, \ldots, a_{kj}) (r_{1j}, \ldots, r_{kj}) \right\} \cup \left\{ \left( l_{1q}, \ldots, l_{kp} \right) \rightarrow (a_{1q}, \ldots, a_{kp}) \right\}$

where $m_j$ is not a terminal matrix and $m_q$ is a terminal matrix for $1 \leq j, q \leq p$.

$G'_{2i} = (V'_{2i}, I'_{2i}, P'_{2i}, S'_i)$ for $i = 1, \ldots, k'$ where $k' = I'_1$.

$M'$ can be constructed as follows: $M' = \{ m'_1, \ldots, m'_t \}$

If the $i$-th rule of $P_1$ is $v_f \rightarrow s_q v_g$, i.e.

$\begin{align*}
    1_i &= v_f \\
    a_i &= s_q \\
    r_i &= v_g
\end{align*}$

then we have $m'_i$ as follows:

$\begin{align*}
    m'_i = \left\{ \left( a_{11}, \ldots, a_{kp} \right) \rightarrow a_{qp} (a_{1p}, \ldots, a_{kp}) \right\}
\end{align*}$

and if the $i$-th rule of $P_1$ is $v_f \rightarrow s_q$, i.e. a terminal rule, we have

$\begin{align*}
    m'_i = \left\{ \left( a_{11}, \ldots, a_{kp} \right) \rightarrow a_{qp} (a_{1p}, \ldots, a_{kp}) \right\}
\end{align*}$

$V'_{2i}, I'_{2i}, P'_{2i}, S'_i$ can be obtained through the above construction.

For instance, $V'_{2i}$ is the union of $k$-tuples of the $i$-th rules of all $m_1$ $I'_{2i}$ is the union of singletons of the $i$-th rules of all $m_1$ and $P'_2$ is obtained by taking the union of the $i$-th rules of all matrices. These constructions are illustrated in Example 3.3.2.

It can be seen that $G'$ is also $\text{a}(3;3) \text{MG}$ and that the class of $L(G')$ is equivalent to the class of $L(G'$), i.e.
An algorithm of constructing $G$ given $G'$ such that $L(G) = L(G')$ can be similarly obtained, hence omitted.

An example is illustrated in Example 3.3.2.

Example 3.3.2 Let $G = (G_1, G_2, M)$ be an $(R;R)MG$ where

$G_1 = (\{s, s_1, s_2\}, \{s_1, s_2^1\}, P_1, s)$, where

$P_1 = \{S \rightarrow s_1, S \rightarrow s_2, S \rightarrow s^1, S \rightarrow s^2, S \rightarrow s^2_1, S \rightarrow s^2_2\}$, and

$G_2 = \{G_{21}, G_{22}\}$ where

$G_{21} = (\{s_1, s_1^1, s_2\}, \{a, b\}, P_{21}, s_1)$, with

$P_{21} = \{(1) s_1 \rightarrow a s_1^1, (2) s_1 \rightarrow b s_1^1, (3) s_1 \rightarrow b s_2, (4) s_1 \rightarrow b s_2^1, (5) s_2 \rightarrow b\}$, and

$G_{22} = (\{s_2, s_2^1, s_2^2\}, \{c, d\}, P_{22}, s_2)$ with

$P_{22} = \{(a) s_2 \rightarrow c s_2^1, (b) s_2 \rightarrow c s_2^1, (c) s_2 \rightarrow d s_2^2, (d) s_2 \rightarrow d s_2^2, (e) s_2 \rightarrow c s_2^1, (f) s_2 \rightarrow d\}.

M = \{m_1, m_2, \ldots, m_{14}\}$ described as follows:

\[
\begin{array}{cccccccccccccc}
m_1 & m_2 & m_3 & m_4 & m_5 & m_6 & m_7 & m_8 & m_9 & m_{10} & m_{11} & m_{12} & m_{13} & m_{14} \\
(1) & (2) & (2) & (2) & (2) & (3) & (3) & (3) & (3) & (4) & (4) & (4) & (4) & (5) \\
(a) & (b) & (c) & (d) & (e) & (b) & (c) & (d) & (e) & (b) & (c) & (d) & (e) & (f) \\
\end{array}
\]

It is seen that

$L(G) = \left\{ (a_m)^+ (b_m)^+ b_m^m | m, n \geq 1 \right\}.$

For instance:
To construct $G'$, $G' = (G'_1, G'_2, M')$, where

$G'_1 = (V'_1, I'_1, P'_1, S')$, with

$V'_1 = V_{21} \times V_{22}$, where $V_{21} = \{(s_1, s_{11}, s_{12})\}$ $V_{22} = \{(s_2, s_{21}, s_{22})\}$

$I'_1 = I_{21} \times I_{22}$, where $I_{21} = \{a, b\}$ $I_{22} = \{c, d\}$, and

$P'_1 = \{(s_1, s_2) \rightarrow (a, c) (s_{11}, s_{21}),
            (s_{11}, s_{21}) \rightarrow (a, c) (s_{11}, s_{21}),
            (s_{11}, s_{21}) \rightarrow (a, d) (s_{11}, s_{22}),
            (s_{11}, s_{22}) \rightarrow (a, d) (s_{11}, s_{22}),
            (s_{11}, s_{22}) \rightarrow (a, c) (s_{11}, s_{21}),
            (s_{11}, s_{21}) \rightarrow (b, c) (s_{12}, s_{21}),
            (s_{11}, s_{21}) \rightarrow (b, d) (s_{12}, s_{22}),
            (s_{11}, s_{22}) \rightarrow (b, d) (s_{12}, s_{22}),
            (s_{11}, s_{22}) \rightarrow (b, c) (s_{12}, s_{21}),
            (s_{12}, s_{21}) \rightarrow (b, c) (s_{12}, s_{21}),
            (s_{12}, s_{21}) \rightarrow (b, d) (s_{12}, s_{22}),
            (s_{12}, s_{22}) \rightarrow (b, d) (s_{12}, s_{22}),
            (s_{12}, s_{22}) \rightarrow (b, c) (s_{12}, s_{21})\}$
\[ (s_{12}, s_{22}) \Rightarrow (b, d) \] .

\[ M' = \{ m'_1, m'_2, m'_3, m'_4, m'_5 \} \]
described as follows:

\[ m'_1 = \begin{cases} 
(a, c) \rightarrow a(a, c) \\
(a, d) \rightarrow a(a, d) \\
(b, d) \rightarrow b(b, d) \\
(b, c) \rightarrow b(b, c) 
\end{cases} \]

\[ m'_2 = \begin{cases} 
(a, c)_1 \rightarrow a(a, c)_1 \\
(a, d)_1 \rightarrow a(a, d)_1 \\
(b, d)_1 \rightarrow b(b, d)_1 \\
(b, c)_1 \rightarrow b(b, c)_1 
\end{cases} \]

\[ m'_3 = \begin{cases} 
(a, c)_2 \rightarrow c(a, c)_2 \\
(a, d)_2 \rightarrow d(a, d)_2 \\
(b, d)_2 \rightarrow d(b, d)_2 \\
(b, c)_2 \rightarrow c(b, c)_2 
\end{cases} \]

\[ m'_4 = \begin{cases} 
(a, c)_2 \rightarrow c(a, c)_2 \\
(a, d)_2 \rightarrow d(a, d)_2 \\
(b, d)_2 \rightarrow d(b, d)_2 \\
(b, c)_2 \rightarrow c(b, c)_2 
\end{cases} \]

\[ m'_5 = \begin{cases} 
(a, c)_2 \rightarrow c \\
(a, d)_2 \rightarrow d \\
(b, d)_2 \rightarrow d \\
(b, c)_2 \rightarrow c 
\end{cases} \]

It is seen that \( G' \) is an \( (R:R) \) MG and
For $w \in L(G) \implies w \in L(G')$.

For instance:

$$S \xrightarrow{*} (a,c)(a,c)(a,c)(a,c)(a,d)(b,d)(b,c)(b,d)(b,d)$$

$$\downarrow^*$$

$$\begin{array}{cccccccc}
da & a & a & a & a & b & b & b \\
a & a & a & a & a & b & b & b \\
c & c & c & c & d & d & c & d \\
c & c & c & c & d & d & c & d \\
c & c & c & c & d & d & c & d \\
\end{array}$$

$\in L(G')$

Now we are going to show the following:

**Theorem 3.3.4** The class of (1:1)ML is closed under row concatenation, for $0 \leq i \leq 3$.

Proof: We'll show the case for (3:3)ML. All other cases can be obtained similarly.

Let $G=(G_1,G_2,M)$ and $G'=(G'_1,G'_2,M')$ be two (3:3)MG. From Theorem 3.3.3 we see that we can find two (3:3)MG $\bar{G}=(\bar{G}_1,\bar{G}_2,\bar{M})$ and $\bar{G}'=(\bar{G}'_1,\bar{G}'_2,\bar{M}')$ $\implies$ $L(G)=L_\prec(\bar{G})$ and $L(G')=L_\prec(\bar{G}')$. From Theorem 3.8 of [11], we find a type 3 grammar $G''$ such that $L(G'')=L(\bar{G}_1)L(\bar{G}'_1)$. Therefore, we can find a (3:3)MG $G^*=(G^*_1,G^*_2,M^*)$ such that $L(G^*)=L_\prec(\bar{G}) \cup L_\prec(\bar{G}') = L(G) \cup L(G')$. Finally, using Theorem 3.3.3 again, we can find a (3:3)MG $G=(G_1,G_2,M)$ such that $L(G)=L_\prec(G^*)=L(G) \cup L(G')$. This shows that (3:3)ML is closed under row concatenation. Q.E.D.
Theorem 3.3.5 The class of (1:1)ML is closed under row Kleene's closure, $0 \leq i \leq 3$.

Proof: We'll show for the case (3:3)ML. All other cases could be obtained similarly.

Let $G=(G_1',G_2,M)$ be a (3:3)MG. From Theorem 3.3.3 we see that we can find a (3:3)MG $G''=(G_1'',G_1',M')$ such that $L(G)=L(G'')$. From Theorem 3.9 of [11], we can find a regular string grammar $G_1''$ such that $L(G_1'')=L(G_1')^*$. Therefore, applying the same technique to the productions in each matrix we can construct a (3:3)MG $G'''=(G_1''',G_2'',M'')$ where $G_2''=G_2'$ and $M''=M'$ such that $L(G'')=L(G''')^*$ (similar to the proof of Theorem 3.3.2). Finally, using Theorem 3.3.3 again, we can find a (3:3)MG $G^*=(G_1^*,G_2^*,M^*)$ such that $L(G^*)=L(G''')$. This shows that (3:3)ML is closed under row Kleene's closure. Q.E.D.

Corollary 3.3.1 The class of (3:3)ML is closed under transpose.

Proof: Directly seen from Proposition 3.3.1 and Theorem 3.3.3. Q.E.D.

Corollary 3.3.2 The class of (3:3)ML is closed under reflection along rightmost axis.

Proof: Given a (3:3)MG $G=(G_1,G_2,M)$. Let $G'=G_1',G_2',M').$ Then $L(G')=L(G')^R$ (refer to Theorem 9.1 of [11]). Then $L(G)=L(G')$. Q.E.D.

Corollary 3.3.3 The class of (3:3)ML is closed under quarter turn.

Proof: Given a rectangle matrix $M$, $Q(M)=\tilde{Q}(M)$. Since MG are closed under $\tilde{C}$ and $\sim$, so is $Q$. Q.E.D.

Corollary 3.3.4 The class of (3:3)ML is closed under half turn.

Proof: A half turn equals two successive quarter turns. Q.E.D.

Corollary 3.3.5 The class of (3:3)ML is closed under reflection along the base.

Proof: $M=\tilde{C}(Q(M))$. Q.E.D.
From the closure properties of type 0, type 1 and type 2 string languages and the same technique employed for \((j;j)\)ML we can conclude

**Theorem 3.3.6** The class of \((i;i)\)ML is closed under union, row and column catenation and row and column Kleene's closure, \(i = 0,1,2,3\).

**Theorem 3.3.7** The class of \((i;i)\)ML is closed under operations transpose, quarter turn, half turn and reflection (along rightmost vertical axis and base), \(i=0,1,2,3\).

**Remark:** The Normal Forms for \((i;j)\)ML, \(i,j=2\), which are similar to the Chomsky and Greibach Normal Forms, should follow directly from the string case, applied to horizontal or vertical rules.
CHAPTER IV

FINITE TURN REPETITIVE CHECKING AUTOMATA

In this chapter we are going to construct machines that capture the family of (3:3) matrix languages. In Section 4.1 the Finite-Turn Repetitive Checking Automaton (FTRCA) is introduced. In Section 4.2 the equivalence relation between (3:3)ML and (3:3)FTRCA is established and an example of constructing a (3:3)FTRCA given a (3:3)MG is illustrated. This concept is generalized for other classes of FTRCA and a hierarchy is got.

We adapt the notations and definitions employed in [9,10].

4.1 Finite-Turn Repetitive Checking Automata

A Finite-Turn Repetitive Checking Automaton is essentially three finite state machines working in a sequential fashion and communicating with each other via tapes and a register that can assume nonnegative integer values less than or equal to k. The machines $M_1, M_2,$ and $M_3$ work roughly as follows: $M_1$ writes a string of symbols from $I_1$ followed by $\varnothing$ on tape 1 and places a value between 1 and k in register R. Machine $M_2$ reads a symbol from tape 1 and a symbol in the first column of the array simultaneously, beginning at the left and top respectively. The value in R remains constant. When a $\varnothing$ in tape 1 and a $\varnothing$ at the bottom of column are encountered simultaneously, the value of R changes to a value between 1 and k, a symbol is written on tape 3 and $M_2$ changes state and goes to the top of the next column. It then repeats the same process for the 2nd column beginning again with the leftmost symbol on tape 1. This goes on...
until all of the columns of the array have been read. Now $M_3$ takes over and reads tape 3 beginning with the leftmost or first symbol written on tape 3 by $M_2$. An array is accepted if $M_3$ enters a final state after read tape 3. A formal definition is given as follows:

Definition 4.1.1 A k-turn repetitive checking automaton (k-turn RCA) is an ordered quadruple

$$M = (M_1, M_2, M_3, R)$$

where $R$ is a finite set of integers, $R = \{0, 1, \ldots, k\}$,

$$M_1 = (\text{Ini}, T, R, \delta_1, q_0, \varnothing)$$

where \text{Ini} is a finite set of intermediate initial state $\{q_{i0}, \ldots, q_{k0}\}$,

$q_0$ is the initial state,

$T$ is a finite set of tape 1(t_1) symbols,

$\varnothing \in T$ is the endmarker

$$\delta_1: \{q_0\} \times \{0\} \rightarrow 2^{\{q_0\} \times T \times \{0\} \cup 2^{I_1 \times T \times \{0\}}} \text{Ini} \times \{\varnothing\} \times (R-\{0\})$$

with the following restriction:

1. for $c \in T$, $(q_0, c, 0) \in \delta_1(q_0, 0)$,

2. if $b \in \text{Ini}$, then $(b, \varepsilon, d) \in \delta_1(q_0, 0)$ where $d \in R-\{0\}$ and $b = q_d$

$M_2 = (\bar{K}, I, I', T, R, F, \delta_2, \varepsilon, \text{Ini})$

where $\bar{K}$ is a finite set of intermediate states, $\bar{K} = \bigcup_{i=1}^{k} K_i$, $K_i \cap K_j = \emptyset$, $i \neq j$ and

$q_{i0} \in K_i$, $i = 1, \ldots, k$,

$I$ is a finite set of input symbols, $\varnothing \notin I$,

$I'$ is a finite set of tape 2(t_2) symbols, $I' = \{s_1, \ldots, s_k\}$,

$F \in \bar{K}$ is a finite set of intermediate final states, $F = \bigcup_{i=1}^{k} F_i, F_i \in K_i$,

$$\delta_2: \bar{K} \times I \times \{\varepsilon\} \times T \times (R-\{0\}) \rightarrow 2^{\bar{K} \cup \{q_0\} \times (R-\{0\}) \times (I \cup \{\varepsilon\})}$$

with the following restrictions: $\forall b \in I, c \in T, d \in R-\{0\}$, $a, e \in \bar{K}$
(1) \((e, d, E) \in \delta(a, b, c, d)\),

(2) if \(\delta(a, b, c, d)\) contains \((e, d, E)\) and \((e', d, E)\) then \(e = e'\),

(3) if \(\delta(a, b, c, d)\) contains \((e, d, E)\) and \((a', b', c, d)\) contains
\((e, d, E)\) then \(a = a'\), \(b = b'\),

(4) if \(a \in F_d\) then
\[\delta(a, g, s, d) = \{(e, g, s_d), (q_0', g, s_d) | g \in R - \{0\}, e = q_0, s_d \in I'\}\]

(5) once in state \(q_0 \in K_1\), the transition states will be in \(K_1\) till

the input reading head encounters a "e".

\[M_3 = (K', \delta_3', F', q'_0)\]

where \(K'\) is a finite set of \(M_3\) states,
\(q'_0\) is the initial state of \(M_3\), \(q'_0 \in K'\),
\(F'\) is a finite set of final states, \(F' \subseteq K'\),
\(\delta_3': K' \times I' \rightarrow 2^{K'}\).

The construction of a k-turn RCA is shown in Figure 4.1.1.

Notice that the behavior of \(M_2\) and \(M_3\) are quite similar to a finite
state machine. We could call such a k-turn RCA an (FS:FS)kTRCA or (type 3:
type 3)kTRCA or just (3:3)kTRCA.

**Definition 4.1.2** Let \(M_a = (M_1, M_2, F_d, q, R)\) be a (3:3)kTRCA as defined in Definition 4.1.1. A configuration or instantaneous description (ID) of \(M_a\) is an
ordered 7-tuple
\[(q, (1, h), x, y, i, j, p)\]

where \(q \in K\),

\((1, h)\): the position of the input symbol under scanning (the position
of \(H_3\)), \(1 \leq 1 \leq m+1, 1 \leq h \leq n+1,\)
\(x \in T^*\) is the string in \(t_1'\),
\(y \in I'\) is the string in \(t_2'\),
\(1 \leq i \leq |x| \) \(+1\) is the position of \(H_1\) (Figure 4.1.1).
Figure 4.1.1 Construction of a (3:3) kTRGA
We define the following relations between ID's:

Initially ID = $(q_0, (0,1), \varepsilon, \varepsilon, 1,1,0)$

1. If ID = $(q_0, (0,1), x, \varepsilon, |x|, 1,0)$ and
   - (i) $(q_0, z_t, 0)$ is in $\delta_1(q_0, 0)$ then
     - $(q_0, (0,1), x, \varepsilon, |x|, 1,0) \vdash (q_0, (0,1), xz_t, \varepsilon, |x| +1, 1,0)$
   - but if
     - (ii) $(q_{po}, z, p)$ is in $\delta_1(q_0, 0)$, $1 \leq p \leq k$
     - then $(q_0, (0,1), x, \varepsilon, |x| +1, 1,0) \vdash (q_{po}, (1,1), x, \varepsilon, 1,1,0)$

2. If ID = $(q_{pd}, (1,h), xz,y,i,|y|+1,p), 1 \leq i \leq |x|+1$, the $i$-th symbol in $t_1$ is $z_t$, $y \in (I')^+$, and
   - (i) $(q_{pd}, (1,h), xz,y,i,|y|+1,p) \vdash (q_{pd}, (1,h), xz,y,i,|y|+1,p)$
   - but if $a = \varepsilon$ and $\varepsilon$ is not the last $\varepsilon$, then
     - (ii) $(q_{go}, z, s, p)$ is in $\delta_2(q_{pd}, a, z_t, p), a \neq \varepsilon$, then
       - $(q_{pd}, (1,h), xz,y,i,|y|+1,p) \vdash (q_{go}, (1,h+1), xz,y,i+1,|y|+1,p)$
   - and if this is the last column, $(q_{po}, p, s, p)$ is in $\delta_2(q_{pd}, \varepsilon, x, p)$, then
     - (iii) $(q_{pd}, (m+1,n), xz,y,|x|+1,|y|+1,p) \vdash (q_{po}, (1,n+1), xz,y,|x|+1,|y|+1,p)$

3. If ID = $(q, (1,n+1), xz,y, 1, j, p)$ and if $q'$ is in $\delta_3(q,a)$, $a$ is the $j$-th symbol of $y$ (in $t_2$) then
   - $(q, (1,n+1), xz,y, 1, j, p) \vdash (q', (1,n+1), xz,y, 1, j+1, p)$

Denote $\delta_1^*, \delta_2^*$ and $\delta_3^*$ be the transitive closure of $\delta_1$, $\delta_2$ and $\delta_3$ respectively, we can define the following:

**Definition 4.2.3** The language accepted by $M_a$ is defined as
\[ T(M_a) = \{ w \mid (q_o, (0,1), \varepsilon, \varepsilon, 1, 1, 0) \]  
\[ \delta_1^* (q_{po}, (1,1), x, \varepsilon, 1, 1, p) \]  
\[ \delta_2^* (q_{pj}, (m,1), x, \varepsilon, |x|, 1, p) \]  
\[ \delta_2^* (q_{pf}, (m+1,1), x, \varepsilon, |x| + 1, 1, p) q_{pf} \in F_p \]  
\[ \delta_2^* (q_{go}, (1,2), x, s, 1, 2, g) \]  
\[ \vdots \]  
\[ \delta_2^* (q_{vf}, (m+1,n), x, y, |x| + 1, |y| + 1, v) q_{vf} \in F_v \]  
\[ \delta_2^* (q_{q'}, (1,n+1), x, y, s, 1, u) 1 \leq u \leq k \]  
\[ \delta_3^* (q_{q'}, (1,n+1), x, y, s, 1, |y| + 2, u) q_{q'} \in F', x \in T^*_1 y \in (I')^*, w \in I^{++} \} . \]

4.2 The Equivalence of \((3:3)k\)TRCA and \((3:3)ML\)

We now want to show that \((3:3)k\)TRCA is equivalent to \((3:3)ML\).

**Theorem 4.2.1** If \(L\) is a \((3:3)ML\) generated by a \((3:3)MG\) \(G = (G_1, G_2, M)\), \(G_2 = \prod_{i=1}^{k} G_{2i}\) then there exists a \((3:3)k\)TRCA \(M_a\) such that \(L(G) = T(M_a)\).

**Proof:** Let \(M = \{ m_i \mid i = 1, \ldots, p \} \).

The \((3:3)k\)TRCA is given by \(M_a = (M_1, M_2, M_3, R)\). Set \(T = \{ Z_i \mid i = 1, \ldots, p \} \)

\(R = \{1, \ldots, k\}\), \(K\) can be found during the construction of the machine.

1. \((q_o, Z_t, 0)\) in \(\delta_1 (q_o, 0) 1 \leq t \leq p\)
2. \((q_{jo}, z, j)\) in \(\delta_1 (q_o, 0) 1 \leq j \leq k\)
3. For every matrix of the form \(m_i = \begin{bmatrix} s_1 \rightarrow a_{ii} r_{ii} \\ \vdots \\ s_k \rightarrow a_{ki} r_{ki} \end{bmatrix} \)

we have \((r_{ji}, j, \varepsilon)\) is in \(\delta_2 (q_{jo}, a_{ji}, Z_t, j), 1 \leq j \leq k\).
For every matrix of the form

\[
\begin{pmatrix}
S_1 \rightarrow a_{11} \\
S_2 \rightarrow a_{21} \\
\vdots \\
S_k \rightarrow a_{k1}
\end{pmatrix}
\]

we have \((q_{j1}, j, \xi) \in \delta_2(q_{j0}, a_{j1}, z_{1}, j), q_{jf} \in F_j, 1 \leq j \leq k\)

(3) For

\[
m_h = \begin{pmatrix}
l_{1h} \rightarrow a_{1h} r_{1h} \\
l_{2h} \rightarrow a_{2h} r_{2h} \\
\vdots \\
l_{kh} \rightarrow a_{kh} r_{kh}
\end{pmatrix}
\]

we have \((r_{jh}, j, \xi) \in \delta_2(1_{jh}, a_{jh}, z_{h}, j), 1 \leq j \leq k\)

If it is terminal as

\[
m_h = \begin{pmatrix}
l_{1h} \rightarrow a_{1h} \\
l_{2h} \rightarrow a_{2h} \\
\vdots \\
l_{kh} \rightarrow a_{kh}
\end{pmatrix}
\]

then \((q_{ji}, j, \xi) \in \delta_2(1_{jh}, a_{jh}, z_{h}, j), 1 \leq j \leq k\)

(4) Add \((q_{go}, g, s_j) \in \delta_2(q_{jf}, g, g, j), q_{jf} \in F_j, 1 \leq g \leq k, 1 \leq j \leq k, 1 \leq i \leq p.\)

\(\delta_j\) and \(M_j\) can be constructed from \(G_1\) the same way as a finite state machine constructed from a type 3 grammar (refer Theorem 3.4 of [11]).

It can be seen that \(w \in L(G) \iff w \in T(M_a).\) Q.E.D.
Example 4.2.1 Consider an (R:R)MG $G = (G_1, G_2, M)$ where $G_1 = (V_1, P_1, I_1, S)$ with $P_1 = \left\{ S \rightarrow s_1 S, S \rightarrow s_1 S_1, S_1 \rightarrow s_2 S_1, S_1 \rightarrow s_2 \right\}$.

$G_2 = \left\{ G_{21} \right\} \cup \left\{ G_{22} \right\}$

$M = \left\{ m_i \mid i = 1, \ldots, 4 \right\}$.

\[
\begin{align*}
    n_1 &= \left\{ \begin{array}{l}
        s_1 \rightarrow B_{s_1} \\
        s_2 \rightarrow R_{s_2}
    \end{array} \right. \\
    n_2 &= \left\{ \begin{array}{l}
        s_1 \rightarrow B_{s_1} \\
        s_2 \rightarrow R_{s_2}
    \end{array} \right. \\
    n_3 &= \left\{ \begin{array}{l}
        s_{11} \rightarrow R_{s_{11}} \\
        s_{21} \rightarrow B_{s_{21}}
    \end{array} \right. \\
    n_4 &= \left\{ \begin{array}{l}
        s_{11} \rightarrow R \\
        s_{21} \rightarrow B
    \end{array} \right.
\end{align*}
\]

It can be seen that

\[
L(2) = \left\{ \left( \begin{array}{l} \mathbb{R}_n \end{array} \right)^+ \left( \begin{array}{l} \mathbb{R}_h \end{array} \right)^+ \mid n \geq 1, h \geq 1 \right\}
\]

For instance, $S \xrightarrow{\ast} s_1 s_1 s_2 s_2 s_2 s_2$ 

\[
\begin{array}{cccccc}
    B & B & R & R & R \\
    B & B & R & R & R \\
    R & R & B & B & B \\
    R & R & B & B & B \\
    R & R & B & B & B \\
\end{array}
\]

$\in L(G)$
To construct a (3:3)kTRCA that captures this language, we have

\[ M_a = (M_1', M_2', M_3', R) \]

where \( R = \{1, 2\} \). \( M_1', M_2' \) and \( M_3' \) can be obtained from the following:

1. \( \delta_1(q_0, 0) = \{(q_0, Z_t, 0), (q_0, \varepsilon, j) \mid t = 1, \ldots, 4, j = 1, 2\} \)
2. \( \delta_2(q_{10}, B, Z_1, 1) = \{(q_{10}, 1, \varepsilon )\} \)
   \( \delta_2(q_{20}, R, Z_1, 1) = \{(q_{20}, 2, \varepsilon )\} \)
   \( \delta_2(q_1, B, Z_2, 1) = \{(s_{11}, 1, \varepsilon )\} \)
   \( \delta_2(q_{20}, R, Z_2, 1) = \{(s_{21}, 2, \varepsilon )\} \)
   \( \delta_2(s_{11}, R, Z_3, 1) = \{(s_{11}, 1, \varepsilon )\} \)
   \( \delta_2(s_{21}, R, Z_3, 2) = \{(s_{21}, 2, \varepsilon )\} \)
   \( \delta_2(s_{11}, R, Z_4, 1) = \{(q_{10}, 1, \varepsilon )\} \)
   \( \delta_2(s_{21}, R, Z_4, 2) = \{(q_{20}, 2, \varepsilon )\} \)
   \( \delta_2(q_{1f}, \varepsilon, \varepsilon, i) = \{(q_{10}, j, s_1), (q_0, j, s_1) \mid i = 1, 2\} \)
3. \( \delta_3(q_0', s_1) = \{q_0', s_1\} \)
   \( \delta_3(s_1, s_2) = \{s_1, q_{1f}\} \)

\( F_1 = \{q_{1f}\}, \quad F_2 = \{q_{2f}\} \)
\( K_1 = \{q_{10}, q_{1f}, R_{11}\}, \quad K_2 = \{q_{20}, q_{2f}, s_{21}\}, \quad \text{Ini} = \{q_{10}, q_{20}\} \)
\( T' = \{s_1, s_2\} \).

It can be seen that \( L(G) = T(M_a) \). For the input array

```
B B R R R
B B R R R
R R B B B
R R B B B
\varepsilon \neq \varepsilon \neq \varepsilon
```

the recognition process is shown in the following sequence of ID's:
(q_o, (0,1), \epsilon, \epsilon, 1, 1, 0)

\begin{align*}
\delta_1^* (q_o, (0,1), & Z_1 Z_2 Z_3 Z_4 \epsilon, \epsilon, 6, 1, 0) \\
\delta_1 (q_{10}, (1,1), & Z_1 Z_2 Z_3 Z_4 \epsilon, \epsilon, 1, 1, 1) \\
\delta_2 (q_{10}, (1,2), & Z_1 Z_2 Z_3 Z_4 \epsilon, \epsilon, 2, 1, 1) \\
\delta_2 (s_{11}, (1,3), & Z_1 Z_2 Z_3 Z_4 \epsilon, \epsilon, 3, 1, 1) \\
\delta_2 (s_{11}, (1,4), & Z_1 Z_2 Z_3 Z_4 \epsilon, \epsilon, 4, 1, 1) \\
\delta_2 (s_{11}, (1,5), & Z_1 Z_2 Z_3 Z_4 \epsilon, \epsilon, 5, 1, 1) \\
\delta_2 (q_{1f}, (1,6), & Z_1 Z_2 Z_3 Z_4 \epsilon, \epsilon, 6, 1, 1) \\
\delta_2 (q_{10}, (2,1), & Z_1 Z_2 Z_3 Z_4 \epsilon, s_1, 1, 1, 2) \\
\vdots \\
\delta_2^* (s_{21}, (6,5), & Z_1 Z_2 Z_3 Z_4 \epsilon, s_1 s_1 s_2 s_2, 5, 5, 2) \\
\delta_2 (q_{2f}, (6,6), & Z_1 Z_2 Z_3 Z_4 \epsilon, s_1 s_1 s_2 s_2 s_2, 6, 5, 2) \\
\delta_3 (q_f, (1,7), & Z_1 Z_2 Z_3 Z_4 \epsilon, s_1 s_1 s_2 s_2 s_2, 1, 1, 2) \\
\delta_3 (q_f, (1,7), & Z_1 Z_2 Z_3 Z_4 \epsilon, s_1 s_1 s_2 s_2 s_2, 1, 2, 2) \\
\delta_3 (s_1, (1,7), & Z_1 Z_2 Z_3 Z_4 \epsilon, s_1 s_1 s_2 s_2 s_2, 1, 3, 2) \\
\delta_3 (s_1, (1,7), & Z_1 Z_2 Z_3 Z_4 \epsilon, s_1 s_1 s_2 s_2 s_2, 1, 4, 2) \\
\delta_3 (q_f, (1,7), & Z_1 Z_2 Z_3 Z_4 \epsilon, s_1 s_1 s_2 s_2 s_2, 1, 5, 2)
\end{align*}

Since q_f \in F', it is accepted.
Next we are going to show the following:

**Theorem 4.2.2** If $M_a = (M_1, M_2, M_3, R)$ is a $(3:3)k$TRCA, then there exists a $(3:3)k$MG $G$ such that $L(G) = T(M_a)$.

**Proof:** Let $T = \{Z_0, \ldots, Z_{p-1}\}$, $R = \{1, \ldots, k\}$.

The construction of $G = (G_1, G_2, M)$ can be obtained as follows:

Intuitively, each matrix of $M = \{m_1, m_2, \ldots, m_p\}$ is obtained from the $S_2$ functions with the same values of $T$ symbols and $R$ values ranging from 1 to $k$. Notice that from the definition of a $(3:3)k$TRCA, fixed $T$ values there should be exactly $k$ transitions. Therefore in each column matrix there should be exactly $k$ rules. Formally, with the help of Figure 4.2.1 we see that:

For $\delta_2(q_{jt}, a_j, z_d, j) = \{(q_{jt}, j, c) | 1 \leq j \leq k, 0 \leq t, d \leq p-1\}$, we have

$$
\begin{bmatrix}
q_{1t} \rightarrow a_1 q_{1t'} \\
\vdots \\
q_{kt} \rightarrow a_k q_{kt'}
\end{bmatrix} \in M
$$

For $q_{it} \in F_1$, $\delta_2(q_{jt}, a_j, z_d, j) = \{(q_{jt}, j, c)\}$

we have

$$
\begin{bmatrix}
q_{1t} \rightarrow a_1 \\
\vdots \\
q_{kt} \rightarrow a_k
\end{bmatrix} \in M
$$

$P_{2i}$ is obtained by taking the union of the $i$-th rule of each matrix, with $s_i = q_{10}$, $i = 1, \ldots, k$. $G_1$ is obtained from $M_2$ just as a grammar from a
Figure 4.2.1 The proof of Theorem 4.2.2
finite state machine (please refer to Theorem 3.5 of[11]).

It can be seen that \(L(G) = T(M_a).\)

Q.E.D.

4.3 Recognition of other Sequential/Parallel Matrix Languages

Recall Chomsky hierarchy of machines. Let's define type 3=finite state, type 2=pushdown, type 1=linear bounded and type 0=Turing. As in Section 4.2, we can find \((i:j)kTRCA\) corresponding to \((j:i)ML, i,j=0,1,2,3\).

For instance, if we make \(M_2\) and \(M_3\) pushdown automata and appropriately modify \(\delta_2\) and \(\delta_3\), we can have a \((2:2)kTRCA\) (see Figure 4.3.1).

Informally we can define a \((2:2)kTRCA\) as follows:

\[M_a = (M_1, M_2, M_3, R)\]

where \(M_1\) and \(R\) are exactly as before,

\[M_2 = (K, l, l', T, R, F, \delta_2, \theta, Ini, \Gamma_1)\]

where \(K, l, l', T, R, Ini\) are as before, \(\Gamma_1\) is the set of pushdown store 1 (pds1) symbols for vertical grammars. \(\delta_2\) takes care of \(\Gamma_1\) symbols just like a pushdown automaton with empty store acceptance.

\[M_3 = (K', \delta_3, \Gamma_2, q_0')\]

where \(K', q_0'\) are as before, \(\Gamma_2\) is the set of pushdown store 2 (pds2) symbols for horizontal grammar. \(\delta_3\) takes care of \(\Gamma_2\) symbols just like a pushdown automaton with empty store acceptance.

\(M_a\) operates similarly as \((3:3)kTRCA\), except \(M_2\) and \(M_3\) work as pushdown automata with appropriate constraints (see Section 4.1).

The acceptance of a matrix can be defined in the following way:

Initially, an initial symbol \(Z_0^1 \in \Gamma_1\) is in pds1, an initial symbol \(Z_0^2 \in \Gamma_2\) is in pds2. Everything else is exactly the same as a \((3:3)kTRCA\). After the pooling of \(t_1\) is finished, \(M_3\) begins to scan downward. The construction of \(\delta_2\) function is very similar to Section 4.2 plus the method
Figure 4.3.1 Construction of a (2:2) kTRCA
for a pushdown automaton transition function (empty store acceptance, Theorem 5.2 [11]). Everytime a column is successfully scanned, pds1 should be empty. H_3 then adds Z_0^2 to pds1, H_2 writes a symbol on t_2, R changes value, H_3 moves to the top of next column, ready to scan downward and so on. When all the column scanning are successfully done, pds1 must be empty and H_2 moves to the bottom of t_2, ready to scan the string in t_2. From now on H_2 and H'_2 are working exactly as a pushdown automaton according to \( \delta_3 \) function. The matrix accepted if when H_2 exhausts the string in t_2, pds2 is empty.

A formal notion of \( (3:3)kTRCA \), its ID's, relation between ID's and definition of acceptance by empty stores could be formulated quite similar as in Section 4.2 and hence omitted here.

**Theorem 4.3.1** The class of \( (2:2)kTRCA \) is equivalent to \( (2:2)ML \).

Proof: The proof is quite similar to that in **Theorem 4.2.1**, Theorem 4.2.2 and Theorems 5.2 and 5.3 of [11].

Q.E.D.

From the above description, it is quite clear that the behavior of a kTRCA depends on \( M_2 \) and \( M_3 \). With appropriate modification we could similarly construct \( (i:j)kTRCA \), \( i=0,1,2,3 \), \( j=0,1,2,3 \) and it is easy to see that

**Theorem 4.3.2** The class of \( (i:j)kTRCA \) is equivalent to the class of \( (j:i)ML \), \( j,i = 0,1,2,3 \).

Comparing our model with Siromoney's matrix automaton [25] we find that the basic differences between a kTRCA and a matrix automaton is that a matrix automaton does not have

(1) the capability of pooling up \( t_1 \) symbols,
Briefly, the input is as follows:

\[
\begin{array}{cccc}
\ldots & \ldots & a_{1n} \\
\vdots & \vdots \\
a_{m1} & \ldots & a_{mn} \\
\$ & \ldots & \$
\end{array}
\]

The machine starts from \(a_{11}\), scanning downward until \(\$\), then it prints a symbol from \(T\) on the tape and goes to the top of next column and repeat the same process. Until the last \(\$\) is scanned and a symbol from \(T\) is written on the tape, the machine begins to scan the tape symbols from left-right as if it is a finite state machine.

As will be discussed in Chapter VI, \((3; i) \subseteq iML, i = 0, 1, 2, 3\), we can see that Siromoney's matrix automaton is a special case of a kTRCA.
CHAPTER V

ARRAY GRAMMARS AND LANGUAGES

A simple property-specifying device for a class of languages has the definite advantage that each language in the class can be defined explicitly by the device, in contrast to the implicit representation of languages by grammars. For instance, a regular language can be represented by a regular expression. However, no such characterization is known for 2-dimensional languages. In this chapter, we introduce array grammars and regular-like expressions that describe sets of well-formed formula, which can be interpreted as rectangular arrays with fixed proportions. We adapt the definitions and notations from Salomaa[23].

5.1 Definitions and Notations

Definition 5.1.1 Let a be a letter, and let L and L_1 be string languages. The a-substitution of L_1 into L, in symbols, LS_aL_1, is defined by

\[ LS_aL_1 = \{ P \mid P = P_1P_2\ldots P_kQ_kP_{k+1}, \quad k \geq 0, \]
\[ \text{and } Q_j \in L_1, \text{ for } 1 \leq j \leq k \} \]

Notice that for each a, S_a is an associative operation. Therefore we may write "products" associatively, for instance, LS_aL_1S_aL_2S_aL_3S_aL_4.

The notation S_a(L_i) means the S_a-product with i \geq 2 factors L. By definition, S_a(L_1) = L. The iterated a-substitution closure of L, in symbols, L^{+a}, consists of all words P which satisfy the following two conditions: (1) For some i \geq 1, P \in S_a(L_i) and (2) a is not a subword of P.
The iterated a-substitution closure is closely related to the operations superscript asterisk(*) and plus(+). It is a direct consequence of the definition that if \( L \) is a language over an alphabet not containing \( a \), then \( L^* = (L a U L)^a \), \( L^+ = (L a U L)^+ a \).

**Definition 5.1.2** Assume that \( V_1 \) and \( V_2 = \{ \emptyset, 0, 1, +, a, \Lambda, (,) \} \) are disjoint alphabets, where \( \emptyset \) and \( 0 \) are row catenation and column catenation operators defined as in Section 3.1, \( U \) means union, \(+\) is a special repetition operator, \( \emptyset \) means empty set, \( \Lambda \) means empty array, \( (\) means left parenthesis, \( ) \) means right parenthesis.

A regular-like expression over \( V_1 U V_2 \) can be defined recursively as follows:

(i) \( \emptyset, \Lambda \) and a letter of \( V_1 \) are regular-like expressions,

(ii) If \( a \in V_1 \), \( Q \) and \( R \) are regular-like expressions, so are \( (Q \emptyset R), (Q \oplus R), (QUR) \) and \( (Q)^a \),

(iii) Nothing else is a regular-like expression.

**Example 5.1.1** \( ((a \emptyset c \oplus b U \Lambda)^c \emptyset d U \Lambda)^d \) is a regular-like expression. (Notice that we have omitted unnecessary parenthesis).

This regular-like expression denotes the set of arrays \( \{a^i b^j \mid i, j \geq 0\} \).

Now we are in a position to define array grammars and languages.

**Definition 5.1.3** An array grammar (AG) is a quadruple \( G = (V, T, P, S) \), where \( V = V_1 U V_2 \), where

- \( V_1 \) is a finite set of nonterminals,
- \( V_2 \) is a finite set of intermediates,
- \( T \) is a finite set of terminals, and
- \( P = P_1 U P_2' \) where
- \( P_1 \) is a finite set of intermediate rules,
$P_2$ is a finite set of terminal rules, and

$S \in V_1$ is the start symbol, $(,) \notin V_2$.

Detailed explanations are given as follows:

$P_1$ is a set of ordered pairs $(u,v)$ written as $u \rightarrow v$, $u,v \in (V_1 \cup V_2)^*$, $v$ must be a well-formed formula (wff) which is defined as follows:

(i) each member in $V_1 \cup V_2 - \{(,)\}$ is a wff,
(ii) if $a,b$ are wffs so are $(a \land b)$, $(a \lor b)$,
(iii) nothing else is a wff.

The strings generated by $P_1$ is $\{x | S \rightarrow x \in V_2^*\}$. The intermediate languages generated by $P_2$ are matrix languages as defined in Chapter III. For convenience we denote the language generated by $P_2$ with initial symbol $A \in V_1$ by $L_A = \{x | A \rightarrow^{*+} x \in T^+\}$.

Thus we may think of $P_2$ as a set of pairs $P_2 = \bigcup_{A \in V_1} P(A)$, each pair consisting of two sets of rules, the first one of which is for horizontal, sequential productions; the second of which is for vertical, parallel productions.

The language generated by an array grammar is explained as below:

Starting with $S$, the rules of $P_1$ are applied until all nonterminals are replaced with intermediates. Thus we obtain a horizontal string over $V_2$. Now each symbol $A \in V_2 - \{(,)\}$ will be replaced according to $P_2$ until every symbol is replaced by a terminal. If each rule in $P_2$ is $(CF:CF)$, $P_1$ is $CF$, then according to Theorem 11.4 of [23] we see that the collection of all wffs generated by an array grammar can be represented by a regular-like expression defined in Definition 5.1.

It can be interpreted as a set of rectangular arrays by properly processing operators such as $\theta$, $\varnothing$, $U$ and the repetition operator $+$, subject to the condition imposed by the row and column catenation.
5.2 An Example: Digitized English Letter "R" With Fixed Proportions

In this section we are going to show an example of array grammar which generates a set of digitized English letter "R" with fixed proportion, say 1:1.

Example 5.2.1 Let $G = (V, T, P, S)$ be an array grammar, where

\[ V = \{S, S_1\} \cup \{A, B, C, D, E, F, G, H, K, (, ), \emptyset, \emptyset\} \]

\[ T = \{., x\} \]

\[ P_1 = \{S \rightarrow (F \emptyset ((D \emptyset S_1) \emptyset E) \emptyset G), \]

\[ S_1 \rightarrow (C \emptyset ((A \emptyset S_1) \emptyset E) \emptyset C), \]

\[ S \rightarrow H, S_1 \rightarrow K \} \]

\[ L_K = \{x\ x\ x\} = (\emptyset, \emptyset, \emptyset) \oplus (x\emptyset \x x) \oplus (\emptyset, \emptyset, \emptyset) \]

\[ L_A = \{(.,)^n.(.)^n | n \geq 1\} = ((.\emptyset\emptyset\emptyset) \cup (\emptyset, \emptyset, \emptyset))^+ \]

\[ L_B = \{(.,)^n x(.,)^n | n \geq 1\} = ((.\emptyset\emptyset\emptyset) \cup (\emptyset x\emptyset, \emptyset))^+ \]

\[ L_C = \{(.,)^n | n \geq 1\} = ((. \emptyset \emptyset \emptyset) \cup (\emptyset x \emptyset, \emptyset))^+ \]

\[ L_D = \{x^n | n \geq 1\} = ((x \emptyset a) \cup x)^+ \]

\[ L_E = \{(.,)^{n-1} x^n | n \geq 1\} = ((. \emptyset a \emptyset x) \cup x)^+ \]

\[ L_F = \{(x,)^n | n \geq 5\} = ((x \emptyset a) \cup (x \emptyset x \emptyset x \emptyset x \emptyset x))^+ \]

\[ L_G = \{(.,)^{n-2} | n \geq 2\} = ((x \emptyset a \emptyset \emptyset) \cup (x \emptyset x))^+ \\emptyset x \]

\[ L_H = \{x\ x\ x\ x\ x\ x\} = (x \emptyset x \emptyset x) \oplus (x \emptyset x \emptyset x) \oplus (x \emptyset x \emptyset x) \]

Again we omit all unnecessary parenthesis.
It can be seen that

\[ L(G) = (L_D \oplus ((L_A \oplus L_B) \ominus L_C) \cup L_K)^+ \ominus L_E \ominus L_C \cup L_H \]

which can be interpreted as a set of all digitized English letter "R" with fixed proportion 1:1. The first four members of \( L(G) \) are listed in Figure 5.2.1. A derivation process is illustrated in Figure 5.2.2.

Remark: The method introduced in this Chapter is still rudimentary. We may want to call the array grammar in Example 5.2.1 a \((CF:(CF:CF))\) or \((2:(2:2))\) AG. We still don't know how to represent a context-sensitive language in terms of an expression defined similarly as in Section 5.1.1. One possible suggestion to define such an expression would be to find a grammar that generates a class of regular-like expressions. We hope to explore this in the future and find a hierarchy for array languages.
Figure 5.2.1 The first four members of L(G).
(1) Replace a by Lₖ, we obtain 
\[
\begin{array}{ccc}
\cdot & x & x \\
\cdot & x & x \\
\cdot & x & x \\
\cdot & x & x \\
\cdot & x & x \\
\end{array}
\]

(2) Replace Lₐ by \ldots, combine with Lₖ, we have 
\[
\begin{array}{ccc}
\cdot & x & x \\
\cdot & x & x \\
\cdot & x & x \\
\cdot & x & x \\
\cdot & x & x \\
\end{array}
\]

(3) Replace Lₐ by \ldots., combine with above, we have 
\[
\begin{array}{ccc}
\cdot & x & x \\
\cdot & x & x \\
\cdot & x & x \\
\cdot & x & x \\
\cdot & x & x \\
\end{array}
\]

(4) Replace Lₖ by \ldots, combine with the above, we have 
\[
\begin{array}{ccc}
\cdot & x & x \\
\cdot & x & x \\
\cdot & x & x \\
\cdot & x & x \\
\cdot & x & x \\
\end{array}
\]

(5) Replace Lₐ by \ldots., combine with the above 
\[
\begin{array}{ccc}
\cdot & x & x \\
\cdot & x & x \\
\cdot & x & x \\
\cdot & x & x \\
\cdot & x & x \\
\end{array}
\]

result, we have 
\[
\begin{array}{ccc}
\cdot & x & x \\
\cdot & x & x \\
\cdot & x & x \\
\cdot & x & x \\
\cdot & x & x \\
\end{array}
\]

(6) Replace Lₕ by \ldots, and replace Lₖ by \ldots, combine with above, we have 
\[
\begin{array}{ccc}
\cdot & x & x \\
\cdot & x & x \\
\cdot & x & x \\
\cdot & x & x \\
\cdot & x & x \\
\end{array}
\]

\[\in L(G)\]

*Figure 5.2.2* The derivation process of \(M_3\).
CHAPTER VI

DISCUSSIONS AND CONCLUSIONS

In dealing with two-dimensional pattern processing, we observe that our model has the following advantages over others:

(1) Some very natural hierarchies based upon Chomsky’s are induced. This has not been the case with other methods. [13, 16, 17] Many results in one-dimensional formal language theory can be applied to our method, e.g. closure property, uvwxy intercalation theorem, hierarchies of languages and acceptors, and regular-like expressions.

(2) As has been pointed out in Miller and Shaw [15] one of the limitations of their Picture Descriptive Languages (PDL) is that they cannot describe simple operations such as rotations, reflection, transpose, etc. In our model, these simple operations are easily and elegantly described.

(3) In Narasimhan [16], it is pointed out that it is desirable to work with operations on digitized pictures which can be defined in terms of functions having very few arguments. A general form of homomorphism can be defined which corresponds to magnification, topological transformation and elongation. These transformations are either directly obtained from the grammar itself (as in the case of reflection, rotation, etc.) or defined as a function of the terminal letters only (as in the case of magnification, topological transformation etc.). Thus the number of arguments for the
functions are few.

(4) From sequential/parallel point of view, our method is a compromise (horizontally, sequentially and vertically in parallel) between purely sequential methods (which take too much time for large arrays) and purely parallel methods (which take too much hardware construction for large arrays).

(5) Our matrix grammars can generate some interesting patterns which can not be generated by Siromoney's matrix grammars[25]. For instance the pattern introduced in the end of Chapter II can be generated by the following (CF,R)MG

\[ G = (G_1, G_2, M) \], where

\[ G_1 = (V_1, I_1, P_1, S_1) \]

\[ V_1 = \{ S_1, S_{11}, S_{12} \} \]

\[ I_1 = \{ s_1, s_2 \} \]

\[ P_1 = \{ \begin{array}{c} S_1 \rightarrow s_1 S_{11} , \\ S_{11} \rightarrow s_1 S_{11} , \\ S_{11} \rightarrow s_2 S_{12} , \\ S_{12} \rightarrow s_2 S_{12} , \\ S_{12} \rightarrow s_2 \end{array} \} \]

\[ G_2 = \{ G_{21} \} \cup \{ G_{22} \}, \text{ where} \]
\[ G_{21} = (V_{21}, I_2, P_{21}, s_1) \]

\[ V_{21} = \{ s_1, s_{11}, s_{12} \} \]

\[ I_2 = \{ R, B \} \]

\[ G_{22} = (V_{22}, I_2, P_{22}, s_2) \]

\[ V_{22} = \{ s_2, s_{21}, s_{22} \} \]

\[ M = \{ m_i \mid i = 1, \ldots, 5 \} \]

where

\[ m_1 = \begin{cases} 
  s_1 \rightarrow B s_{11} \\
  s_2 \rightarrow R s_{21}
\end{cases} \]

\[ m_2 = \begin{cases} 
  s_{11} \rightarrow B s_{11} \\
  s_{21} \rightarrow R s_{21}
\end{cases} \]

\[ m_3 = \begin{cases} 
  s_{11} \rightarrow R s_{12} \\
  s_{21} \rightarrow B s_{21}
\end{cases} \]

\[ m_4 = \begin{cases} 
  s_{12} \rightarrow R s_{12} \\
  s_{22} \rightarrow B s_{22}
\end{cases} \]

\[ m_5 = \begin{cases} 
  s_{12} \rightarrow R \\
  s_{22} \rightarrow B
\end{cases} \]

It can be seen that \( L(G) = \left\{ \left( \begin{array}{c} B^m \\ R_n \end{array} \right)^p \left( \begin{array}{c} R^m \\ p_n \end{array} \right)^p \mid p, m, n \geq 2 \right\} \).

Before we make a comparison between our matrix grammar and Siromoney's matrix grammar, let's recall the definition from Siromoney[25].

**Definition 6.1** A phrase-structure matrix grammar (PSMG) (Context-sensi-
tive matrix grammar (CSMG), context-free matrix grammar (CFMG), right-linear matrix grammar (RLMG) is a pair \( G = (G_1, G_2) \), where \( G_1 = (V_1, I_1, P_1, S) \) is a phrase-structure grammar (PSG), (Context-sensitive grammar (CSG), context-free grammar (CFG), right linear grammar (RLG)), with \( : V_1 = \) a finite set of horizontal nonterminals, \( I_1 = \) a finite set of intermediates = \( S_1, \ldots, S_k \), \( P_1 = \) a finite set of PSG(CSG,CFG,RLG) production rules called horizontal production rules, and \( S \) is the start symbol, \( S \in V_1 \), and \( V_1 \cap I_1 = \emptyset \). \( G_1 = \bigcup_{i=1}^{k} G_{2i} \) where \( G_{2i} = (V_{2i}, I_2, P_{2i}, S_i) \), \( i = 1, \ldots, k \) are right linear grammars with \( : I_2 = \) a finite set of terminals, \( V_{2i} = \) a finite set of vertical nonterminals, \( S_i = \) the start symbol of \( G_{2i} \) and \( P_{2i} = \) a finite set of right linear production rules, \( V_{2i} \cap V_{2j} = \emptyset \) if \( i \neq j \). Derivations are defined as follows: First a string \( a_1a_2\ldots a_n \in I_1^* \) is generated horizontally using the horizontal production rules in \( P_1 \), i.e. \( S \xrightarrow{*} a_1a_2\ldots a_n \in I_1^* \). Vertical derivations proceed as follows: the choice of a vertical production rule for a particular column at any stage is fixed by the initial symbol of that column determined by the horizontal string. The rules in the \( k \) right linear grammars are assumed to be of the form \( A \rightarrow aB \) or \( A \rightarrow a \), a \( \in I_2 \) so that at each stage, a single terminal is generated in each column vertically. The set of rules applied at one stage is not fixed but restricted by the horizontal string generated at the first stage. The application of the vertical production rules is simultaneous. For each column, a rule from one of \( G_{2i} \) must be applied at each stage. The derivation halts when all symbols are terminals. The language generated by \( G \) is defined as the set of all terminal matrices derived from \( G_1 \) and \( G_2 \).
That the language on page 48 can not be generated by any Siromoney's matrix grammar can be seen as follows:

Let $G = (G_1, G_{21} \cup G_{22})$ be a CFMG such that $L(G) = \{B^p R^m | p R^m \geq 2\}$.

We have $L(G_1) = \{S_1^n S_2^n | n \geq 2\}$, and

$L(G_{21}) = \{B^m R^n | m, n \geq 2\}$, $L(G_{22}) = \{R^m B^n | m, n \geq 2\}$.

From the definition of matrix grammar [25], when vertical rules are applied simultaneously, there is no control over columns starting from different intermediates. Consider the following derivation:

$$
\begin{align*}
S & \Rightarrow S_1 \ldots S_1 S_2 \ldots S_2 \\
& \downarrow^* \\
& \{B \quad B \quad R \quad R \} \\
m & \quad \ldots \quad \ldots \quad \ldots \\
& \{B \quad B \quad R \quad R \} \\
R & \quad R \quad B \quad B \\
& \quad \ldots \quad \ldots \quad \ldots \\
& \quad \ldots \quad \ldots \quad \ldots \\
R & \quad R \quad B \quad B \\
\end{align*}
$$

Since there is lack of control over columns starting from different intermediates, we can also derive the following:

$$
\begin{align*}
\{B \ldots B R \ldots R \} \\
m & \quad \ldots \quad \ldots \quad \ldots \\
& \{B \quad B \quad B \quad B \} \\
R & \quad R \quad \ldots \quad \ldots \\
& \quad \ldots \quad \ldots \quad \ldots \\
R & \quad R \quad B \quad B \\
\end{align*}
$$

This shows that if $\forall p, m, n, m', n \geq 2, (B^p R^m) R^m (B^p R^m) L(G) \iff (B^p R^m) (B^p R^m) L(G)$.

This is a contradiction.
Now we will show that the generative power of the matrix grammar introduced in Chapter III is greater than that of Siromoney's.

**Theorem 6.1** \( \texttt{iML} \subseteq (\texttt{iR})\texttt{ML} \) \( i = 0, 1, 2, 3 \), where \( \texttt{iML} \) means type \( i \) ML.

**Proof:** We prove the case for \( i = 1 \). The horizontal generative capabilities of a CFMG and a \((\texttt{CF:R})\texttt{MG}\) are the same, i.e. that of a context-free grammar. We now want to show that for every CFMG \( G=(G_1,G_2) \) as defined in [25], there exists a \((\texttt{CF:R})\texttt{MG}\) \( G'=(G'_1,G'_2,M) \) such that \( L(G') = L(G) \).

Suppose \( G_2 = \bigcup_{i=1}^{k} \{ G_{2i} \} \), then let \( G'_2 = G_2 \), \( G'_{2i} = G_{2i} \). The control matrix \( M \) is obtained by combining all possible combinations of the rules in each \( P_{2i} \). For instance, let \( k = 2 \) and

\[
P_{21} = \{ a_1, a_2, \ldots, a_m \},
\]

\[
P_{22} = \{ b_1, b_2, \ldots, b_n \}, \text{ with } m, n \neq 1
\]

then we have the following matrices:

\[
\begin{bmatrix}
a_1 \\
b_1
\end{bmatrix}
\begin{bmatrix}
a_1 \\
b_2
\end{bmatrix}
\ldots
\begin{bmatrix}
a_1 \\
b_m
\end{bmatrix}
\begin{bmatrix}
a_2 \\
b_1
\end{bmatrix}
\ldots
\begin{bmatrix}
a_2 \\
b_n
\end{bmatrix}
\ldots
\begin{bmatrix}
a_m \\
b_1
\end{bmatrix}
\ldots
\begin{bmatrix}
a_m \\
b_n
\end{bmatrix}
\]

It is seen that \( w \notin L(G) \iff w \notin L(G') \), This shows that \( \texttt{CFML} \subseteq (\texttt{CF:R})\texttt{ML} \).

An example on page 48 shows that \( \texttt{CFML} \subseteq (\texttt{CF:R})\texttt{ML} \). Similarly for the case \( i = 0, 1, 2, 3 \). Some examples of RML, CFML and CSML are shown in Figures 6.1-6.3.

Q.E.D.

The above results are summarized in Figure 6.4.

(6) The extension to higher dimensions is easy.
An effective construction for the hierarchy of machines that recognize the hierarchy of matrix languages was given in Chapter IV.

There are still some disadvantages in our matrix grammars. For instance, they can not generate patterns with fixed proportions, nor can they generate patterns containing non-vertical, non-horizontal lines. This is because of the independence between horizontal and vertical derivations. However, these disadvantages are overcome by array grammars introduced in Chapter VI.

Example 4.2.1. An array grammar which generates a language containing non-vertical, non-horizontal line is shown as follow.

Consider the following \((\text{CF:}(R:R))\) AG

\[
G = (V,I,P,S), \text{ with } V= \{S,B,C\} \quad \text{and} \quad I = \{.,X\}
\]

\[
P_1 = \left\{ S \rightarrow C \circ (B \circ S \circ B) \circ C \right. \\
S \rightarrow X . X \\
S \rightarrow X . X \\
\left. \right\}
\]

\[
L_B = \left\{ (.)^n \mid n \geq 3 \right\}
\]

\[
L_C = \left\{ X(\cdot)^n X \mid n \geq 3 \right\}
\]

It is seen that \(L(G) = \{\text{set of all digitized "X" with 1:1}\}\).

For instance

\[
\begin{align*}
X & \quad . & \quad X & \quad . & \quad X \\
X & \quad . & \quad X & \quad X & \quad . \\
. & \quad X & \quad . & \quad X & \quad . \\
X & \quad . & \quad X & \quad . & \quad X \\
X & \quad . & \quad X & \quad . & \quad X \\
X & \quad . & \quad X & \quad . & \quad X \\
\end{align*}
\]

\(\text{etc } \in L(G)\)
Figure 6.1 The set of all digitized English letter "L" with different proportions, which falls in RML
Figure 6.2 The set of all digitized English letter "I" with different proportions, which falls in CFML
Figure 6.3 A language which falls in CSML
Figure 6.4 The relation between iML and (i:R)_ML, i=0,1,2,3
Now we are going to discuss some topics for future research.

Since our method provides a compromise between purely sequential methods and purely parallel methods for 2-dimensional patterns, we would like to explore further in this field; e.g. the complexities of sequential/parallel mixed mode matrix languages. Basically there are four different kinds of modes, i.e.

seq/par/seq, seq/par/par, par/par/seq, and par/par/par.

Notice that the middle "par" means the simultaneous application of all rules in the same matrix. The left "seq" or "par" refers to the horizontal derivations, and the right "seq" or "par" refers to the vertical derivations. The method introduced in CHAPTER III (Matrix Grammars) serve as mode seq/par/seq. In other words, the horizontal derivations are sequential, then in the vertical derivations, all the rules in the same matrix are applied in parallel (simultaneously) but each rule in each column is applied sequentially.

One may ask, what if we apply each of these vertical derivations in parallel. It would be interesting to explore such sequential/parallel complexities.

In addition to the above, the following research is also planned:

(1) The relation between regular-like expressions defined in Chapter V and various classes of sequential/parallel matrix languages. For instance, what expressions characterize the (1:1)ML?

(2) What is the relation between matrix array languages and isotonic array languages? Recall that both sides of production rules in an isotonic array grammar have the same shape.
(3) The machines introduced in Chapter IV are basically non-deterministic. How about a deterministic model for sequential/parallel matrix languages? In string case, it is well known that non-deterministic finite state machines are equivalent to deterministic finite state machines and that the class of non-deterministic pushdown automata properly contains deterministic pushdown automata. How about kTRCA's? We would like to explore the relation between deterministic and non-deterministic machines for various (i:j)kTRCA's, i,j=0,1,2,3.
BIBLIOGRAPHY


11. J.E. Hopcroft and J.D. Ullman, "Formal Languages and Their


28 R. A. Smith, Two-Dimensional Formal Languages and Pattern Recognition by Cellular Automata, IEEE Symp. on Switching and Automata Theory, 12, (1971), 144-152.


