18. Parsing 2-Dimensional Language

Masaru Tomita

School of Computer Science and Center for Machine Translation
Carnegie Mellon University

Abstract

2-Dimensional Context Free Grammar (2D-CFG) for 2-dimensional input text is introduced and efficient parsing algorithms for 2D-CFG are presented. In 2D-CFG, a grammar rule's right hand side symbols can be placed not only horizontally but also vertically. Terminal symbols in a 2-dimensional input text are combined to form a rectangular *region*, and regions are combined to form a larger region using a 2-dimensional phrase structure rule. The parsing algorithms presented in this chapter are 2D-Earley algorithm and 2D-LR algorithm, which are a 2-dimensionally extended version of Earley's algorithm and the Generalized LR algorithm, respectively.

18.1 Introduction

Existing grammar formalisms and formal language theories, as well as parsing algorithms, deal only with one-dimensional strings. However, 2-dimensional layout information plays an important role in understanding a text. It is especially crucial for such texts as title pages of articles, business cards, announcements and formal letters to be read by an optical character reader (OCR). A number of projects (Akiyama & Masuda, 1986; Inagaki, Kato, Hiroshima, & Sakai, 1984; Kubota, *et al.*, 1984; Wong, Casey, & Wahl, 1982), most notably by Fujisawa *et al.* (1988), try to analyze and utilize the 2-dimensional layout information. Fujisawa *et al.*, unlike others, uses a procedural language called Form Definition Language (FDL) (Higashino, Fujisawa, Nakano, & Eijiri, 1986; Yashiro, *et al.*, 1989) to specify layout rules. On the other hand, in the area of image understanding, several attempts have been also made to define a language to describe 2-dimensional images (Fu, 1977; Watanabe, 1972).

This chapter presents a formalism called 2 Dimensional Context Free Grammar (2D-CFG), and two parsing algorithms to parse 2-dimensional language with 2D-CFG. Unlike all the previous attempts mentioned above, our approach is to extend existing well-studied (one dimensional) grammar formalisms and parsing techniques to handle 2-dimensional language. In the rest of this sec-

M. Tomita (ed.), *Current Issues in Parsing Technology*
© Kluwer Academic Publishers 1991

tion, we informally describe the 2-dimensional context free grammar (2D-CFG) in comparison with the 1-dimensional traditional context free grammar.

Input to the traditional context free grammar is a string, or *sentence*; namely a one-dimensional array of terminal symbols. Input to the 2-dimensional context free grammar, on the other hand, is a rectangular block of symbols, or *text*; namely, a 2-dimensional array of terminal symbols.

In the traditional context free grammar, a non-terminal symbol represents a *phrase*, which is a substring of the original input string. A grammar rule is applied to combine adjoining phrases to form a larger phrase. In the 2-dimensional context free grammar, on the other hand, a non-terminal represents a *region*, which is a rectangular sub-block of the input text. A grammar rule is applied to combine two adjoining regions to form a larger region. Rules like

(1) $A \rightarrow B\ C$

are used to combine horizontally adjacent regions. In addition, rules like

(2) $A \rightarrow \begin{matrix} B \\ C \end{matrix}$

can be used in the 2-dimensional context free grammar to combine vertically adjacent regions.

A region can be represented with a non-terminal symbol and four positional parameters: x, y, X, and Y, which determine the upper-left position and the lower-right position of the rectangle (assuming that the coordinate origin is the upper-left corner of the input text).

Horizontally adjacent regions, (B, x_B, y_B, X_B, Y_B) and (C, x_C, y_C, X_C, Y_C), can be combined only if

- $y_B = y_C$,
- $Y_B = Y_C$, and
- $X_B = x_C$.

The first two conditions say that B and C must have the same vertical position and the same height, and the last condition says that B and C are horizontally adjoining.

Similarly, vertically adjacent regions, B and C, can be combined only if

- $x_B = x_C$,
- $X_B = X_C$, and
- $Y_B = y_C$.

A new region, (A, x_B, y_B, X_C, Y_C), is then formed. Figure 18.1 shows examples of adjacent regions, and Figure 18.2 shows the results of combining them using Rules (2) and (1).

Let G be a 2D-CFG $(G(N), G(S), P_H, P_V, S)$, where

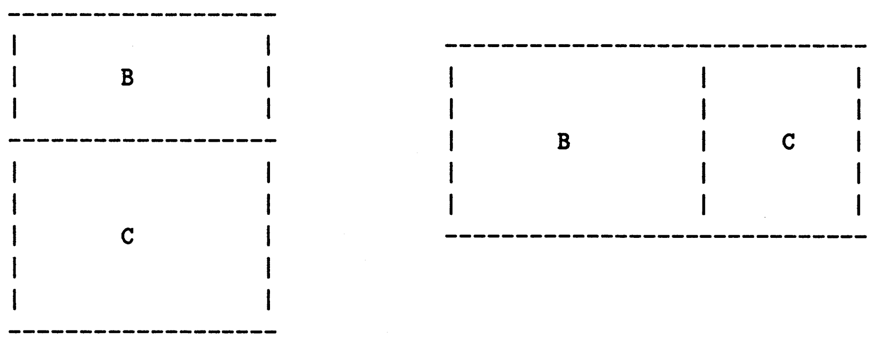

Figure 18.1. Examples of adjacent regions.

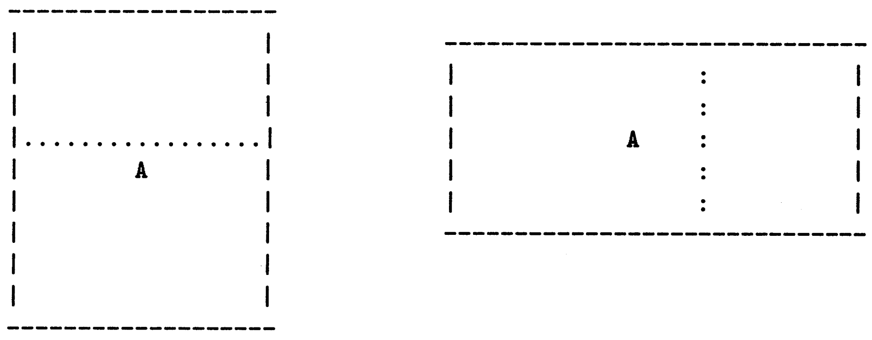

Figure 18.2. After applying Rule (2) and (1), respectively.

N: a set of non-terminal symbols

G(S): is a set of terminal symbols

P_H: a set of horizontal production rules

P_V: a set of vertical production rules

S: start symbol

Let LEFT(p) be the left hand side symbol of p. Let RIGHT(p, i) be the i-th right hand side symbol of p. Without loss of generality, we assume each rule in P_H is either in the form of

$$A \rightarrow B\ C \text{ or } A \rightarrow b$$

and each rule in P_V is in the form of

$$A \rightarrow \begin{array}{c} B \\ C \end{array}$$

Where A,B,C $\in$ N and b $\in$ G(S). This form of grammar is called *2-dimensional Chomsky Normal Form (2D-CNF)*, and an arbitrary 2D-CFG can be converted into 2D-CNF. The conversion algorithm is very similar to the standard CNF conversion algorithm, and we do not describe the algorithm in this chapter.

The subsequent two sections present two efficient 2D parsing algorithms, 2D-Earley and 2D-LR, based on Earley's algorithm (1970) and the Generalized LR algorithm (Tomita, 1985, 1987), respectively.

18.2 The 2D-Earley parsing algorithm

Input:

2D-CFG G = (G(N), G(S), P_H, P_V, S) and an input text

$$
\begin{array}{l}
a_{11}\ a_{21} \cdots a_{n1} \\
a_{12}\ a_{22} \cdots a_{n2} \\
\cdots \\
a_{1m}\ a_{2m} \cdots a_{nm}
\end{array}
$$

where $a_{ij} \in$ G(S).

Output:

A parse table

$$
\begin{array}{l}
I_{00}\ I_{10} \cdots I_{n0} \\
I_{01}\ I_{11} \cdots I_{n1} \\
\cdots \\
I_{0m}\ I_{1m} \cdots I_{nm}
\end{array}
$$

I_{ij} is a set of *items*, and each item is (p, d, x, y, X, Y), where p is a rule in P_H or P_V, d is an integer to represent its dot position ($0 \leq d \leq |p|$, where |p| represents the length of p's left hand side). The integers x and y represent the item's origin (x,y) or the upper-left corner of the region being constructed by the item. The integers X and Y represent its perspective lower-right corner, and the parser's horizontal (vertical) position should never exceed X (Y) until the item is completed.

Method:

For each p $\in P_H \cup P_V$ such that LEFT(p) = S, add an item (p, 0, 0, 0, n, m) to I_{00}.

For each item (p, d, x, y, X, Y) in I_{ij},

If d = |p|, do COMPLETOR

If RIGHT(p, d+1) $\in$ G(N), do PREDICTOR

If RIGHT(p, d+1) $\in$ G(S), do SHIFTER

PREDICTOR:

For all q $\in P_H \wedge P_V$ such that LEFT(q) = RIGHT(p, d+1), add an item (q, 0, i, j, X, Y) to $\mathbf{I}_{ij}$.

SHIFTER:

If $a_{i+1,\,j+1}$ = RIGHT(p, d+1), and if $i<X \wedge j<Y$, then add an item (p, d+1, i, j, X, j+1) to $\mathbf{I}_{i+1,\,j}$.

COMPLETOR:

For all item (p', d', x', y', X', Y') in $\mathbf{I}_{xy}$ such that RIGHT(p', d'+1) = LEFT(p), do the following:

- **Case 1.** $p\in P_H \wedge p'\in P_H$ — Add an item (p', d'+1, x', y' X', Y) to $\mathbf{I}_{ij}$, if $Y'=Y \vee d'=0$.
- **Case 2.** $p\in P_V \wedge p'\in P_H$ — Add an item (p', d'+1, x', y' X', Y) to $\mathbf{I}_{Xy}$, if $Y'=Y \vee d'=0$.
- **Case 3.** $p\in P_H \wedge p'\in P_V$ — Add an item (p', d'+1, x', y' X, Y') to $\mathbf{I}_{xY}$, if $X'=X \vee d'=0$.
- **Case 4.** $p\in P_V \wedge p'\in P_V$ — Add an item (p', d'+1, x', y' X, Y') to $\mathbf{I}_{ij}$, if $X'=X \vee d'=0$.

18.3 The 2D-LR parsing algorithm

A 2D-LR(0) parsing table consists of three parts: ACTION, GOTO-RIGHT and GOTO-DOWN. Table 18.2 is a 2D-LR(0) table obtained from the grammar in Table 18.1.

As in Standard LR parsing, the runtime parser performs shift-reduce parsing with a stack guided by this 2D-LR table. Unlike standard LR(0), however, each item in the stack is represented as (s, x, y, X, Y), where s is an LR state number, and (x,y) represents the current position in the input text. X and Y represent right and lower limits, respectively, and no positions beyond these limits should never be explored until this state is popped off from the stack.

Initially the stack has an item (0, 0, 0, n, m), where n and m are the number of columns and rows in the input text, respectively.

Now let the current elements in the stack be

$\ldots$ — $(s_3, x_3, y_3, X_3, Y_3)$ — B_2 — $(s_2, x_2, y_2, X_2, Y_2)$ — B_1 — $(s_1, x_1, y_1, X_1, Y_1)$

```
         .            |      .            |        .
    S --> A A 0,0,2,2 | B --> b   0,0,2,1 | B --> b   1,0,2,1
                      |        .          |
          .           | S --> A A 0,0,2,2 |          .
    A --> B   0,0,2,2 |                   | S --> A A 0,0,2,2
0         C           |      .            |
                      | A --> B   1,0,2,2 |
         .            |       C           |
    B --> b   0,0,2,2 |      .            |
                      | B --> b   1,0,2,2 |
    ------------------b-------------------b-------------------
                      |        .          |        .
    A --> .B  0,0,1,2 | C --> c   0,1,1,2 | C --> d   1,1,2,2
           C          |                   |
                      | A --> .B  1,0,2,2 |
         .            |        C          |
1   C --> c   0,1,1,2 |      .            |
         .            | C --> c   1,1,2,2 |
    C --> d   0,1,1,2 |      .            |
                      | C --> d   1,1,2,2 |
                      |                   |
    ------------------c-------------------d-------------------
                      |                   |
    A --> B   0,0,1,2 | A --> B   1,0,2,2 |
         .C           |      .C           |
                      |                   |
                      |                   |
2                     |                   |
                      |                   |
                      |                   |
                      |                   |
                      |                   |
             0                   1                   2
```

Figure 18.3. An example of 2D Earley parsing.

where the right most element is the top of the stack. Also assume that the current input symbol a_{ij} is b, where $i = x_1+1$ and $j = y_1+1$. According to the parsing table, we perform SHIFT, REDUCE or ACCEPT.

SHIFT:

If ACTION(s_1, b) = sh s_0, then if $x_1 < X_1 \wedge y_1 < Y_1$, push b and ($s_0$, x_1+1, y_1, X_1, y_1+1) onto the stack.

Table 18.1. Example grammar and text.

Grammar Rules								Text	
(1)	S	$\rightarrow$	A A	(4)	C	$\rightarrow$	c	b	b
								c	d
(2)	A	$\rightarrow$	B	(5)	C	$\rightarrow$	d		
			C						
(3)	B	$\rightarrow$	b						

Table 18.2. A 2D-LR parsing table.

ST	Action				Goto-Right				Goto-Down			
	b	c	d	\$	S	A	B	C	S	A	B	C
0	sh3				8	1					4	
1	sh3					2					4	
2	re1	re1	re1	re1								
3	re3	re3	re3	re3								
4		sh6	sh7									5
5	re2	re2	re2	re2								
6	re4	re4	re4	re4								
7	re5	re5	re5	re5								
8				acc								

REDUCE:

If ACTION(s_1, b) = re p, then let k be |p|+1 and do the following:

- **Case 1**. $p \in P_H$ and GOTO-RIGHT(s_k, LEFT(p)) = s_0 — If $Y_{k-1} = Y_1$ then pop 2*—p— elements from the stack, and push LEFT(p) and (s_0, x_1, y_1, X_k, Y_1).
- **Case 2**. $p \in P_H$ and GOTO-DOWN(s_k, LEFT(p)) = s_0 — If $Y_{k-1} = Y_1$ then pop 2*—p— elements from the stack, and push LEFT(p) and (s_0, x_k, Y_1, x_1, Y_k).
- **Case 3**. $p \in P_V$ and GOTO-RIGHT(s_k, LEFT(p)) = s_0 — If $X_{k-1} = X_1$ then pop 2*—p— elements from the stack, and push LEFT(p) and (s_0, X_1, y_k, X_k, y_1).
- **Case 4**. $p \in P_V$ and GOTO-DOWN(s_k, LEFT(p)) = s_0 — If $X_{k-1} = X_1$ then pop 2*—p— elements from the stack, and push LEFT(p) and (s_0, x_1, y_1, X_1, Y_k).

Figure 18.4 shows an example trace of 2D-LR parsing with the grammar in Table 18.1.

```
(0,0,0,2,2)
(0,0,0,2,2)  b  (3,0,1,2,1)
(0,0,0,2,2)  B  (4,0,1,1,2)
(0,0,0,2,2)  B  (4,0,1,1,2)  c  (6,1,1,1,2)
(0,0,0,2,2)  B  (4,0,1,1,2)  C  (5,0,2,2,2)
(0,0,0,2,2)  A  (1,1,0,2,2)
(0,0,0,2,2)  A  (1,1,0,2,2)  b  (3,2,0,2,1)
(0,0,0,2,2)  A  (1,1,0,2,2)  B  (4,1,1,2,2)
(0,0,0,2,2)  A  (1,1,0,2,2)  B  (4,1,1,2,2)  d  (7,2,1,2,2)
(0,0,0,2,2)  A  (1,1,0,2,2)  B  (4,1,1,2,2)  C  (5,1,2,2,2)
(0,0,0,2,2)  A  (1,1,0,2,2)  A  (2,2,0,2,2)
(0,0,0,2,2)  S  (8,2,0,2,2)
```

Figure 18.4. Example trace of 2D-LR parsing.

The example given in this section is totally deterministic. In general, however, a 2D-LR table may have multiple entries, or both GOTO-DOWN and GOTO-RIGHT may be defined from an identical state with an identical symbol. Such nondeterminism can be also handled efficiently using a *graph-structured stack* as in Generalized LR parsing (Tomita, 1985, 1987).

18.4 More interesting 2D grammars

This section presents a couple of more interesting example grammars and texts. Example Grammar I (see Table 18.3 and Figure 18.5) generates nested rectangles of b's and c's, one after the other. In the grammar, B1 represents vertical bars (sequences) of b's, and B2 represents horizontal bars of b's. Similarly, C1 and C2 represent vertical and horizontal bars of c's, respectively. A1 then represents rectangles surrounded by c's. A2 represents rectangles surrounded by c's which are sandwiched by two vertical bars of b's. A3 further sandwiches A2 with two horizontal b bars, representing rectangles surrounded by b's. Similarly, A4 sandwiches A3 with two vertical c bars, and A1 further sandwiches A4 with two horizontal c bars, representing rectangles surrounded by c's.

A similar analysis can be made for Grammar II (see Table 18.4 and Figure 18.6), which generates triangles of b's and c's.

Grammar III (see Figure 18.7) generates all rectangles of a's which have exactly 2 b's somewhere in them. Xn represents horizontal lines of a's with n b's. Thus, X0, X1 and X2 represent lines of a's, keeping track of how many b's inside. Yn then combines those lines vertically, keeping track of how many a's have been seen thus far (n being the number of b's). Therefore, Y2 contains exactly two b's.

Table 18.3. Example Grammar I.

START	$\rightarrow$	A1						
A1	$\rightarrow$	c	B1	$\rightarrow$	b	C1	$\rightarrow$	c
A1	$\rightarrow$	C2 A4 C2	B1	$\rightarrow$	b B1 b	C1	$\rightarrow$	c C1 c
A2	$\rightarrow$	B1 A1 B1	B2	$\rightarrow$	b	C2	$\rightarrow$	c
A3	$\rightarrow$	B2 A2 B2	B2	$\rightarrow$	b B2 b	C2	$\rightarrow$	c C2 c
A4	$\rightarrow$	C1 A3 C1						

```
                                 ccccccccccccc
                                 cbbbbbbbbbbbc
                     ccccccccc   cbcccccccccbc
            bbbbbbb  cbbbbbbbc   cbcbbbbbbbcbc
      ccccc bcccccb  cbcccccbc   cbcbcccccbcbc
  bbb cbbbc bcbbbcb  cbcbbbcbc   cbcbcbbbcbcbc
c bcb cbcbc bcbcbcb  cbcbcbcbc   cbcbcbcbcbcbc
  bbb cbbbc bcbbbcb  cbcbbbcbc   cbcbcbbbcbcbc
      ccccc bcccccb  cbcccccbc   cbcbcccccbcbc
            bbbbbbb  cbbbbbbbc   cbcbbbbbbbcbc
                     ccccccccc   cbcccccccccbc
                                 cbbbbbbbbbbbc
                                 ccccccccccccc
```

Figure 18.5. Example text for Grammar I.

18.5 Formal property of 2D-CFG

Very little is known about 2D-CFG's formal/mathematical properties. Each column or row of a text forms a string. It is easy to show that such strings can get as complex as context-free. Interestingly enough, however, the author recently found that such strings can get even more complex than context-free. The grammar in Table 18.6 will generate texts like those in Figure 18.8 whose rows form a context-sensitive language, $a^n b^n c^n$. In the grammar, it is easy to see from the rules with A1, A2 and A3 that the non-terminal, A, represents a square of a's. Similarly, the non-terminal B represents a square of b's and C represents a square of c's. When we combine these three squares using the

Table 18.4. Example Grammar II.

START	→	A1	A1	→	c	B1	→	b	C1	→	c
			A1	→	A4 C2	B1	→	b B1	C1	→	c C1
			A2	→	A1 B1	B2	→	b	C2	→	c
			A3	→	B2 A2	B2	→	b B2	C2	→	c C2
			A4	→	C1 A3						

```
                                               cbbbbbbbbbbbb
                                               ccbbbbbbbbbbb
                               cbbbbbbbb       cccbbbbbbbbbb
               cbbbbbb         ccbbbbbbb       ccccbbbbbbbbb
        cbbbb  ccbbbbb         cccbbbbbb       cccccbbbbbbbb
   cbb  ccbbb  cccbbbb         ccccbbbbb       ccccccbbbbbbb
c  ccb  cccbb  ccccbbb         cccccbbbb       cccccccbbbbbb
   ccc  ccccb  cccccbb         ccccccbbb       ccccccccbbbbb
        ccccc  ccccccb         cccccccbb       cccccccccbbbb
               ccccccc         ccccccccb       ccccccccccbbb
                               ccccccccc       cccccccccccbb
                                               ccccccccccccb
                                               ccccccccccccc
```

Figure 18.6. Example text for Grammar II.

```
aaaaaaaaaaaaa       aaa       aaaaaaa
aaabaaaaaaaaa       aaa       aaaabaa
aaaaaaaaaaaaa       bba       aabaaaa
aaaaaaaaaaaab       aaa
aaaaaaaaaaaaa       aaa
                    aaa
                    aaa
                    aaa
```

Figure 18.7. Example text for Grammar III.

last rule, all the three squares are forced to have the same height due to the adjoining constraint. If all three squares have the same height, say n, then all

Table 18.5. Example Grammar III.

START	→	Y2	X0	→	[empty]	Y0	→	[empty]
			X0	→	X0 a	Y0	→	Y0
								X0
			X1	→	X0 b	Y1	→	Y0
								X1
			X1	→	X1 a	Y1	→	Y1
								X0
			X2	→	X1 b	Y2	→	Y0
								X2
			X2	→	X2 a	Y2	→	Y1
								X1
						Y2	→	Y2
								X0

Table 18.6. Example Grammar IV.

S	→	A B C									
A	→	a	A	→	A3	A1	→	a	A1	→	A1
					A2						a
A2	→	a	A2	→	A2 a	A3	→	A A1			
B	→	b	B	→	B	B1	→	b	B1	→	B1
					B2						b
B2	→	b	B2	→	B2 b	B3	→	B B1			
C	→	c	C	→	C	C1	→	c	C1	→	C1
					C2						c
C2	→	c	C2	→	C2 c	C3	→	C C1			

the three squares must have the same width, n. Thus, each row string of S is in the language, $a^n b^n c^n$.

Exactly how complex row (or column) strings can get is unknown. There appear to be many other interesting problems like this behind the theory of 2D-CFG.

```
                    aaaaaaaaabbbbbbbbbccccccccc
                    aaaaaaaaabbbbbbbbbccccccccc
                    aaaaaaaaabbbbbbbbbccccccccc
                    aaaaaaaaabbbbbbbbbccccccccc
aaabbbccc           aaaaaaaaabbbbbbbbbccccccccc
aaabbbccc           aaaaaaaaabbbbbbbbbccccccccc
aaabbbccc           aaaaaaaaabbbbbbbbbccccccccc
                    aaaaaaaaabbbbbbbbbccccccccc
                    aaaaaaaaabbbbbbbbbccccccccc
```

Figure 18.8. Example text for Grammar IV.

18.6 Concluding remarks

In this chapter, 2D-CFG, 2-dimensional context free grammar, has been introduced, and two efficient parsing algorithms for 2D-CFG have been presented. Traditional one-dimension context free grammars are well studied and well understood (e.g., Aho & Ullman, 1972), and Many of their theorems and techniques might be extended and adopted for 2D-CFG, as we have done for Earley's algorithm and LR parsing in this chapter.

Acknowledgments

This research was supported by National Science Foundation under the contract IRI-8858085.

References

Aho, A. V., & Ullman, J. D. (1972). *The theory of parsing, translation and compiling.* Englewood Cliffs, NJ: Prentice-Hall.

Akiyama, T., & Masuda, I. (1986). A method of document-image segmentation based on projection profiles, stroke density and circumscribed rectangles. *Trans. IECE, J69-D,* 1187–1196.

Earley, J. (1970). An efficient context-free parsing algorithm. *Communications of ACM, 6,* 94–102.

Fu, K. S. (1977). *Syntactic pattern recognition.* Springer-Verlag.

Fujisawa, H., *et al.* (1988). Document analysis and decomposition method for multimedia contents retrieval. *Proceedings of the Second International Symposium on Interoperable Information Systems* (p. 231).

Higashino, J., Fujisawa, H., Nakano, Y., & Ejiri, M. (1986). A knowledge-based segmentation method for document understanding. *Proceedings of the Eighth International Conference on Pattern Recognition* (pp. 745–748).

Inagaki, K., Kato, T., Hiroshima, T., & Sakai, T. (1984). MACSYM: A hierarchical image processing system for event-driven pattern understanding of documents. *Pattern Recognition*, *17*, 85–108.

Kubota, K., *et al.* (1984). Document understanding system. *Proceedings of the Seventh International Conference on Pattern Recognition* (pp. 612–614).

Tomita, M. (1985). *Efficient parsing for natural language.* Boston, MA: Kluwer.

Tomita, M. (1987). An efficient augmented-context-free parsing algorithm. *Computational Linguistics*, *13*, 31–46.

Watanabe, S. (Ed.) (1972). *Frontiers of pattern recognition.* Academic Press.

Wong, K., Casey, R., Wahl, F. (1982). Document analysis system. *IBM J. Research and Development*, *26*, 647–656.

Yashiro, H., *et al.* (1989). A new method of document structure extraction. *Proceedings of the International Workshop on Industrial Applications of Machine Intelligence and Vision (MIV-89)* (p. 282).