Parsing 2-Dimensional Language

Masaru Tomita
Computer Science Department
and
Center for Machine Translation
Carnegie-Mellon University
Pittsburgh, PA 15213[1]

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 paper are the 2D-Earley algorithm and 2D-LR algorithm, which are 2-dimensionally extended versions of Earley's algorithm and the LR(0) algorithm, respectively.

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 [11, 6, 7, 2], most notably by Fujisawa *et al.* [4], try to analyze and utilize the 2-dimensional layout information. Fujisawa *et al.*, unlike others, uses a procedural language called Form Definition Language (FDL) [5, 12] 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 [3, 10].

This paper 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 section, 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]This research was supported by the National Science Foundation under contract IRI-8858085.

```
A -->  B  C                    (1)
```

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

```
        B
A -->                          (2)
        C
```

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 4 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 1-1 shows examples of adjacent regions, and figure 1-2 shows the results of combining them using rules (2) and (1).

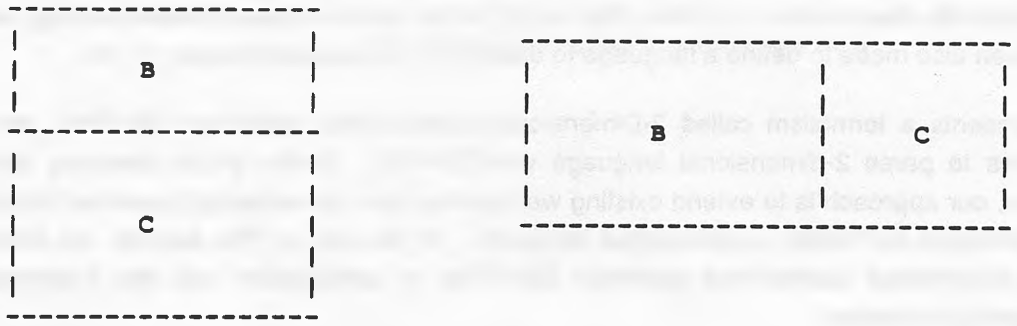

Figure 1-1: Examples of Adjacent Regions

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

N: a set of non-terminal symbols
Σ: 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 --> B C    or    A --> b
```

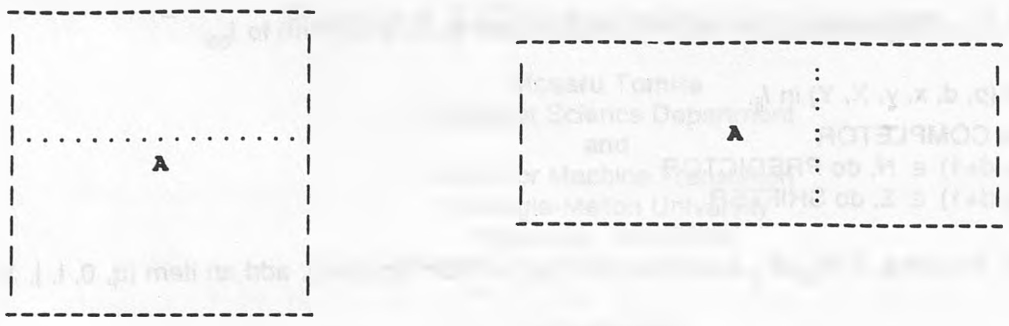

Figure 1-2: After applying rule (2) and (1), respectively

and each rule in P_V is in the form of

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

Where A,B,C $\in$ N and b $\in$ Σ. 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 paper.

The subsequent two sections present two efficient 2D parsing algorithms: 2D-Earley and 2D-LR.

2. The 2D-Earley Parsing Algorithm

Input:

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

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

where $a_{ij} \in \Sigma$.

Output:

A parse table

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

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 ($0 \le d \le |p|$, 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 $RIGHT(p, d+1) \in N$, do PREDICTOR
    If $RIGHT(p, d+1) \in \Sigma$, 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 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 $I_{i+1,j}$.

COMPLETOR: For all items (p', d', x', y', X', Y') in 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 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 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 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 I_{ij}, if $X'=X \vee d'=0$.

```
(1) S --> A A        (3) B --> b              b b
                                              c d
(2) A --> B          (4) C --> c
          C
                     (5) C --> d
```

Figure 2-1: Example Grammar and Text

```
          .                 |      .                 |        .
     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 2-2: An Example of 2D-Earley Parsing

3. The 2D-LR Parsing Algorithm

A 2D-LR(0) parsing table consists of three parts: ACTION, GOTO-RIGHT and GOTO-DOWN. Figure 3-1 is a 2D-LR(0) table obtained from the grammar in Figure 2-1.

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								

Figure 3-1: A 2D-LR Parsing Table

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 ever be explored until this state is popped off 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

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

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.

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 3-2 shows an example trace of 2D-LR parsing with the grammar in Figure 2-1.

```
(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 3-2: Example Trace of 2D-LR Parsing

4. More Interesting 2D Grammars

This section presents a couple of more interesting example grammars and texts. Example Grammar I 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, which generates triangles of b's and c's.

Grammar III 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 are 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.

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 also be handled efficiently using a *graph-structured stack* as in Generalized LR Parsing [8, 9].

```
A1 --> c                     B1 --> b               C1 --> c

A2 --> B1 A1 B1                     b                      c
                             B1 --> B1              C1 --> C1
                                    b                      c
       B2
A3 --> A2
       B2                    B2 --> b               C2 --> c

A4 --> C1 A3 C1              B2 --> b B2 b          C2 --> c C2 c

       C2
A1 --> A4                    START --> A1
       C2

                                                     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 4-1: Example Grammar I

```
A1 --> c                     B1 --> b               C1 --> c

A2 --> A1 B1                        b                      c
                             B1 --> B1              C1 --> C1
       B2
A3 --> A2                    B2 --> b               C2 --> c

A4 --> C1 A3                 B2 --> b B2            C2 --> c C2

A1 --> A4                    START --> A1
       C2

                                                     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 4-2: Example Grammar II

```
X0 --> [empty]            Y0 --> [empty]            Y2 --> Y0
                                                           X2
X0 --> X0 a               Y0 --> Y0
                                 X0                 Y2 --> Y1
X1 --> X0 b                                                X1
                          Y1 --> Y0
X1 --> X1 a                      X1                 Y2 --> Y2
                                                           X0
X2 --> X1 b               Y1 --> Y1
                                 X0                 START --> Y2
X2 --> X2 a

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

Figure 4-3: Example Grammar III

5. Concluding Remarks

In this paper, 2D-CFG, 2-dimensional context-free grammar, has been introduced, and two efficient parsing algorithms for 2D-CFG have been presented. Traditional one-dimensional context-free grammars are well studied and well understood (e.g. [1]), and many of their theorems and techniques might be extended and adopted for 2D-CFG, as we have done in this paper for Earley's algorithm and LR parsing.

References

[1] Aho, A. V. and Ullman, J. D.
The Theory of Parsing, Translation and Compiling.
Prentice-Hall, Englewood Cliffs, N. J., 1972.

[2] Akiyama, T. and Masuda, I.
A Method of Document-Image Segmentation Based on Projection Profiles, Stroke Density and Circumscribed Rectangles.
Trans. IECE J69-D(8):1187-1196, 1986.

[3] K. S. Fu.
Syntactic Pattern Recognition.
Springer-Verlag, 1977.

[4] Fujisawa, H. et al.
Document Analysis and Decomposition Method for Multimedia Contents Retrieval.
Proc. 2nd International Symposium on Interoperable Information Systems :231, 1988.

[5] Higashino, J., Fujisawa, H., Nakano, Y. and Ejiri, M.
A Knowledge-Based Segmentation Method for Document Understanding.
Proc. 8th Int. Conf. Pattern Recognition :745-748, Oct., 1986.

[6] Inagaki, K., Kato, T., Hiroshima, T. and Sakai, T.
MACSYM: A Hierarchical Image Processing System for Event-Driven Pattern Understanding of Documents.
Pattern Recognition 17(1):85-108, 1984.

[7] Kubota, K. et al.
Document Understanding System.
In *Proc. 7th Int. Conf. Pattern Recognition*, pages 612-614. , 1984.

[8] Tomita, M.
Efficient Parsing for Natural Language.
Kluwer Academic Publishers, Boston, MA, 1985.

[9] Tomita, M.
An Efficient Augmented-Context-Free Parsing Algorithm.
Computational Linguistics 13(1-2):31-46, January-June, 1987.

[10] Watanabe, S. (ed.).
Frontiers of Pattern Recognition.
Academic Press, 1972.

[11] Wong, K., Casey, R. and Wahl, F.
Document Analysis System.
IBM J. Research and Development 26(6):647-656, 1982.

[12] Yashiro, H. et.al.
A New Method of Document Structure Extraction.
In *International Workshop on Industrial Applications of Machine Intelligence and Vision (MIV-89)*, pages 282. , April, 1989.