3D-Plex Grammars*

WEI-CHUNG LIN
and
KING-SUN FU

School of Electrical Engineering, Purdue University, West Lafayette, Indiana 47907

Communicated by K. S. Fu

ABSTRACT

The plex grammar proposed by Feder is extended to describe 3D structures. Several useful algorithms, including a derivation algorithm, an object construction algorithm, an connectivity table construction algorithm, and a parsing algorithm, are presented. Examples are given to illustrate the algorithms.

I. INTRODUCTION

Formal language theory deals with mathematical models of languages and the systems used in language generation and processing. In conventional language derivation, each terminal and nonterminal symbol may appear in a string with a symbol to its left or right. The only relation between symbols is the concatenation, that is, each symbol can be connected only in a "head-to-tail" manner. By allowing more versatile interconnection capabilities for the symbols, special grammars such as plex grammar and PDL grammar, and higher-dimensional grammars such as array grammar, web grammar, graph grammar, tree grammar, and shape grammar have been proposed [1].

Although the extensions have increased the descriptive power of grammatical approaches, each symbol can only be "visualized" as a two-dimensional structure (or projection of a three-dimensional structure) and the connections among the symbols performed in a plane. Based on the theoretical framework of the plex grammar proposed by Feder [2], Lin and Fu have proposed the 3D-plex grammars for describing 3D structures [3]. The basic idea is to consider each terminal or nonterminal symbol as a primitive or composite surface having an

*This work was supported by NSF Grant ECS 81-19886.

52 Vanderbilt Ave., New York, NY 10017 0020-0255/84/$03.00

arbitrary number n of attaching curves for joining to other surfaces. A structure of this type is called an n-attaching-curve entity (NACE). Structures formed by interconnecting such entities are called plex structures. Languages called plex languages can be formed from sets of plex structures. A grammar used for the specification of a plex language is called a plex grammar. In this paper, we will review the 3D-plex grammar and present several useful algorithms associated with it.

II. DEFINITION OF 3D-PLEX GRAMMAR

A 3D-plex grammar can be represented by a sextuple $G_p = (N, \Sigma, P, S, I, i_0)$ where

N is a finite nonempty set of NACEs called the nonterminals;
Σ is a finite nonempty set of NACEs called the terminals;
$N \cap \Sigma = \varnothing$;
P is a finite set of productions or rewriting rules;
$S \in N$ is a special NACE called the initial NACE;
I is a finite set of symbols called identifiers;
$I \cap (N \cup \Sigma) = \varnothing$;
$i_0 \in I$ is a special identifier called the null identifier.

The symbols of I are used to identify the attaching curves of NACEs. Every attaching curve of a NACE has an identifier in I, and no two attaching curves of the same NACE have the same identifier. The null identifier i_0 serves as a place marker and is not associated with any attaching curve. Interconnections of NACEs can explicitly be made through the specified attaching curves which are equivalent; "imaginary" connections using the null identifier are not permitted.

Several types of plex grammars may be defined by imposing restrictions on the production rules. An unrestricted 3D-plex grammar has productions of the form [2]

$$\psi \Gamma_\psi \Delta_\psi \rightarrow \omega \Gamma_\omega \Delta_\omega,$$

where ψ is called the left-side component list, ω the right-side component list, Γ_ψ the left-side intersection list, Γ_ω the right-side intersection list, Δ_ψ the left-side tie-curve list, and Δ_ω the right-side tie-curve list.

The component lists are strings of the form $\psi = a_1 a_2 \cdots a_i \cdots a_m$ and $\omega = b_1 b_2 \cdots b_j \cdots b_n$, where a_i and b_j are single NACEs called components; ψ- and ω-lists provide an ordering for the groups of connected NACEs that comprise the respective component lists. The connection of attaching curves of two or more NACEs forms an intersection, and Γ_ψ and Γ_ω specify the way in

which the NACEs of their respective component lists interconnect. The intersection lists, which are unordered, are divided into fields that specify which attaching curves of which NACEs connect at each intersection. Exactly one field is required per intersection. The lengths of the fields for the left and right sides of a production are given by $l(\psi)$ and $l(\omega)$, respectively, where l denotes the length of its string argument. An entry i_k in the jth position of a field indicates that the attaching curve i_k of the jth element (i.e. NACE) of the component list preceding Γ connects at the intersection associated with the field. If the jth NACE is not involved in that particular joint, the null identifier appears in the jth position of the field.

The component and intersection lists $\psi\Gamma_\psi$ and $\omega\Gamma_\omega$, when taken as pairs, define the structure involved in a rewriting rule. These structures attach to the remainder of the plex at a finite number of intersections called tie curves. The tie-curve lists Δ_ψ and Δ_ω give the correspondence between these external connections for the left and right sides of a production. The tie-curve lists are also divided into fields, with exactly one field specifying each tie curve. Since the number of tie curves for the left and right sides of a production must be the same, the number of fields in each tie-curve list is the same. Moreover, the tie-curve lists are ordered, with the kth field on the left corresponding to the kth field on the right.

The general definition of an unrestricted 3D-plex grammar given above is too broad to be of much practical use. In this paper, we use only the context-free 3D-plex grammar. A context-free 3D-plex grammar has productions of the form

$$A\Delta_A \rightarrow \chi\Gamma_\chi\Delta_\chi$$

where A is a single NACE and χ is a arbitrary list of NACEs. It is assumed that χ is not null. The rule states that a single NACE A, appearing in any context, can be replaced by the subplex given by $\chi\Gamma_\chi$. The classification of 3D-plex languages is the same as that of plex languages discussed in [2].

Plex grammar is a special case of attributed grammar. In the production rule $A\Delta_A \rightarrow \chi\Gamma_\chi\Delta_\chi$, the part $A \rightarrow \chi$ is the same as that of a conventional string grammar, while Δ_A, Γ_χ, and Δ_χ are semantic parts. The language generated by the syntactic part of a context-free 3D-plex grammar is still context-free. It is noted that the nonterminal symbols can have different boundaries, but the number and shapes of tie curves always remain the same. To simplify the notation, if the left-side tie-curve list Δ_A is $\{1,2,\ldots,n\}$, $n \geq 1$, we abbreviate it to $\{n\}$. Several examples have been given in [3,4,5] to illustrate the process of describing a 3D object by 3D-plex grammars. In this paper, we will use the object in Example 1 as a running example to illustrate the related algorithms. In the example, the only spatial transformation allowed for a terminal symbol

(surface primitive) is parallel translation. Positive integers are used as identifiers, with 0 being reserved for the null identifier i_0. In the following description, we will use terminal symbol or primitive surface patch interchangeably.

EXAMPLE 1. The following grammar describes the class of objects shown in Figure 1(a):

$$G_p = (N, \Sigma, P, S, I, i_0),$$

where

$$N = \{S, A, B, C\},$$

$$\Sigma = \{a, b, c, d\},$$

$$S = S,$$

$$I = \{0,1,2,3,4\},$$

$$i_0 = 0,$$

and P consists of the following production rules:

(1) $S(\ ) \rightarrow aA(12)(\ )$
(2) $A(3) \rightarrow dB(21,32,43)(21,32,43)$
(3) $A(3) \rightarrow d(\ )(2,3,4)$
(4) $B(3) \rightarrow cbcC(3400,0230,2003,0302,0021)(0040,0100,4000)$
(5) $C(3) \rightarrow aA(12)(01,12,03)$

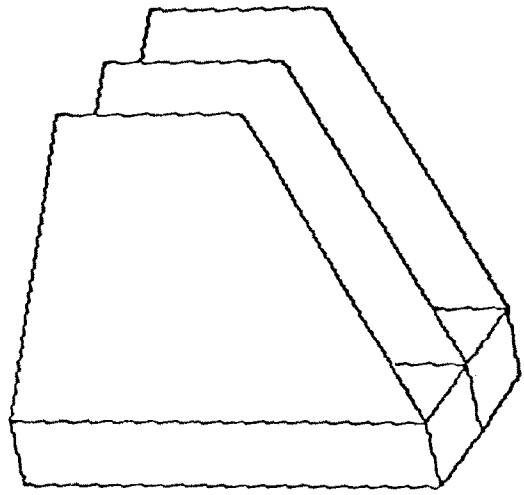
The primitive surface patches are shown graphically in Figure 2. The decomposition of the object is shown in Figure 1(b).

III. ALGORITHMS FOR 3D-PLEX GRAMMARS

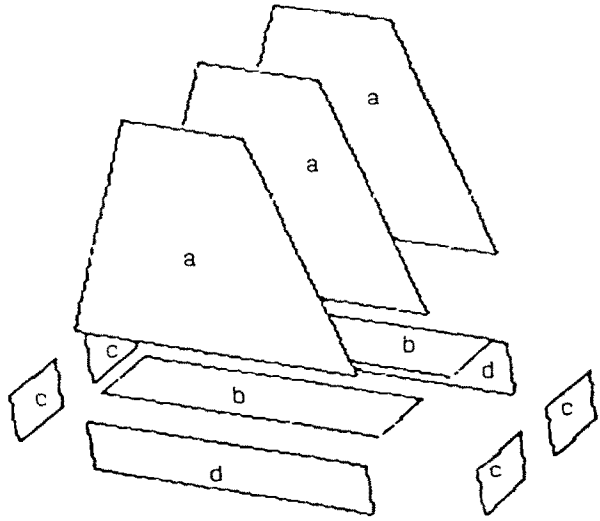
3.1. THE DERIVATION ALGORITHM

The following algorithm explains how a sentence and its associated interconnection matrix are derived from a context-free 3D-plex grammar. The format of the interconnection matrix M_η associated with the sentential form η is shown in Figure 3.

ALGORITHM 3.1. Derivation of a sentence and its associated interconnection matrix.



(a)



(b)

Fig. 1. (a) One instance of the object generated from the grammar in Example 1 and (b) its decomposition.

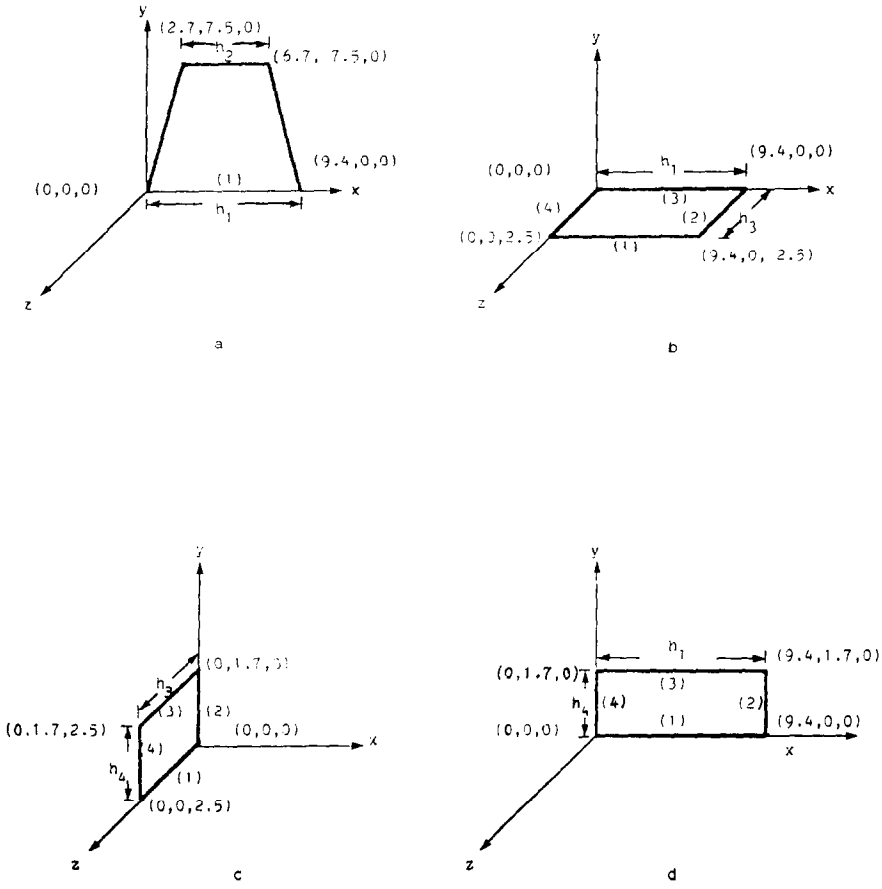


Fig. 2. The primitive surface patches (terminal symbols) of the grammar in Example 1.

$$
\begin{array}{c}
\\
1\\2\\3\\ \vdots \\ m
\end{array}
\begin{array}{c}
\begin{array}{cccc} X_1 & X_2 & \cdots & X_n \end{array}\\
\left[\begin{array}{cccc} & & & \\ & & m_{ij} & \\ & & & \end{array}\right]
\end{array}
$$

Fig. 3. Interconnection matrix M_η where $\eta = X_1 X_2 \cdots X_n$, $X_i \in N \cup \Sigma$ for all $i = 1,2,\ldots,n$. m is the number of interconnections derived, $m_{ij} \in I$.

Input. A context-free 3D-plex grammar $G_p = (N, \Sigma, P, S, I, i_0)$ and a left parse $p_1 p_2 \cdots p_m$.

Output. The sentence and its interconnection matrix generated by applying the left parse.

Method.

(1) Initialize i to 1.

(2) Let the production rule p_1 be

$$S\Delta_S \rightarrow \chi_1 \Gamma_{\chi_1} \Delta_{\chi_1}$$

where $\Delta_S = \Delta_{\chi_1} = \varnothing$. Apply p_1 and obtain the sentential form χ_1 with the initial interconnection matrix from Γ_{χ_1}.

(3) Increase i by 1. If i is greater than m, go to (9).

(4) Let the production rule p_i be

$$A\Delta_A \rightarrow \chi \Gamma_\chi \Delta_\chi$$

where

$\Delta_A = (t_1, t_2, \ldots, t_1)$;
$\chi = X_1 X_2 \cdots X_n$, $X_n \in N \cup \Sigma$ for all $i = 1,2,\ldots,n$;
$\Gamma_\chi = (f_1, f_2, \ldots, f_k)$, $f_q = y_{q_1} y_{q_2} \cdots y_{q_n}$ for all $q = 1,2,\ldots,k$, and exactly two y_{q_i}'s, $1 \leqslant i \leqslant n$, are not null identifiers;
$\Delta_\chi = (g_1, g_2, \ldots, g_1)$, $g_q = z_{q_1} z_{q_2} \cdots z_{q_n}$ for all $q = 1,2,\ldots,l$.

Suppose that before applying production rule p_i, the sentential form is $Y_1 Y_2 \cdots Y_r$, $Y_j \in N \cup \Sigma$ for all $j = 1,2,\ldots,r$, and the interconnection matrix is $[H_1 H_2 \cdots H_s]^T$, where $H_q^T = w_{q_1} w_{q_2} \cdots w_{q_r}$, $w_{q_j} \in I \cup \{i_0\}$, $1 \leqslant q \leqslant s$, $1 \leqslant j \leqslant r$. Suppose that Y_j, $1 \leqslant j \leqslant u-1$, are terminal symbols and $Y_u = A$. Then, after applying production rule p_i, the sentential form becomes $Y_1 Y_2 \cdots Y_{u-1} X_1 X_2 \cdots X_n Y_u Y_{u+1} \cdots Y_r$.

(5) Update the interconnection matrix from the tie-curve list of production rule p_i as follows: For each row $H_q^T = w_{q_1} w_{q_2} \cdots w_{q_r}$, $1 \leqslant q \leqslant s$, in the interconnection matrix of the sentential form, update H_q^T by substituting the q_i's, $1 \leqslant i \leqslant 1$, for w_{q_u} if w_{q_u} is not null. If w_{q_u} is a null identifier, then substitute n null identifiers for w_{q_u} instead.

(6) If $n = 1$, then go to (3).

(7) Create new interconnections from the interconnection list of production rule p_i as follows: For each q, $1 \leqslant q \leqslant k$, add a new row

$$\beta_1 \beta_2 \cdots \beta_{u-1} f_q \beta_u \beta_{u+1} \cdots \beta_r$$

(β_i are null identifiers) to the interconnection matrix.

(8) go to (3).
(9) Delete redundant interconnections in the matrix as follows: If for all non-null identifiers m in column c of row j of the interconnection matrix, there is also an m in column c of row i, then we say that row i contains row j. If row i contains row j, then row j can be deleted from the matrix without loss of information.

EXAMPLE 2. Applying the above algorithm to the grammar in Example 1 and the left parse 1,2,4,5,2,4,5,3, we obtain the leftmost derivation shown in Figure 4 for the 3D-plex describing the scene in Figure 1(a).

3.2. *THE OBJECT-CONSTRUCTION ALGORITHM*

In this section, we present an algorithm which constructs a 3D object by spatially translating its component primitive surface patches. The position of the pivot primitive surface patch in 3D space remains the same as defined and serves as the basis of the construction process. The algorithm first constructs a directed graph from the interconnection matrix and then computes the geometric information for each surface patch (or terminal symbol) in a sentential form. In a directed graph, if there is a directed edge e from a_i to a_j, then a_i and a_j are referred to as the predecessor and successor associated with edge e, respectively.

ALGORITHM 3.2. Construction of a 3D object.

Input.

(1) A sentential form η and its associated interconnection matrix M_η derived from Algorithm 3.1.
(2) The geometric information of the primitive surface patches.
(3) The position of the pivot primitive surface patch in η.

Output. The geometric information of the surface patches (or terminal symbols) in η.

Method.

(1) Let M be the interconnection matrix associated with the terminal string $a_1 a_2 \cdots a_k$ which is obtained by deleting the nonterminal symbols from η. Let $m_{i,j}$ be the element of M in row i and column j where $1 \leqslant i \leqslant l$ and $1 \leqslant j \leqslant k$. Suppose a_{iprm} is the pivot primitive surface patch.
(2) Set icol equal to iprm.
(3) Set iedge equal to zero.
(4) For irow = 1 to l do steps (5)–(9).
(5) If $m_{\text{irow,icol}}$ is not a null identifier, then do steps (6)–(9).

Start symbol = S

Apply production rule 1

we have

a	A
1	2

apply production rule 2

we have

a	d	B
1	3	2
0	2	1
0	4	3

apply production rule 4

we have

a	d	c	b	c	C
1	3	0	1	0	0
0	2	0	0	4	0
0	4	4	0	0	0
0	0	3	4	0	0
0	0	0	2	3	0
0	0	2	0	0	3
0	0	0	3	0	2
0	0	0	0	2	1

apply production rule 5

we have

a	d	c	b	c	a	A
1	3	0	1	0	0	0
0	2	0	0	4	0	0
0	4	4	0	0	0	0
0	0	3	4	0	0	0
0	0	0	2	3	0	0
0	0	2	0	0	0	3
0	0	0	3	0	1	2
0	0	0	0	2	0	1

apply production rule 2

we have

a	d	c	b	c	a	d	B
1	3	0	1	0	0	0	0
0	2	0	0	4	0	0	0
0	4	4	0	0	0	0	0
0	0	3	4	0	0	0	0
0	0	0	2	3	0	0	0
0	0	2	0	0	0	4	3
0	0	0	3	0	1	3	2
0	0	0	0	2	0	2	1

Fig. 4. Leftmost derivation of the object in Figure 1(a).

apply production rule 4

we have

a	*d*	*c*	*b*	*c*	*a*	*d*	*c*	*b*	*c*	*C*
1	3	0	1	0	0	0	0	0	0	0
0	2	0	0	4	0	0	0	0	0	0
0	4	4	0	0	0	0	0	0	0	0
0	0	3	4	0	0	0	0	0	0	0
0	0	0	2	3	0	0	0	0	0	0
0	0	2	0	0	0	4	4	0	0	0
0	0	0	3	0	1	3	0	1	0	0
0	0	0	0	2	0	2	0	0	4	0
0	0	0	0	0	0	0	3	4	0	0
0	0	0	0	0	0	0	0	2	3	0
0	0	0	0	0	0	0	2	0	0	3
0	0	0	0	0	0	0	0	3	0	2
0	0	0	0	0	0	0	0	0	2	1

apply production rule 5

we have

a	*d*	*c*	*b*	*c*	*a*	*d*	*c*	*b*	*c*	*a*	*A*
1	3	0	1	0	0	0	0	0	0	0	0
0	2	0	0	4	0	0	0	0	0	0	0
0	4	4	0	0	0	0	0	0	0	0	0
0	0	3	4	0	0	0	0	0	0	0	0
0	0	0	2	3	0	0	0	0	0	0	0
0	0	2	0	0	0	4	4	0	0	0	0
0	0	0	3	0	1	3	0	1	0	0	0
0	0	0	0	2	0	2	0	0	4	0	0
0	0	0	0	0	0	0	3	4	0	0	0
0	0	0	0	0	0	0	0	2	3	0	0
0	0	0	0	0	0	0	2	0	0	0	3
0	0	0	0	0	0	0	0	3	0	1	2
0	0	0	0	0	0	0	0	0	2	0	1

apply production rule 3

we have

a	*d*	*c*	*b*	*c*	*a*	*d*	*c*	*b*	*c*	*a*	*d*
1	3	0	1	0	0	0	0	0	0	0	0
0	2	0	0	4	0	0	0	0	0	0	0
0	4	4	0	0	0	0	0	0	0	0	0
0	0	3	4	0	0	0	0	0	0	0	0
0	0	0	2	3	0	0	0	0	0	0	0
0	0	2	0	0	0	4	4	0	0	0	0
0	0	0	3	0	1	3	0	1	0	0	0
0	0	0	0	2	0	2	0	0	4	0	0
0	0	0	0	0	0	0	3	4	0	0	0
0	0	0	0	0	0	0	0	2	3	0	0
0	0	0	0	0	0	0	2	0	0	0	4
0	0	0	0	0	0	0	0	3	0	1	3
0	0	0	0	0	0	0	0	0	2	0	2

Fig. 4 continued

(6) Starting from $m_{\text{irow,icol}}$, search in a leftward direction. Suppose the first non-null identifier found is $m_{\text{irow,j1}}$, $1 \leqslant \text{j1} \leqslant \text{icol}$. Then, increment iedge by 1 and construct a directed edge from a_{icol} to a_{j1} with label $(m_{\text{irow,icol}}, m_{\text{irow,j1}})$ is there is no edge between a_{icol} and a_{j1}.
(7) Add iedge to queue 1.
(8) Starting from $m_{\text{irow,icol}}$, search in a rightward direction. Suppose the first non-null identifier found is $m_{\text{irow,j2}}$, $\text{icol} \leqslant \text{j2} \leqslant k$. Then, increment iedge by 1 and construct a directed edge from a_{icol} to a_j with label $(m_{\text{irow,icol}}, m_{\text{irow,j2}})$ if there is no edge between a_{icol} and a_{j2}.
(9) Add iedge to queue 1.
(10) Remove one item from queue 1.
(11) If queue 1 is not empty, then set icol equal to the column number of the symbol which is the successor associated with edge obtained from step (10), and go to (4).
(12) Set ipos equal to iprm and determine the geometric information of a_{ipos} from input.
(13) Add to queue 2 the index of the edge which has a_{ipos} as its predecessor.
(14) Remove one index from queue 2, and find the successor a_s of the edge associated with the index. If queue 2 is empty, then stop.
(15) Compute the geometric information of a_s from the information of its predecessor a_p and the label (l_1, l_2) associated with the directed edge from a_p to a_s if it has not been computed yet.
(16) Set ipos equal to s and go to step (13).

EXAMPLE 3. Figure 5 shows the graph representation for the interconnection matrix in Figure 6, whose generating grammar and primitive surface patches are the same as those of Example 1. The computer printout of the intermediate results is also shown in Figure 6.

3.3. THE CONNECTIVITY-TABLE CONSTRUCTION ALGORITHM

ALGORITHM 3.3. Construction of a table describing the connectivity relationships among the primitive surface patches.

Input. A context-free 3D-plex grammar $G_p = (N, \Sigma, P, S, I, i_0)$.

Output. A table which describes the possible connectivity information of the terminal symbols (primitive surface patches).

Method.

(1) Consider every production rule in P with nonempty right-side join list. Let the right-side component list of the production rule being considered be $X_1 X_2 \cdots X_n$. For each field $f = y_1 y_2 \cdots y_n$ in the join list, with $y \in I$ for all $i = 1, 2, \ldots, n$, if y_p and y_q, $1 \leqslant p, q \leqslant n$, $p \neq q$, are not null identifiers, add the item $X_p y_p \sim X_q y_q$ to the connectivity table provided that it is a new entry. Note that the order of $X_p y_p$ and $X_q y_q$ is not important.

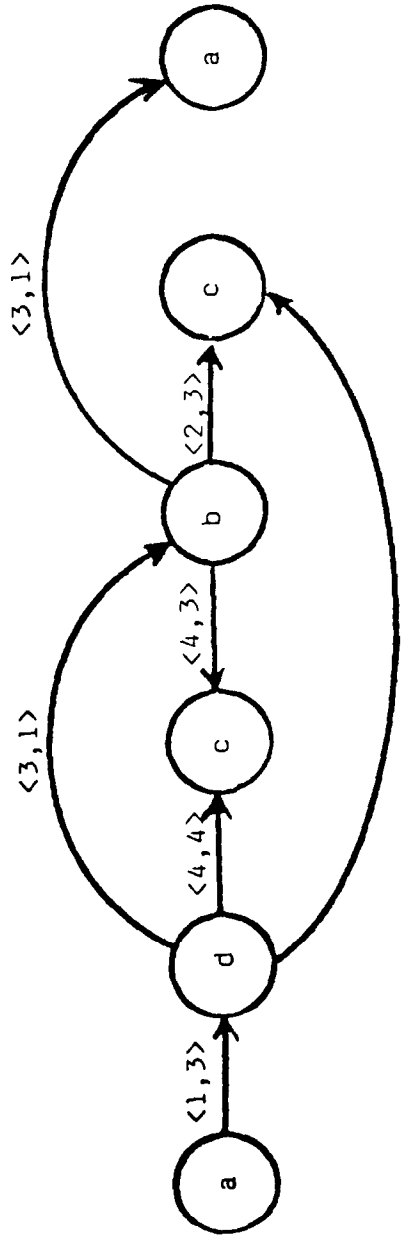


Fig. 5. Graph representation for the interconnection matrix in Figure 6.

The sentential form and interconnection matrix is

a	*d*	*c*	*b*	*c*	*a*
1	3	0	1	0	0
0	2	0	0	4	0
0	4	4	0	0	0
0	0	3	4	0	0
0	0	0	2	3	0
0	0	2	0	0	0
0	0	0	3	0	1
0	0	0	0	2	0

Data for primitive surface patch a

vertex 1:	(	0.	0.	0.)
vertex 2:	(	9.40	0.	0.)
vertex 3:	(	6.70	7.50	0.)
vertex 4:	(	2.70	7.50	0.)

Data for primitive surface patch b

vertex 1:	(	0.	0.	2.50)
vertex 2:	(	9.40	0.	2.50)
vertex 3:	(	9.40	0.	0.)
vertex 4:	(	0.	0.	0.)

Data for primitive surface patch c

vertex 1:	(	0.	0.	2.50)
vertex 2:	(	0.	0.	0.)
vertex 3:	(	0.	1.70	0.)
vertex 4:	(	0.	1.70	2.50)

Data for primitive surface path d

vertex 1:	(	0.	0.	0.)
vertex 2:	(	9.40	0.	0.)
vertex 3:	(	9.40	1.70	0.)
vertex 4:	(	0.	1.70	0.)

Pivot primitive is *a* in position 1

****
Start construct graph representation
****

Construct edge 1
Predecessor is *a* in position 1
Successor is *d* in position 2
Label is 1,3

Construct edge 2
Predecessor is *d* in position 2
Successor is *b* in position 4
Label is 3,1

Construct edge 3
Predecessor is *d* in position 2
Successor is *c* in position 5
Label is 2,4

Fig. 6. The interconnection matrix and the intermediate results computed from Algorithm 3.2.

```
---------------------------------
Construct edge 4
Predecessor is d in position 2
Successor is c in position 3
Label is 4,4
---------------------------------
Construct edge 5
Predecessor is b in position 4
Successor is c in position 3
Label is 4,3
---------------------------------
Construct edge 6
Predecessor is b in position 4
Successor is c in position 5
Label is 2,3
---------------------------------
Construct edge 7
Predecessor is b in position 4
Successor is a in position 6
Label is 3,1
---------------------------------
**********************
   Start constructing 3D object
**********************
---------------------------------
The coordinates for the 1-st symbol in the string is
0.          0.          0.
9.40        0.          0.
6.70        7.50        0.
2.70        7.50        0.
---------------------------------
The coordinates for the 2-nd symbol in the string is
0.          -1.70       0.
9.40        -1.70       0.
9.40        0.          0.
0.          0.          0.
---------------------------------
The coordinates for the 4-th symbol in the string is
0.          0.          0.
9.40        0.          0.
9.40        0.          -2.50
0.          0.          -2.50
---------------------------------
The coordinates for the 5-th symbol in the string is
9.40        -1.70       0.
9.40        -1.70       -2.50
9.40        0.          -2.50
9.40        0.          0.
---------------------------------
The coordinates for the 3-rd symbol in the string is
0.          -1.70       0.
0.          -1.70       -2.50
0.          0.          -2.50
0.          0.          0.
---------------------------------
The coordinates for the 6-th symbol in the string is
0.          0.          -2.50
9.40        0.          -2.50
6.70        7.50        -2.50
2.70        7.50        -2.50
---------------------------------
```

Fig. 6 continued

(2) Construct the correspondence table from G_p as follows: Consider every production rule with nonempty tie-curve lists. Let the production being considered be

$$A\Delta_A \to \chi\Gamma_\chi\Delta_\chi,$$

where

$\Delta_A = (t_1, t_2, \ldots, t_l)$, $t_i \in I$ for all $i = 1,2,\ldots,l$;
$\chi = X_1 X_2 \cdots X_n$, $X_i \in N \cup \Sigma$ for all $i = 1,2,\ldots,n$;
$\Delta_\chi = (g_1, g_2, \ldots, g_l)$, $g_q = z_{q_1} z_{q_2} \cdots z_{q_n}$ for all $q = 1,2,\ldots,l$, $z_{q_i} \in I$ for all $i = 1,2,\ldots,n$.

For all $i = 1,2,\ldots,l$, if z_{i_k}, $1 \leqslant k \leqslant n$, is not a null identifier and $At_i \neq X_k z_{i_k}$, then add the item

$$At_i \to X_k z_{i_k}$$

to the correspondence table if it is a new entry.

(3) For each $X_p y_p$ in the connectivity table, if $X_p \in N$ and there are items

$$X_p y_p \to X_{k1} z_{i_{k1}}, \qquad X_p y_p \to X_{k2} z_{i_{k2}}, \ldots, \qquad X_p y_p \to X_{km} z_{i_{km}}$$

in the correspondence table, then add new items to the connectivity table by replacing $X_p y_p$ with $X_{kl} z_{i_{kl}}$, $l = 1,2,\ldots,m$, if they are new entries.

(4) Repeat step (3) until all the X_i's in the connectivity table are terminal symbols.

EXAMPLE 3. By applying Algorithm 3.3, we obtain the connectivity table of the grammar in Example 1 as follows:

$$a1 \sim d3$$
$$a1 \sim b1$$
$$d2 \sim c4$$
$$d3 \sim b1$$
$$d4 \sim c4$$
$$c3 \sim b4$$
$$b2 \sim c3$$
$$c2 \sim d4$$
$$c2 \sim c4$$
$$b3 \sim a1$$
$$b3 \sim d3$$
$$b3 \sim b1$$
$$c2 \sim d2$$

In the above table, $a1 \sim d3$ means the attaching curve 1 of surface patch a connects to attaching curve 3 of surface patch d, and so forth.

3.4. THE PARSING ALGORITHM

ALGORITHM 3.4. Top-down backtrack parsing algorithm for 3D-plex grammars.

Input. A non-left-recursive context-free 3D-plex grammar $G_p = (N, \Sigma, P, S, I, i_0)$, an input string $w = a_1 a_2 \cdots a_n$, and its associated interconnection matrix M_w. We assume that the productions in P are numbered $1, 2, \ldots, p$.

Output. One left parse for w and M_w if one exists. Otherwise, output "error".

Method.

(1) For each nonterminal A in N, order the alternates for A. Let A_i be the index for the ith alternate of A. For example, if $A\Delta_A \rightarrow \alpha_1\Gamma_{\alpha_1}\Delta_{\alpha_1} \mid \alpha_2\Gamma_{\alpha_2}\Delta_{\alpha_2} \mid \cdots \mid \alpha_k\Gamma_{\alpha_k}\Delta_{\alpha_k}$ are all the A-productions in P and we have ordered the alternates as shown, then A_1 is the index for $\alpha_1\Gamma_{\alpha_1}\Delta_{\alpha_1}$, A_2 is the index for $\alpha_2\Gamma_{\alpha_2}\Delta_{\alpha_2}$, and so forth.

(2) A sextuple $(s, i, \alpha, \beta, \eta, M_\eta)$ will be used to denote a configuration of the algorithm:

   (a) s denotes the state of the algorithm.
   (b) i denotes the state of the algorithm.
   (c) α represents the first pushdown list (L1).
   (d) β represents the second pushdown list (L2).
   (e) η is the derived sentential form.
   (f) M_η is the interconnection matrix associated with η.

   The top of α will be on the right, and the top of β will be on the left. L2 represents the "current" left-sentential form, the one which the expansion of nonterminals has produced. L1 represents the current history of the choices of alternates made and the input symbols over which the input head has shifted. The algorithm will be in one of the three states q, b, or t; q denotes normal operation, b denotes backtracking, and t is the terminating state.

(3) The initial configuration of the algorithm is $(q, 1, \lambda_1, S, \lambda_1, \lambda_2)$, where λ_1 is an empty string and λ_2 is a null matrix.

(4) There are six types of steps. These steps will be described in terms of their effect on the configuration of the algorithm. The heart of the algorithm is the computation of successive configurations defined by a "goes to" relation, $\vdash$. The notation $(s, i, \alpha, \beta, \eta, M_\eta) \vdash (s', i', \alpha', \beta', \eta', M'_{\eta'})$ means that if the current configuration is $(s, i, \alpha, \beta, \eta, M_\eta)$, the parser is to go next into

the configuration $(s', i', \alpha', \beta', \eta', M'_{\eta'})$. Unless otherwise stated, i can be any integer from 1 to $n+1$, α a string in $(\Sigma \cup Q)^*$, where Q is the set of indices for the alternates, and β a string in $(N \cup \Sigma)^*$. M^t_η denotes the part of the interconnection matrix associated with the left terminal string in the sentential form η. The six types of moves are as follows:

(a) Tree expansion:
$(q, i, \alpha, A\beta, \eta_1, M_{\eta_1}) \vdash (q, i, \alpha A, \gamma_1\beta, \eta_2, M_{\eta 2})$,
where $A\Delta_A \rightarrow \gamma_1\Gamma_{\gamma_1}\Delta_{\gamma_1}$ is a production in P, and $\gamma_1\Gamma_{\gamma_1}\Delta_{\gamma_1}$ is the first alternate for A. This step corresponds to an expansion of the partial derivation tree using the first alternate for the leftmost nonterminal in the tree.

(b) Successful match:
$(q, i, \alpha, a\beta, \eta, M_\eta) \vdash (q, i+1, \alpha a, \beta, \eta, M_\eta)$
provided $a_i = a$, $i \leqslant n$, and M^t_η is a submatrix of M_w. If the ith input symbol matches the next terminal symbol derived and the derived interconnection matrix M_η is a submatrix of M_w, we move that terminal symbol from the top of L2 to the top of L1 and increment the input pointer.

(c) Successful conclusion:
$(q, n+1, \alpha, \#, \eta, M_\eta) \vdash (t, n+1, \alpha, \lambda, \eta, M_\eta)$.
The parser has reached the end of the input and has found a left-sentential form which matches the input and an interconnection matrix M_η equals M_w. We can recover the left parse from α by applying the following homomorphism h to α: $h(a) = \lambda$ for all a in Σ; $h(A_i) = p$, where p is the production number associated with the production $A\Delta_A \rightarrow \gamma\Gamma_\gamma\Delta_\gamma$, and $\gamma\Gamma_\gamma\Delta_\gamma$ is the ith alternate for A.

(d) Unsuccessful match:
$(q, i, \alpha, a\beta, \eta, M_\eta) \vdash (b, i, \alpha, a\beta, \eta, M_\eta)$
if $a_i \neq a$ or M^t_η is not a submatrix of M_w. The parser goes into the backtrack mode as soon as the left-sentential form being derived is not consistent with the input or the derived interconnection matrix M^t_η is not a submatrix of M_w.

(e) Backtracking on input:
$(b, i, \alpha a, \beta, \eta_1, M_{\eta_1}) \vdash (b, i-1, \alpha, a\beta, \eta_2, M_{\eta_2})$
for all a in Σ. In the backtracking mode, the parser shifts input symbols back from L1 to L2 and generates the associated interconnection matrix M_{η_2}.

(f) Try next alternate:
$(b, i, \alpha A_j, \gamma_j\beta, \eta_1, M_{\eta_1}) \vdash$
(i) $(q, i, \alpha A_{j+1}, \gamma_j\beta, \eta_2, M_{\eta_2})$, if $\gamma_{j+1}\Gamma_{j+1}\Delta_{j+1}$ is the $j+1$th alternate for A. (Note that γ_j is replaced by γ_{j+1} on the top of L2.)

(ii) No configuration, if $i = 1$, $A = S$, and there are only j alternates for S. (This condition indicates that the parser has exhausted all possible left-sentential forms and their associated interconnection matrices with the input w and input interconnection matrix M_w without having found a parse for w.)

(iii) $(b, i, \alpha, A\beta, \eta_2, M_{\eta_2})$ otherwise. (Here, the alternates for A are exhausted, and we backtrack by removing A_j from L1, replacing γ_j by A on L2, and changing the interconnection matrix.)

The execution of the algorithm is as follows.

Step 1. Starting in the initial configuration, compute successive next configurations $C_0 \vdash C_1 \vdash \cdots \vdash C_i \vdash \cdots$ until no further configurations can be computed.

Step 2. If the last computed configuration is $(t, n+1, \gamma, \lambda, w, M_w)$, emit $h(\gamma)$ and halt. $h(\gamma)$ is the first found left parse. Otherwise, emit the error signal.

Let us consider a computer in which the space needed to store a configuration of the parsing algorithm is proportional to the size of the interconnection matrix and the lengths of the two stacks. It is also reasonable to assume that the time spent computing configuration C_2 from C_1, if $C_1 \vdash C_2$, is a constant independent of the configurations involved. Under these assumptions, we expect the algorithm to take quadric space and at most exponential time as functions of the input length. It is well known that although an exponential function such as 2^n grows faster than any polynomial function of n, for small values of n an $O(2^n)$-time-bounded algorithm can be more efficient than many polynomial-time-bounded algorithms. For example, 2^n itself does not overtake n^{10} until n reaches 59 [6].

EXAMPLE 4. With the input as shown in Figure 7, Algorithm 3.4 computes the sequence of configurations as shown in Figure 8. The notation above each "$\vdash$" sign denotes the step number used in the parsing algorithm.

a	d	c	b	c	a	d
1	3	0	1	0	0	0
0	2	0	0	4	0	0
0	4	4	0	0	0	0
0	0	3	4	0	0	0
0	0	0	2	3	0	0
0	0	2	0	0	0	4
0	0	0	3	0	1	3
0	0	0	0	2	0	2

Fig. 7. The input in Example 4.

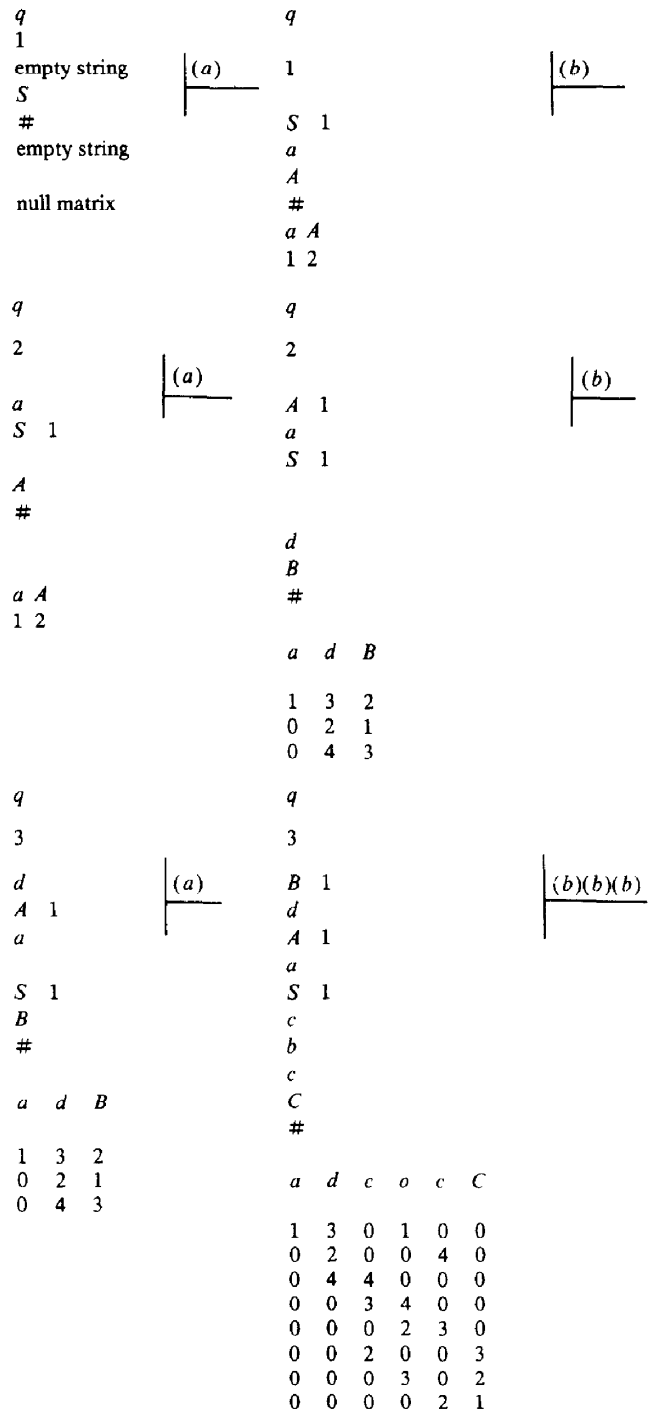
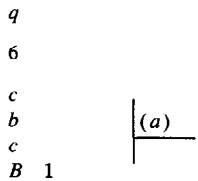
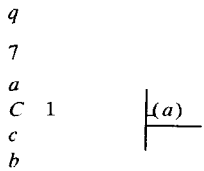


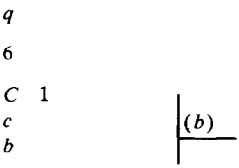
Fig. 8. The sequence of configurations in parsing the input in Figure 7.



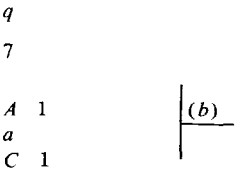
a	d	c	b	c	C
1	3	0	1	0	0
0	2	0	0	4	0
0	4	4	0	0	0
0	0	3	4	0	0
0	0	0	2	3	0
0	0	2	0	0	3
0	0	0	3	0	2
0	0	0	0	2	1



a	d	c	b	c	a	A
1	3	0	1	0	0	0
0	2	0	0	4	0	0
0	4	4	0	0	0	0
0	0	3	4	0	0	0
0	0	0	2	3	0	0
0	0	2	0	0	0	3
0	0	0	3	0	1	2
0	0	0	0	2	0	1

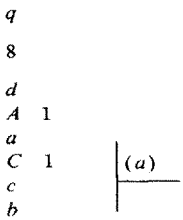


a	d	c	b	c	a	A
1	3	0	1	0	0	0
0	2	0	0	4	0	0
0	4	4	0	0	0	0
0	0	3	4	0	0	0
0	0	0	2	3	0	0
0	0	2	0	0	0	3
0	0	0	3	0	1	2
0	0	0	0	2	0	1

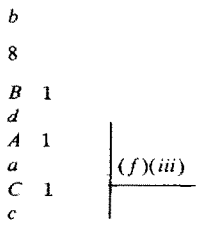


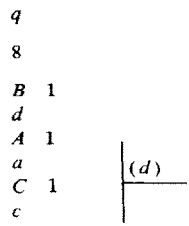
a	d	c	b	c	a	d	B
1	3	0	1	0	0	0	0
0	2	0	0	4	0	0	0
0	4	4	0	0	0	0	0
0	0	3	4	0	0	0	0
0	0	0	2	3	0	0	0
0	0	2	0	0	0	4	3
0	0	0	3	0	1	3	2
0	0	0	0	2	0	2	1

Fig. 8 continued



a	d	c	b	c	a	d	B
1	3	0	1	0	0	0	0
0	2	0	0	4	0	0	0
0	4	4	0	0	0	0	0
0	0	3	4	0	0	0	0
0	0	0	2	3	0	0	0
0	0	2	0	0	0	4	3
0	0	0	3	0	1	3	2
0	0	0	0	2	0	2	1





a	d	c	b	c	a	d	c	b	c	C
1	3	0	1	0	0	0	0	0	0	0
0	2	0	0	4	0	0	0	0	0	0
0	4	4	0	0	0	0	0	0	0	0
0	0	3	4	0	0	0	0	0	0	0
0	0	0	2	3	0	0	0	0	0	0
0	0	2	0	0	0	4	4	0	0	0
0	0	0	3	0	1	3	0	1	0	0
0	0	0	0	2	0	2	0	0	4	0
0	0	0	0	0	0	0	3	4	0	0
0	0	0	0	0	0	0	0	2	3	0
0	0	0	0	0	0	0	2	0	0	3
0	0	0	0	0	0	0	0	3	0	2
0	0	0	0	0	0	0	0	0	2	1

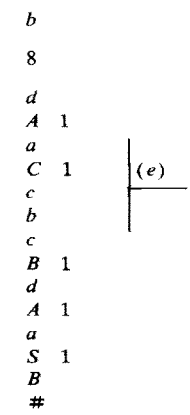
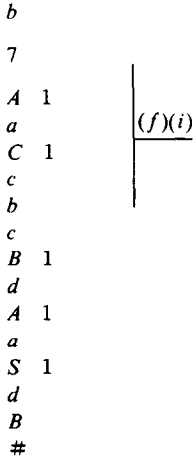
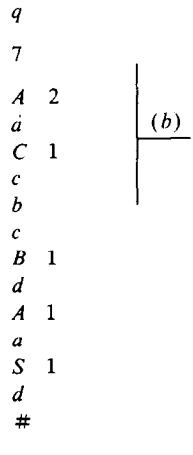


Fig. 8 continued

a	*d*	*c*	*b*	*c*	*a*	*d*	*c*	*b*	*c*	*C*
1	3	0	1	0	0	0	0	0	0	0
0	2	0	0	4	0	0	0	0	0	0
0	4	4	0	0	0	0	0	0	0	0
0	0	3	4	0	0	0	0	0	0	0
0	0	0	2	3	0	0	0	0	0	0
0	0	2	0	0	0	4	4	0	0	0
0	0	0	3	0	1	3	0	1	0	0
0	0	0	0	2	0	2	0	0	4	0
0	0	0	0	0	0	0	3	4	0	0
0	0	0	0	0	0	0	0	2	3	0
0	0	0	0	0	0	0	2	0	0	3
0	0	0	0	0	0	0	0	3	0	2
0	0	0	0	0	0	0	0	0	2	1

a	*d*	*c*	*b*	*c*	*a*	*d*	*B*
1	3	0	1	0	0	0	0
0	2	0	0	4	0	0	0
0	4	4	0	0	0	0	0
0	0	3	4	0	0	0	0
0	0	0	2	3	0	0	0
0	0	2	0	0	0	4	3
0	0	0	3	0	1	3	2
0	0	0	0	2	0	2	1

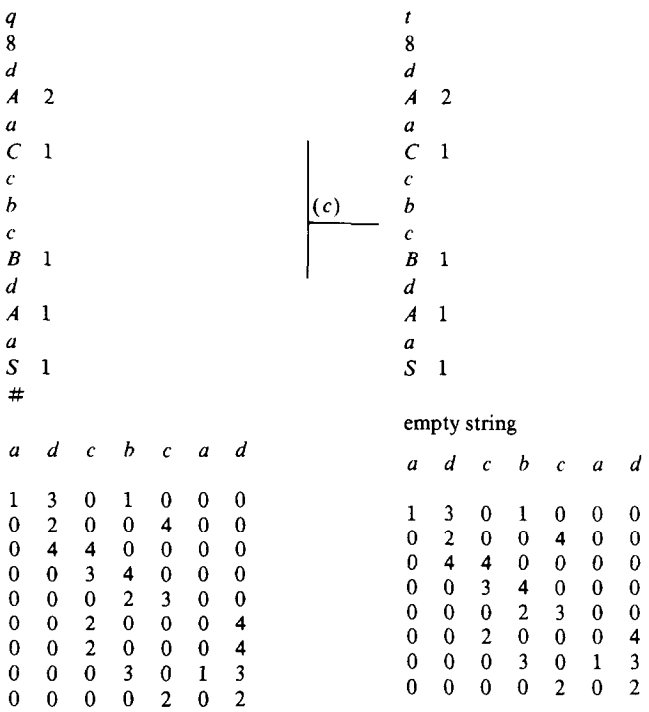




a	*d*	*c*	*b*	*c*	*a*	*d*	*B*
1	3	0	1	0	0	0	0
0	2	0	0	4	0	0	0
0	4	4	0	0	0	0	0
0	0	3	4	0	0	0	0
0	0	0	2	3	0	0	0
0	0	2	0	0	0	4	3
0	0	0	3	0	1	3	2
0	0	0	0	2	0	2	1

a	*d*	*c*	*b*	*c*	*a*	*d*
1	3	0	1	0	0	0
0	2	0	0	4	0	0
0	4	4	0	0	0	0
0	0	3	4	0	0	0
0	0	0	2	3	0	0
0	0	2	0	0	0	4
0	0	0	3	0	1	3
0	0	0	0	2	0	2

Fig. 8 continued



accept with left parse 1,2,4,5,3

Fig. 8 continued

IV. CONCLUDING REMARKS

We have reviewed the 3D-plex grammars and presented several useful algorithms for context-free 3D-plex grammars. In practice, 3D-plex grammars can be used to model 3D objects for computer vision application [5]. This approach provides a hierarchical and systematic paradigm for 3D object recognition by reducing the problem of identifying a 3D object to subproblems of primitive-surface-patch identification and utilizing the structural-relationship descriptive power of 3D-plex grammars to perform structural analysis.

REFERENCES

1. K. S. Fu, *Syntactic Pattern Recognition and Applications*, Prentice-Hall, Englewood Cliffs, N.J., 1982.
2. J. Feder, Plex languages, *Inform. Sci.* 3:225–247 (1971).
3. W. C. Lin and K. S. Fu, A syntactic approach to 3D object representation, *IEEE Trans. Pattern Analysis and Machine Intelligence* PAMI-6(3):351–364 (May 1984).

4. W. C. Lin, K. S. Fu, and T. Sederberg, Estimation of 3D object orientation for computer vision systems with feedback, *J. Robotic Systems* 1(1):59–82 (1984).
5. W. C. Lin, A syntactic approach to 3D object representation and recognition, Ph.D. Thesis, School of Electrical Engineering, Purdue Univ., Aug. 1984.
6. A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974, Chapter 10.

Received 29 May 1984; revised 29 May 1984