ISSN 2321 3361 © 2018 IJESC

Research Article Volume 8 Issue No.6

Higher Dimensional Grammar for Detection of Various Objects

Taniya Seal[1], Dr. Satyendra Singh Tomar[2], Samir Kumar Bandyopadhyay[3]
PG Student[1], Assistant Registrar[2], Advisor to Chancellor[3]
Department of Computer Science & Engineering
University of Calcutta, India[1]
JIS University, India[2, 3]

Abstract:
PLEX grammar involving primitive structures called n-attaching point entity (NAPE) and a set of identifiers associated with each NAPE has been used for pattern generation.Context-free grammars for languages of chemicalstructures, logic diagrams, electrical circuits, and flowcharts are given. Plex languages areused to specify the interconnection of encoded geometric curves to form mesh-like linepatterns, and methods are given by which languages of such line patterns can be classified.

Keywords: Parsing algorithm; Plex grammar; Syntactic pattern recognition;String Grammar

I. INTRODUCTION

In conventional language derivations, each terminal and non-terminal symbol may appear in a string with a symbol to its left or right. Each symbol can be visualized as having two “attaching points,” a left one and a right one, at which it may join to or associate with other symbols. A more general symbol can be envisaged as having an arbitrary number, n, of attaching points for joining to other symbols. A symbol of this type is called an n attaching-point entity (NAPE). Structures formed by interconnecting such entities are called plex structures.’ Plex structures are quite general in nature and include labelledgraphs as a sub case. Chemical structures, electrical circuits and a variety ofother physical systems can be modelled using these structures. Line patternscan be processed to yield plex structures [5]. Languages calledplex languages can be formed from sets of plex structures. Plex languages can be specified using an extension of the phrase-structure scheme used for generating string languages. A grammar used for the specification of a plex language is called a plex grammar. The idea for the plex grammar scheme to be presented was obtained from a generative grammar for a subset of the English letters [8]. Plex structures are similar to PDLs. HoweverPDL patterns always had only two attaching points which limited their use in defining more complicated drawings. This led to development of PLEX structures .A nape denoted by hor(1,2,3), denotes a horizontal line segment with 3 attaching points, labelled by numbers. More complicated napes are defined by first specifying a list of primitive blocks(hor,ver), and then specifying their connections. Some examples have been shown in the figure1.The nape (ver, hor, hor)(110,210)() denotes a structure which is composed of one ver and two hor. This forms the first component of the description. The second component describes how they are connected with each other. (110)denotes that point 1 of ver is connected to point 1 of hor. 201 denotes that point 2 of ver is connected to point 1 of the second hor. A string grammar that generates a set of plex structures is called a plex grammar. The terminals in thesegrammars correspond to napes. There may also be some non-terminal napes, which are described by a setof attaching points. The third component of a nape description is used to specify exactly that.Plex grammars have been used for representing 3D objects [11]. They were the firstgrammar concept to come up with description of real world objects. In this work, they also described aPlex parser for recognizing plex structures.

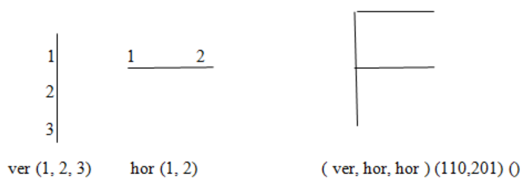


Figure.1. Examples of some nape structures

II. LITERATURE REVIEW

Syntactic approach to pattern recognition use primitives and their relation in theform of production rules to represent complex patterns [3]. Production rules can unambiguously describe how each of the primitive is related to each other.Syntactic approach to pattern recognition, in past, has used object representationusing primitives to form grammar of object [3]. Since then applicability of this approachhas been improved by researchers. Feder described the method to extract context-free rules from the object primitives [2]. Syntactic representation of 3-D objectshas also been studied where the primitives are the surfaces of object [7]. Syntacticpattern recognition uses the rules to construct syntax trees which are then compared torecognize object. Also, automata matching are performed to recognize objects. Recentresearch has focused on representing the object or even the complete scene using itsvarious parts to form parse trees [5], [11] notwithstanding the fact that the applicabilityof this approach has been limited due to the difficulty in extraction of theprimitives.

Proposed Method

The string representation of patterns is quite adequate for structurally simpler forms of patterns. The classical string grammars are, however, weak in handling noisy and structurally complex pattern classes. This is because the only relationship supported by string grammars is the concatenation relationship between the pattern primitives. Here each primitive element is attached with only two other primitive elements-one to its right and the other to its left. Such a simple structure thus may not be sufficient to characterize more complex patterns, which may require better connectivity relationship for their description. An appropriate extension of string grammars has been suggested in the form of high-dimensional grammars. These grammars are more powerful as generators of language and are capable of generating complex patterns like chromosome patterns, nuclear bubble chamber photographs, etc. A class of grammars was suggested where a set of primitive elements may be used with multiple connectivity structure [4]. These grammars are known as PLEX grammars. PLEX grammar involving primitive structures called n-attaching point entity (NAPE) and a set of identifiers associated with each NAPE has been used for pattern generation. The n-attaching point entities are primitive elements in which there are n number of specified points on the primitive elements where other attaching elements may be connected. Thus this class of grammars have more generating capabilities compared to the string grammars.

A plex grammar can be represented by a six-tuple, $[V_T, V_N, P, S, Q, q_0]$, where:

V_T, is a finite non-empty set of NAPES called the terminal vocabulary,

V_N is a finite non-empty set of NAPES called the nonterminal vocabulary,

$V_T \cap V_N = \Phi$,

P is a finite set ofproductions or replacement rules,

S € V, is a special NAPE called the initial NAPE,

Q is a finite set of symbols called identifiers,

$Q \cap (V_T \cap V_N) = \Phi$,

q0€ Q is a special identifier called the null identifier.

The parallel between the terminal vocabulary, non-terminal vocabulary,productions and initial NAPE and the components of a string languagegrammar [3] should be self-evident. The symbols of Q are used to refer to theattaching points of the NAPES. Every attaching point of every NAPE has anassociated identifier. No two attaching points of the same NAPE have thesame identifier. The null identifier, q_0 serves as a place marker and is notassociated with any attaching points. Attachment of NAPES must occur viaactual attaching points of the NAPES; “imaginary” connections using thenull identifier are not permitted. The number of identifiers required for thegrammar is equal to one more than the number of attaching points possessedby the NAPE with the greatest number of attaching points.A set of NAPES is said to be connected if there exists a path through NAPESof the set from any NAPE to any other NAPE in the set. When a closed pathcan be traversed among a set of NAPES, the set is said to contain a loop.Language classes for plex languages are formed by imposing successiverestrictions on an initial unrestricted form of replacement rule as has beendone for string languages. The following notational conventions are adopted :small roman letters are used for denoting lists of NAPES ϵV_Ncapital letters for lists of NAPES ϵV_N; Greek letters for arbitrary lists of NAPES; earlyletters of all alphabets for single NAPES; late letters of all alphabets for listsof NAPES. The unrestricted rule is of the form:

$$\Psi \longrightarrow \omega \quad , \Psi, \omega\text{=connected}$$

The left part of the rule is called the definitum and the right part the definition.The symbol $\longrightarrow$ is read as “can be replaced by.” The meaning is analogous to the meaning of the symbol when used in a conventional grammar. Whereas string language rewrite rules specify only the exchange of one substring for another, the new rules must describe the exchange of one connected substructure for another. The information that is given in the rule is as follows:

Ψ, is called the definitum component list,

ω, is called the definition component list,

Γ_Ψ, is called the definitum joint list,

Γ_ω, is called the definition joint list,

Δ_Ψ, is called the definitum tie-point list,

Δ_ω, is called the definition tie-point list.

The definitum and definition component lists are strings of the form,(G = a_1, a_2 ....a_i.........a_m and w =b_1, b_2,.......b_j ....b_n where a_i and b_j, are single NAPES called components. Ψ and ω list and provide an ordering for the groupsof connected NAPES that comprise the definitum and definition, respectively.The connection of attaching points of two or more NAPES forms a joint. Thedefinitum and definition joint lists specify the way in which the NAPES oftheir respective component lists interconnect. The joint lists are divided intofields. The fields are lists of identifiers of the form, qi1, qi2 . . . qik, that specifywhich attaching points of which NAPES connect at each joint. One field isrequired per joint. The length of the fields for definitum and definition is givenby I(Ψ) and l(ω), respectively, where I denotes the length of its string argument. The entry qi in the jth position of a field indicates that attaching point qi ofthe jth component of the component list preceding r connects at the joint.If the j th component is not involved at the particular joint, then the null identifier, q_0, appears in this position. Each joint list field must contain at leasttwo non-null identifiers.The substructures for the definitum and definition connect to the remainderof the plex at a finite number of joints called tie-points. The tie-point lists givethe correspondence between these external connectors for definitum anddefinition. The tie-point lists are divided into fields; one field specifies eachtie-point. Since the number of tie-points for the definitum and definition mustbe the same, the number of fields for both is the same. The correspondence between tie-points is given by the ordering of the fields: the tie-point specifiedby the pth field of the definitum tie-point list corresponds to the tie-pointspecified by the pth field of the tie-point list for the definition. The tie-pointlist fields specify tie-points in the same way that the joint-list fields specifyjoints. Both types of fields have the same length. Each tie-point list field mustcontain at least one non-null identifier. In cases in which a joint is mentionedin both joint and tie-point lists, there is redundancy in the rule; information about only one of the components meeting at the joint need be furnished in the particular field to achieve a complete specification[10]. The following restrictions are placed on all plex productions in order to avoid ambiguity.

(i) A NAPE cannot connect to itself. A grammar rule definition specifying such a connection is illegal.

(ii) No interconnection among the components of a delinitum or a definition other than that described in the joint list(s) can exist.

(iii) Separate tie-points of a definitum or definition cannot refer to the same joint or attaching point.

(iv) Every attaching point of every NAPE in a grammar rule definitum (definition) must either connect with another NAPE in the definitum (definition) or be one of the tie-points. It follows that every attaching point of every
NAPE in a definitum (definition) must be referenced in at least one field of the definitum (definition).The unrestricted grammar rule can specify the exchange of any substructure for any other substructure. Grammars composed of rules of this type are called unrestricted plex grammars and the languages defined by such grammars are called unrestricted plex languages. Languages and grammars of this type are analogous

to unrestricted string languages and grammars.A context-sensitive rule is obtained from the unrestricted rule by stipulating that $\Psi\Gamma_{\Psi}\Delta_{\Psi}$and $\omega\Gamma_{\omega}\Delta_{\omega}$, must be decomposable as follows:

$$\Psi = A_{\Psi 1};$$
$$\omega = \chi_{\Psi 1};$$
$$\Gamma_{\Psi} = \Gamma_{\Psi 1}\Gamma_{A\Psi 1};$$
$$\Gamma_{\omega} = \Gamma_{\chi}\Gamma_{\Psi 1}\Gamma_{\chi\Psi 1};$$
$$\Delta_{\Psi} = \Delta_{A}\Delta_{\Psi 1};$$
$$\Delta_{\omega} = \Delta_{\chi}\Delta_{\Psi 1};$$

where x ≠null and $\chi\Gamma_{\chi}$ is connected. In the above:

$\Gamma_{\Psi 1}$ is a joint list containing fields of length l(Ψ1,) that describes the interconnection of the NAPES listed as $\Psi 1$;

Γ_{χ} is a joint list containing fields of length l(x) that describes the interconnection of the NAPES listed as χ;

$\Gamma_{A\Psi 1}$, is a joint list containing fields of length l(A) + l($\Psi 1$) = 1 +I($\Psi 1$) &$\Gamma_{A\Psi 1}$ gives the interconnection of A and the components of $\Psi 1$, by listing the joints connecting A to $\Psi 1\Gamma_{\Psi 1}$;

$\Gamma_{\chi\Psi 1}$, is a joint list containing fields of length l(χ) + l($\Psi 1$) that gives the interconnection between $\chi\Gamma_{\chi}$ and $\Psi 1\Gamma_{\Psi 1}$;

Δ_{A} and Δ_{χ}, are tie-point lists that give the correspondence between theattaching points of A and the tie-points of $\chi\Gamma_{\chi}$;

Δ_{Ψ}lists the tie-points of $\Psi 1\Gamma_{\Psi 1}$,.

It is not necessary to state $\Delta_{\Psi 1}$ the context-sensitive rule can be written as follows:

$A_{\Psi 1}\Gamma_{\Psi 1}\Gamma_{A\Psi 1}\Delta_{A} \longrightarrow \chi\Psi 1\Gamma_{\chi}\Gamma_{\Psi 1}\Gamma_{\chi\Psi 1}\Delta_{\chi}$

χ≠ null, $\chi\Gamma_{\chi}$=connected,

This rule states that the NAPE, A, appearing in any context, can be replaced by the subplex given by $\chi\Gamma_{\chi}$. A context-free plex grammar and context-free plex language are obtained by restricting to rules of this type. String languages containing a finite number of strings and the grammars for such languages are referred to asfinite. The analogue of a finite string grammar is afinite plex grammar. The rules are of the form:

$$S \longrightarrow z\Gamma_{z}\,;\ z\Gamma_{z}\ \text{connected}$$

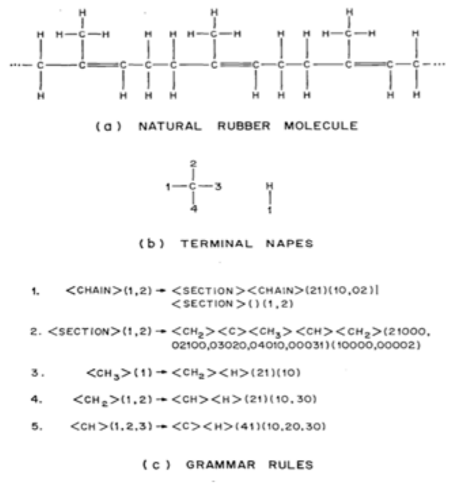
The corresponding language is called a finite plex language[10]. The $z\Gamma_{z}$, are strings on an alphabet, QUV_{T} and can be considered to be encoded versions of the plexes that comprise the finite plex language.

Applications of Plex Grammar

The main drawback of PDL is its limitations to only two concatenation points of subexpressions. This limitation is overcome by plex structures. Such a structure consists of list of primitive elements, a list of internals and a list of external connections. In contrast with PDL,a primitive element in plex structure may have any number of concatenation points. The language generated by a plex grammar is the set of all terminals plex structure that can be derived from the starting symbol by means of productions.The application of plex grammars to pattern recognition is very similar to PDL.That is a class of patterns is represented by means of a plex grammar.Any unknown pattern to be recognized is first converted to a symbolic representation in terms of a plex structure.Then this symbolic representation is parsed according to the underlying grammar.In Plex grammar,the generalization of string grammar into more dimensionalis accomplished by introducing special relational symbols. It has advantages in that the grammatical rules can be described in the predicate form and the parsing can easily be performed using the inference mechanism. But C-Plex grammar as another description of Plex grammar by the clauses of predicate logic, as well ***as*** the top-down parsing of context-free C-Plex grammar based on logic programming by Prolog. C-Plex grammar based on predicate logic programming has the advantage of high descriptive ability and easy parsing. The feature of C-Plex grammar is demonstrated using the structural analysis of the electronic circuit. By this elaboration, not only the hierarchical structure among circuit elements and the macroelements, but also the connection relation among themcan be analysed [9]. An applicationof Plex grammars to the modelling of relevant assemblyinformation for product families expressed by graphs. Anextension of the classical Plex grammar, required byspecific assembly features, is also proposed. The goal ofthis research is to build a database allowing a quickvalidation of a new product we want to insert into anexisting assembly system forflexibility of assembly of the product. The main idea is to create a bridge betweenboth product design and assembly process design forproduct families[12]. 3D plex grammar have been used for representing 3D objects [8]. They were the first grammar concept to come up with description of real world objects. A more recent application of these structures has been in the field of engineering drawing. One such application is the analysis of dimensions in a technical drawing[9]. In this, the document is first digitized and connected components are extracted. These components are then recognized as segments, arrows or text blocks. The dimension model is defined using a dimension grammar [10], and the extracted set of primitive components are then parsed by this grammar. The underlying structure is that of the plex grammar. The parser used also handles noise, which may be present due to the digitization process. Parsers for plexlanguages are more complex than those corresponding to the strings.

Some Examples of Context- free Plex Languages

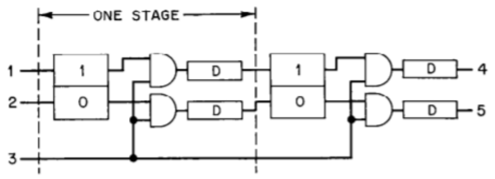
In all examples positive integers are used as identifiers. Zero is reserved as the null identifier, q0. Parentheses are used to enclose the fields comprising Γ and Δ. Successive fields in Γ and Δ are separated by commas. Figures 3, 4, 5, and 6 give some more context-free plex grammars. The grammar shown in Figure 3 describes the repetitive chemical structure of a natural rubber molecule. This structure is shown diagrammed in Figure 3a. The terminal NAPES of the grammar are shown in Figure 3b and consist of the carbon atom, <C>, with four attaching points and the hydrogen atom,<H>, with one attaching point. The grammar rules are given in Figure 3c.The non-terminal NAPES are <CHAIN>, <SECTION>, <CH>,<CH2>, and <CH3>. <CHAIN> is the initial NAPE. Recursion is used in the first rule of the grammar to specify <CHAIN> as an arbitrary number of repetitions of<SECTION>.



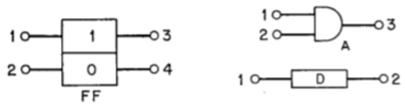
1. <CHAIN>(1,2) → <SECTION><CHAIN>(21)(10,02)| <SECTION>()(1,2)
2. <SECTION>(1,2) → $<CH_2><C><CH_3><CH><CH_2>$(21000, 02100,03020,04010,00031)(10000,00002)
3. $<CH_3>$(1) → $<CH_2>$<H>(21)(10)
4. $<CH_2>$(1,2) → <CH><H>(21)(10,30)
5. <CH>(1,2,3) → <C><H>(41)(10,20,30)

Figure 3 Grammar for natural rubber molecule

Figure 4 gives a grammar that describes a simple shift register formed by assembling a number of identical shift stages. A two-stage shift register of this type is shown diagrammed in Figure 4a. The terminal NAPES of the grammarare shown in Figure 4b and the grammar rules in 4c. Only two grammar rules are used. The first is recursive and describes the initial NAPE, <SHIFT RGSTR>, as consisting of any number of repetitions of the non-terminal,<SHFT STAGE>. The second rule describes <SHFT STAGE> in terms of terminal NAPES.



(a) SIMPLE SHIFT REGISTER



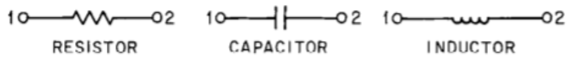
(b) TERMINAL NAPES

```
1. <SHFT RGSTR>(1,2,3,4,5) → <SHFT STGE><SHFT RGSTR>
                             (41,52,33)(10,20,33,04,05)|
                             <SHFT STGE>()(1,2,3,4,5)

2. <SHFT STGE>(1,2,3,4,5) → <FF><A><A><D><D>
                            (31000,40100,02200,03010,00301)
                            (10000,20000,02200,00020,00002)
```

(c) GRAMMAR RULES

FIGURE 4. Grammar for simple shift register.

A grammar for simple resonant circuits is given in Figure 5. The terminal NAPES for the grammar are shown in Figure 5a and consist of the circuit elements, <RESISTOR>, <CAPACITOR> and <INDUCTOR>. The initial A grammar for simple resonant circuits is given in Figure 5. The terminal NAPES for the grammar are shown in Figure 5a and consist of the circuit elements, <RESISTOR>, <CAPACITOR> and <INDUCTOR>. The initial NAPE is <RES CRCT>. The first rule of the grammar (Fig. 5b) defines this NAPE as either a simple series resonant circuit (<SER RES CRCT>), or a simple parallel resonant circuit (<PAR RES CRCT>). <SER RES CRCT> and <PAR RES CRCT> are described in rules 2 and 3 in terms of the non-terminals<R>, <L>, and <C>. <R>, <L>, and <C> are networks of resistors,inductors, and capacitors, respectively. These networks may consist of any number of instances of their respective components connected in series and parallel in arbitrary fashion. In rule 2, a series resonant circuit is described as an <R> followed by an<L> followed by a <C>. Other orderings can be obtained by the use of additional alternatives in this rule.



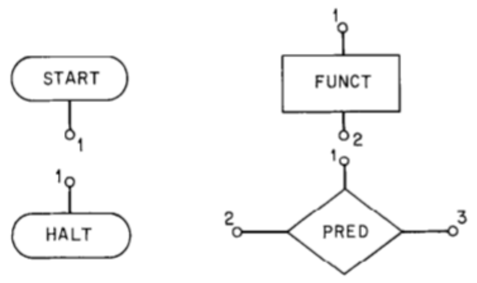
(a) TERMINAL NAPES

```
1.     <RES CRCT>(1,2) → <SER RES CRCT>()(1,2)|
                         <PAR RES CRCT>()(1,2)

2. <SER RES CRCT>(1,2) → <R><L><C>(210,021)
                         (100,002)|<L><C>(21)(10,02)

3. <PAR RES CRCT>(1,2) → <R><L><C>(111,222)(111,222)|
                         <L><C>(11,22)(11,22)

4.              <R>(1,2) → <RESISTOR><R>(21)(10,02)|
                           <RESISTOR><R>(11,22)(11,22)|
                           <RESISTOR>( )(1,2)

5.              <C>(1,2) → <CAPACITOR><C>(21)(10,02)|
                           <CAPACITOR><C>(11,22)(11,22)|
                           <CAPACITOR>( )(1,2)

6.              <L>(1,2) → <INDUCTOR><L>(21)(10,02)|
                           <INDUCTOR><L>(11,22)(11,22)|
                           <INDUCTOR>( )(1,2)
```

(b) GRAMMAR RULES

FIGURE 5. Grammar for simple resonant circuits.

Figure 6 gives two grammars for generating flowcharts composed of start,halt, function, and predicative (decision) blocks. The terminal NAPES are shown in Figure 6a. The grammar in 6b generates a class of flowcharts containingno loops. Figure 6c gives a grammar for a much broader class of flowcharts containing loops. This grammar is a linguistic expression of a result by Jacopini [I]. Any mapping of a set into itself that can be flowcharted can berepresented by a flowchart generated by the grammar in Figure 6c. Other types of physical structures and diagrams can be thought of as statements in plex languages and generated using plex grammars. Although only examples of context-free plex grammars have been given, in some casesmore powerful grammars are required or can be used to advantage.



(a) TERMINAL NAPES

```
1.    <PROG> → <START><END>(11)

2.  <END>(1) → <HALT>(1)|<FUNCT><END>(21)(10)|
               <PRED><END><END>(210,301)(100)
```

(b) GRAMMAR FOR FLOWCHARTS CONTAINING NO LOOPS

```
1.    <PROG> → <START><P><HALT>(110,021)

2.   <P>(1,2) → <FUNCT>()(1,2)|<FUNCT><P>(21)(10,02)|
                <PRED><P>(21,12)(21,30)|<PRED><P>
                <P>(210,301,022)(100,022)
```

(c) GRAMMAR FOR FLOWCHARTS CONTAINING LOOPS

FIGURE 6. Flowchart grammars.

Example of 3D Plex grammar

The following grammar describes the class of objects as shown in Figure 7.

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

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) $C(3) \rightarrow aA(12)(01,12,03)$

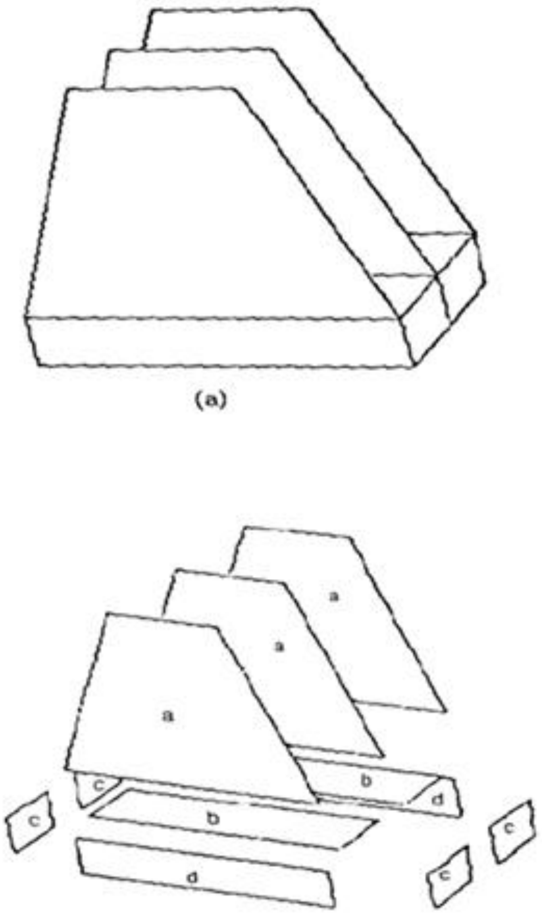


Figure 7 Describes Objects

To generate patterns for the letters, “A” and “H” and sub patterns using a context-free plex grammar given below:

G_p=(N,∑,P,S,I,i_0) where

N={<A>,<H>,<SIDE1>,<SIDE2>,<LETTER>}, ∑={<SL1>, <SL2>,<SL3>, <SL4>,<SL5>}

S=<LETTER>, I={0,1,2}

i_0=0,

and P:

<LETTER><A>|<H> $\longrightarrow$ (1)

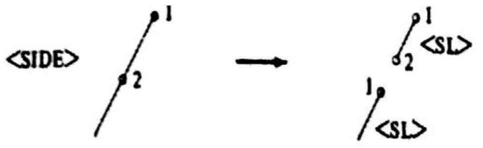
<A><SIDE1><SIDE2><SL5>(110,201,022)() $\longrightarrow$ (2)

<H><SIDE1><SIDE2><SL5>(201,022)() $\longrightarrow$ (3)

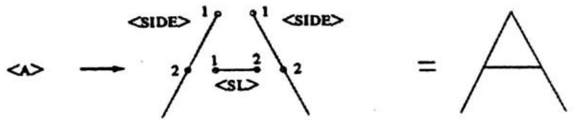
<SIDE1>(1,2) <SL1><SL2>(21)(10,21)$\longrightarrow$ (4)

<SIDE2>(1,2)<SL3><SL4>(21)(10,21)$\longrightarrow$ (5)

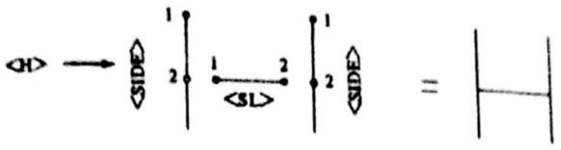
The terminals NAPE in this grammar is <ST1>,<ST2>,<ST3>,<ST4>,<ST5>. This NAPE represents a straight line defined without regard to orientation, with attaching points, denoted by the identifiers 1 and 2, at each end. The non-terminals NAPES for the grammar are <LETTER>, <A>, <H>, <SIDE1>,<SIDE2>.. Only the last of these has associated attaching points. (LETTER) is the initial NAPE. The first rule of the above grammar defines <LETTER> as either an <A> or an <H>. Rule 2 describes an "A" as being constructed of <SIDE1> and <SIDE2> plus a "crossbar" (<ST5>). Three joints are formed; thus there are three fields in Γ.The first field, 110, describes the joint formed by the connection of the two SIDEs. This field states that attaching point 1 of the first component of the definition (<SIDE1>) connects to attaching point 1 of the second component of the definition (<SIDE2>). The zero in the third position indicates that the third component of the definition (<ST5>) is not involved atthe joint. The remaining two joints occur where the "sides" of the "A" connect to the crossbar is described in rule 2 and is similar to that of an "A" except for the absence of the joint connecting the two sides. The component interconnection that takes place in rules 2-5 is illustrated in Figure 8. Rule 4 and 5 of the grammar is the only one containing entries for Δ. These rules describe the joining of two <ST1> and<ST2> components to form the non-terminal NAPE <SIDE21>. The single joint formed is described by Γ = (21). Since <SIDE1> has two attaching points, two tie-point list fields are needed. Attaching point 2 of <SIDE1> corresponds to the tie-point described by 10 (attaching point 1 of the <ST1> component). Attaching point 1 of <SIDE2> corresponds to the tie-point described by 21. This is the joint that is described byΓ.



Rule 4



Rule 2



Rule 3

Figure 8 Explanation of Rules

A phrase structure diagram[11] of an "A" according to the grammar is given in Figure 9. The diagram for an "H" is the same except that the two sides do not meet. The pattern description accomplished by this grammar is essentially topologic. The distinction between an "A" and "H" is based on the presence or absence, respectively, of the extra joint. Since the primitives for the grammar are straight lines defined without regard to orientation, many patterns that would not be considered an "A" or "H" are included in the language.Grammars in which the patterns of interest constitute a subset of the patterns capable of being generated by the grammar can be used to parse patterns in pattern analysis. In this case a knowledge is used of the patterns that are expected to appear.

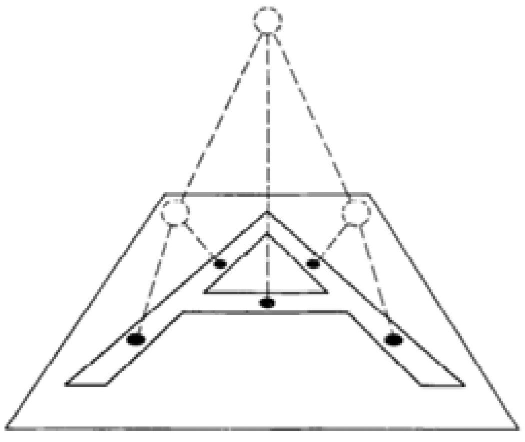


Figure .9. Phrase structure diagramof an "A"

III. ALGORITHM

1) Let there is a list of production rules named **rules as input.**

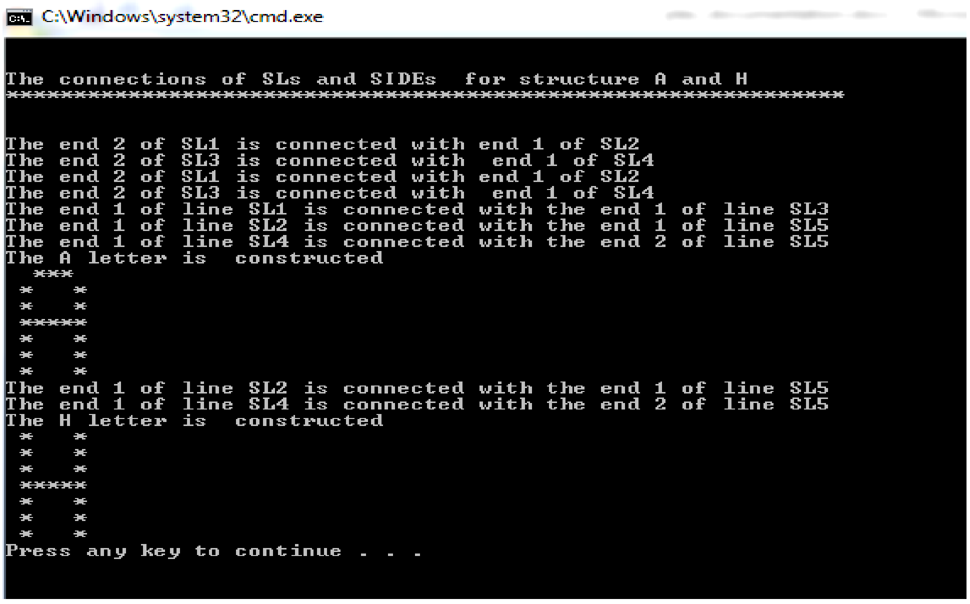
- Let production rule contains terminal, nonterminal named leftpro1,rightpro1,rightpro2,rightpro3. .
- The connection points are named as right1pro1conect, right1pro2conect and right1pro3 connect for 1st connection point, right2pro1conect, right2pro2conect and right2pro3conect for 2nd connection point. right3pro1conect, right3pro2conect, right3pro3 connect for 3rd connection point.
- The attaching points are named as leftpro1end1 and leftpro1end2 for left hand side of the production.rightpro1end1 and rightpro2end1 for 1st attaching point,rightpro2end2 and rightpro2end2 for 2nd attaching point.

```
2) Another inputs be the set of terminals(ter),nonterminals(Nonter),starting symbol(StartSymbol).
3) Initialize f=0,f1=0.
4) foreach Rule r in rules
5)   foreach Rule s in rules
6)     if(s.leftpro1 = r.rightpro1 and  s.leftpro1end1 $\neq$ 0 and  s.leftpro1end2 $\neq$ 0)
7)          for  i=0 to ter.length
8)              for  j=0 to ter.length
9)                    if(ter[i] = s.rightpro1) and  (ter[j] = s.rightpro2)
10)                          s.rightpro1 and s.rightpro2 are connected by their connection points.
11)                  endif
12)                  end for
13)          end for
14)     end if
15)     else if (s.leftpro1 = r.rightpro2 and  s.leftpro1end1 $\neq$ 0 and  s.leftpro1end2 $\neq$ 0)
13)                  for  i=0 to ter.length
14)                for   j=0 to ter.length
```

```
15)                 if(ter[i] = s.rightpro1) and  (ter[j] = s.rightpro2)
16)                         s.rightpro1 and s.rightpro2 are connected by their connection points.
17)                  endif
18)             end for
19)         end for
20)     end if
21)       end else if
22)       else  foreach Rule t in rules
23)             if (ter[i] = t.rightpro3 and s.leftpro1 = t.rightpro2 and r.leftpro1 = t.rightpro1)
24)             if (((t.right1pro1conect =r.leftpro1end1) or (t.right1pro1conect = r.leftpro1end2))
          and  ((t.right1pro2conect = s.leftpro1end1) or (t.right1pro2conect = s.leftpro1end2)))
25)                     if ((t.right1pro1conect = r.leftpro1end1) and (t.right1pro2conect =        s.leftpro1end1))
26)                      The end r.rightpro1end1 of r.rightpro1 is connected with end s.rightpro1end1of s.rightpro1.
27)             end if
28)                     else if ((t.right1pro1conect = r.leftpro1end2) and (t.right1pro2conect = s.leftpro1end2))
29)                       The end  r.rightpro2end2 of r.rightpro2 is connected with end s.rightpro2end2 of s.rightpro2.
30)                       Print "Other than A will be formed ".
31)                       Set f = 1.
32)                     end else if.
33)             else if ((t.right1pro1conect = r.leftpro1end1) and (t.right1pro2conect = s.leftpro1end2))
34)                       The end  r.rightpro1end1 of r.rightpro1 is connected with end s.rightpro2end2 of s.rightpro2.
35)                       Print "Other than A will be formed ".
36)                       Set f = 1.
37)                     end else if.
38)             else if ((t.right1pro1conect = r.leftpro1end2) and (t.right1pro2conect = s.leftpro1end1))
39)                       The end  r.rightpro2end2 of r.rightpro2 is connected with end s.rightpro1end1 of s.rightpro1.
40)                       Print "Other than A will be formed ".
41)                       Set f = 1.
42)             end else if.
43)             Steps 20 to 34 are repeated for other 2 connection of structure "A".
44)             if (((t.right1pro1conect =r.leftpro1end2) or (t.right1pro1conect =r.leftpro1end1)) and
          (t.right1pro3conect = t.rightlineend1)or      (t.right1pro3conect == t.rightlineend2)))
45)                     if ((t.right1pro1conect = r.leftpro1end2) and (t.right1pro3conect =        t.rightlineend1))
46)                      The end r.rightpro2end2ofr.rightpro2is connected with end t.rightlineend1  of t.rightpro3.
47)                    end if
48)                     else if ((t.right1pro1conect =r.leftpro1end1) and (t.right1pro3conect = t.rightlineend2))
49)                     The end r.rightpro1end1 of  r.rightpro1 is connected with end t.rightlineend2 of t.rightpro3.
50)                       Print "Other than H will be formed ".
51)                       Set f1 = 1.
52)                     end else if.
53)                     else if ((t.right1pro1conect =r.leftpro1end1) and (t.right1pro3conect = t.rightlineend1))
54)                     The end r.rightpro1end1 of  r.rightpro1 is connected with end t.rightlineend1 of t.rightpro3
               .
55)                       Print "Other than H will be formed ".
56)                       Set f1= 1.
57)                     end else if.
58)                     else if ((t.right1pro1conect = r.leftpro1end2) and (t.right1pro3conect = t.rightlineend2))
59)                     The end r.rightpro2end2 of  r.rightpro2 is connected with end t.rightlineend2  of t.rightpro3.
60)                       Print "Other than H will be formed ".
61)                       Set f1 = 1.
62)                     end else if.
63)             Steps 36 to 54 are repeated for other 1 connection of structure "H".
64)       end if
65)             if (rule1.rightpro1 =t.leftpro1)
66)               if f=1,break
67)           Otherwise print "The A letter is constructed".
68)             Draw the letter "A".
69)             end if
70)             if (rule2.rightpro1 =t.leftpro1)
71)            if f1=1,break
72)             Otherwise print "The H letter is constructed".
73)Draw the letter "H".
74)       end if.
75)       end else.
76)end foreach .
77)end foreach .
```

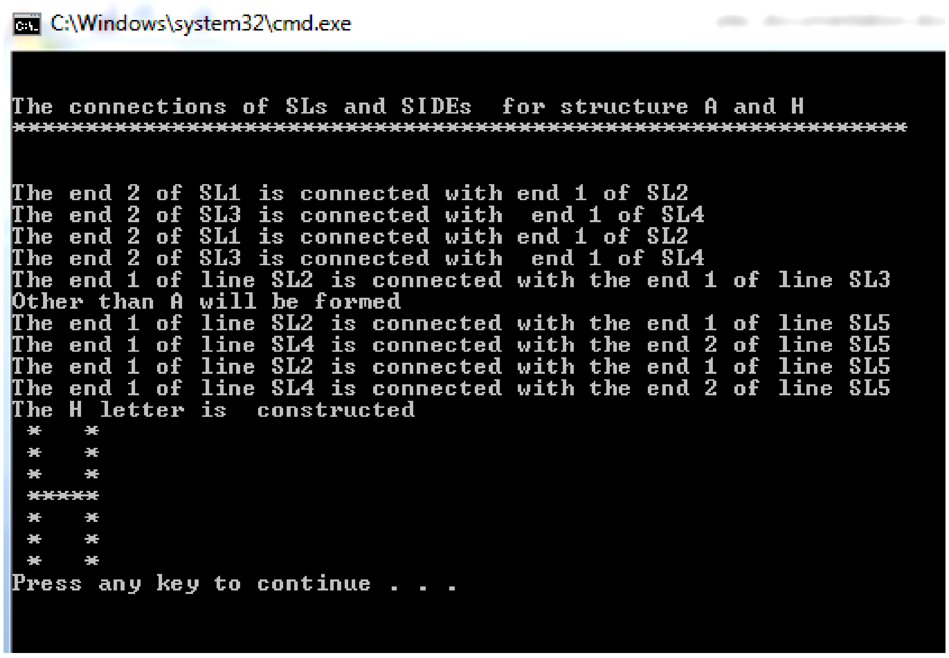
IV. OUTPUT

1. The letter "A" and "H" are constructed.



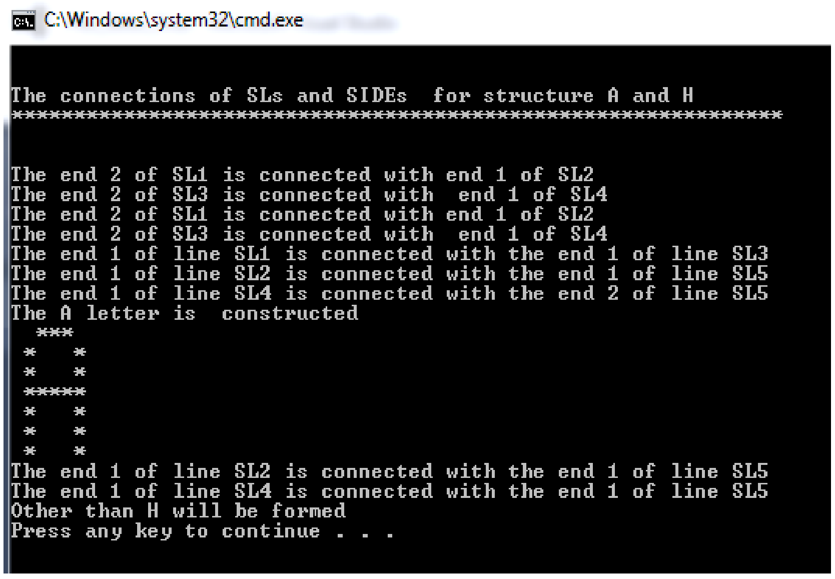
```
The connections of SLs and SIDEs  for structure A and H
*************************************************************


The end 2 of SL1 is connected with end 1 of SL2
The end 2 of SL3 is connected with  end 1 of SL4
The end 2 of SL1 is connected with end 1 of SL2
The end 2 of SL3 is connected with  end 1 of SL4
The end 1 of line SL1 is connected with the end 1 of line SL3
The end 1 of line SL2 is connected with the end 1 of line SL5
The end 1 of line SL4 is connected with the end 2 of line SL5
The A letter is  constructed
  ***
 *   *
 *   *
 *****
 *   *
 *   *
 *   *
The end 1 of line SL2 is connected with the end 1 of line SL5
The end 1 of line SL4 is connected with the end 2 of line SL5
The H letter is  constructed
 *   *
 *   *
 *   *
 *****
 *   *
 *   *
 *   *
Press any key to continue . . .
```

2.Sub pattern of letter A and H are constructed.



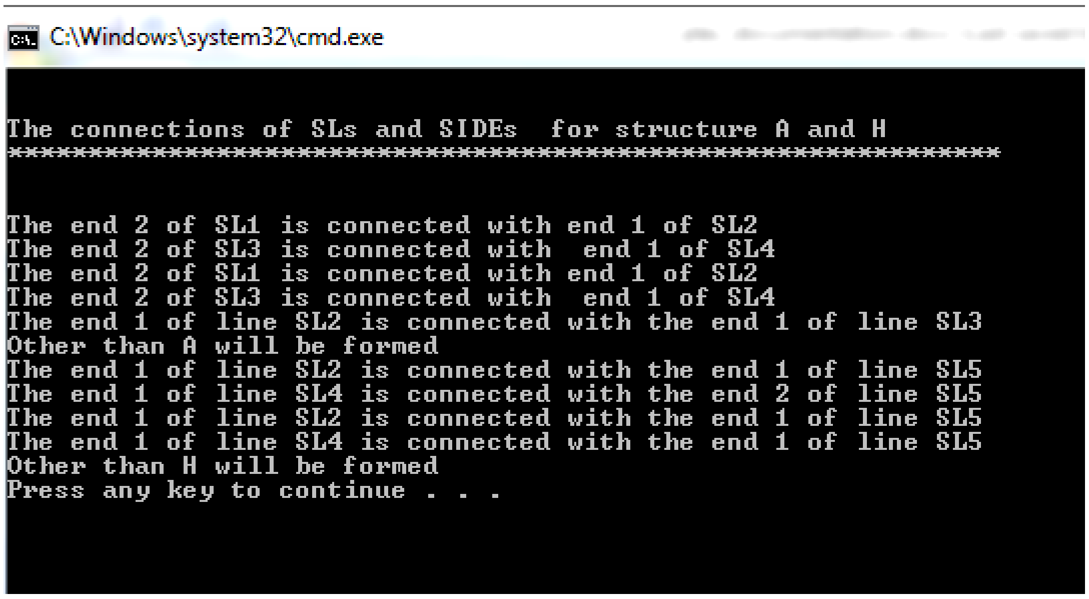
```
The connections of SLs and SIDEs  for structure A and H
*************************************************************


The end 2 of SL1 is connected with end 1 of SL2
The end 2 of SL3 is connected with  end 1 of SL4
The end 2 of SL1 is connected with end 1 of SL2
The end 2 of SL3 is connected with  end 1 of SL4
The end 1 of line SL2 is connected with the end 1 of line SL3
Other than A will be formed
The end 1 of line SL2 is connected with the end 1 of line SL5
The end 1 of line SL4 is connected with the end 2 of line SL5
The end 1 of line SL2 is connected with the end 1 of line SL5
The end 1 of line SL4 is connected with the end 2 of line SL5
The H letter is  constructed
 *   *
 *   *
 *   *
 *****
 *   *
 *   *
 *   *
Press any key to continue . . .
```

3.Subpattern of letter H and A are constructed.



```
The connections of SLs and SIDEs  for structure A and H
*************************************************************


The end 2 of SL1 is connected with end 1 of SL2
The end 2 of SL3 is connected with  end 1 of SL4
The end 2 of SL1 is connected with end 1 of SL2
The end 2 of SL3 is connected with  end 1 of SL4
The end 1 of line SL1 is connected with the end 1 of line SL3
The end 1 of line SL2 is connected with the end 1 of line SL5
The end 1 of line SL4 is connected with the end 2 of line SL5
The A letter is  constructed
  ***
 *   *
 *   *
 *****
 *   *
 *   *
 *   *
The end 1 of line SL2 is connected with the end 1 of line SL5
The end 1 of line SL4 is connected with the end 1 of line SL5
Other than H will be formed
Press any key to continue . . .
```

4.Subpatterns of both letter A and H are constructed.



```
The connections of SLs and SIDEs  for structure A and H
**************************************************************

The end 2 of SL1 is connected with end 1 of SL2
The end 2 of SL3 is connected with  end 1 of SL4
The end 2 of SL1 is connected with end 1 of SL2
The end 2 of SL3 is connected with  end 1 of SL4
The end 1 of line SL2 is connected with the end 1 of line SL3
Other than A will be formed
The end 1 of line SL2 is connected with the end 1 of line SL5
The end 1 of line SL4 is connected with the end 2 of line SL5
The end 1 of line SL2 is connected with the end 1 of line SL5
The end 1 of line SL4 is connected with the end 1 of line SL5
Other than H will be formed
Press any key to continue . . .
```

V. CONCLUSION

An image is described as a composition of its components, called sub images and primitives (the simplest subimages). This approach draws an analogy between the structure of images in terms of components and relations among components and the syntax of a language in terms of grammar rules. For one-dimensional signal and line patterns, the one-dimensional string representation appears to be quite natural and efficient. However, for two-dimensional images and three-dimensional scenes, an extension from the one-dimensional string language approach to higher dimensions will often result in a more efficient representation. Our parser is able to recognize not only complete structures generated by a plex grammar but also partial ones. The strength of syntactic pattern recognition is a well-founded background from conventional language theory. However, the conventional language theory has been established originally for a string model. Parsing for high-dimensional grammars such as plex grammars essentially involves matching structures more complex than strings and is, in general, complex and inefficient. There have been two trends of research to solve the parsing of such grammar models: (1)to develop effective methods to reduce patterns to string or tree representation, (2)to extend the existing language theory to include more general models. However, this paper is concerned with a study of the latter. Plex structures specified by plex grammars are sets of symbols which interconnect in multi-direction. The essential problem in parsing plex structures is the matching of interconnections of symbols in an input plex structure according to a plex grammar. We propose a new parsing scheme for plex grammars, which consists of two phases for symbols and their interconnections in an input plex structure, respectively. An important feature of our parsing scheme is that the complicated interconnections of symbols are examined separately from the recognition process for symbols. In other words, we do not recognize the interconnections of symbols in a parsing process but generate possible interconnections from a given plex grammar to compare them with the interconnections in the input structure. This simplifies the parsing of plex structures, because we are required to parse only a set of symbols instead of a plex structure in our parsing scheme. The task of the first phase is to recognize a set of symbols and to generate interconnections imposed possibly on the symbols from a given grammar. Algorithms for the first phase are primarily discussed in this document

VI. REFERENCES

[1]. Bohm, C., and Jacopini, G. (1966), Flow diagrams, Turing machines and languages with only two formation rules, Comm. ACM 9,366-371.

[2]. Chomsky, N., and Miller, G. A. (1958), Finite state languages, Information and Control, 1,91-112.

[3]. Chomsky, N. (1959), On certain formal properties of grammars, Information and Control,2, 137-167.

[4]. Feder, J. (1968), Languages of encoded line patterns, Information and Control 13,230-244.

[5]. Feder, J. (1969), Linguistic Specification and Analysis of Classes of Line Patterns, NYU Technical Rept. 403- Department of Electrical Engineering, New York University, Bronx, N. Y. 10453.

[6]. Freeman, H. (1961), On the encoding of arbitrary geometric configurations, IEEE Trans. Electronic Computers EC-lo, 260-268.

[7]. Kleene, S. C. (1956), Representation of Events in Nerve Nets and Finite Automata, in Shannon, C. E., and McCarthy, J., Automata Studies, Princeton University Press Princeton, N. J., pp. 341..

[8] .Narasimhan, R. (1966), Syntax-directed interpretation of classes of pictures &mm. ACM9, 166-173.

[9]. K. S. Fu, Syntactic Pattern Recognition and Applications, Prentice-Hall, E&wood Cliffs,NJ., 1982.

[10]. J. Feder, Plex languages, Inform. Sci. 3:225-247 (1971).

[11]. 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).

[12]. G. Boothroyd. Assembly Automation and Product Design. Marcel Dekker, Inc, pp 229-308, 1992.