CHAPTER - 3

SYNTACTIC PATTERN RECOGNITION AND FORMAL LANGUAGE THEORY

The principal objective of this chapter is to provide an introduction to basic concepts of syntactic pattern recognition. The theory of automata and formal languages is one of the principal elements in the study of digital machines and their processing capabilities. Syntactic pattern recognition employes this theory in innovative ways to develop pattern recognition approaches that are based on knowledge of the underlying structure of pattern classes. This chapter presents also the fundamentals of formal language and automata theory as they apply to pattern recognition.

3.1 Basics

Pattern recognition techniques are among the most important tools used in the field of machine intelligence. **Pattern recognition** can be defined as the categorization of input data into identifiable classes via the extraction of significant features or attributes of the data from a background of irrelevant detail [135].

A **pattern** is essentially an arrangement. It may be defined as a quantitative or structural description of an object or some other entity of interest. A **pattern class** is a set of patterns that share some common properties. The subject matter of

pattern recognition by machine deals with techniques for assigning patterns to their respective classes, automatically and with as little human intervention as possible. The study of pattern recognition problems may be logically divided into two major categories :

1. The study of the pattern recognition capability of human beings and other living organisms.
2. The development of underlying theory and practical techniques for machine implementation of a given recognition task.

The first area falls in the domain of psychology, physiology and biology while the second area is in the domain of engineering, computer science and applied mathematics.

Approaches to pattern recognition system design may be divided into two principal categories : (1) the decision-theoretic approach ; and (2) the syntactic approach.

The decision-theoretic approach is based on the utilization of decision functions for classifying pattern vectors and is ideally suited for applications where patterns can have meaningful representation in vector form. There are applications, however, where the structure of a pattern plays an important role in the classification process. In these situations, the decision-theoretic approach has serious drawbacks because it lacks a suitable formalism for handling pattern structures and their relationships. For example, the decision-theoretic approach finds few applications to ECG analysis, since in this case the structure and relationships of the various components of ECG are of fundamental importance in establishing a meaningful recognition scheme.

The syntactic approach to pattern recognition has been receiving increased attention during the past few years because it possesses the structure-handling capability lacked by the decision-theoretic approach. Syntactic pattern recognition is based on concepts from formal language theory, the origins of which may be traced to the middle 1950s with the development of mathematical models of grammars by Noam Chomsky. Basic to the syntactic pattern recognition approach is the decomposition of patterns into **subpatterns** or primitives. By tracking a complex pattern it is possible to detect and encode the primitives in the form of a string of qualifiers. Suppose that we interpret each primitive as being a symbol permissible in some **grammar**, where a grammar is a set of rules of syntax for the generation of **sentences** from the given symbols. Once the grammar has been established, the syntactic pattern recognition process is, in principle, straight forward. Given a sentence representing an input pattern, the problem is to decide whether the input pattern represents a valid sentence. If a pattern is not a sentence of the language under consideration, it is assigned to a rejection class.

3.2 String grammars and language

As indicated in section 3.1, the methods of syntactic pattern recognition are based on mathematical systems in which the patterns of a class are represented as elements of a language. One of the principal requirements in designing a syntactic pattern recognition system is the development of a grammar capable of generating a given class of patterns. Most of the material presented here deals with the definition and interpretation of

grammars suitable for pattern representation and recognition. The following concise definitions establish some of the principal notation and concepts used in formal language theory.

Two categories of sets of interest in formal language theory are **finite sets**, which have a finite number of elements, and **countably infinite sets** whose elements can be placed in a one-to-one correspondence with the positive integer. Given sets A and B, their **Cartesian product** $A \times B$ is the collection of all ordered pairs (a,b), a in A and b in B. For sets $A_1, A_2, \ldots, A_n$ this may be extended to the n-fold Cartesian product $A_1 \times A_2 \times \ldots \times A_n$, which is the set of all n-tuples $(a_1, \ldots, a_n)$ for a_1 in $A_1, \ldots, a_n$ in A_n. The **null set** is the set containing no elements.

A **relation** R from set A to set B is a subset of $A \times B$, i.e., $R \subseteq A \times B$. If for each element a of A there is exactly one element b of B such that (a,b) is in R, then the relation is called a **function (or mapping)** from A to B and is sometimes written $R : A \rightarrow B$.

An **alphabet** V is a finite set of symbols, such that as the binary set $\{ 0,1 \}$.

A **sentence** x over alphabet V is a string of finite length formed with symbols from V. The words 'sentence' and 'string' are used synonymously. The **length** of x, denoted by $|x|$, is the number of symbols used in its formation. The **concatenation** of x with y is the sentence xy with length

$$|xy| = |x|+|y|$$

The **empty string**, denoted by λ, is the sentence with no symbols; it follows that $|\lambda| = 0$.

The **empty** string must be distinguished from the **null**

string ϕ which nulifies when used in concatenation. For any sentence x over V

$$\lambda x = x\lambda = x$$

$$\phi x = x\phi = \phi \ .$$

For any alphabet V, the countably infinite set of all sentences over V, including λ, is the **closure** of V, denoted by V^*. The **positive closure** of V is the set

$$V^+ = V^* - \{ \lambda \}$$

For instance, given alphabet $V = \{ a,b \}$ these sets are

$$V^* = \{ \lambda, a, b, aa, ab, ba, bb, aaa, \ldots \}$$

$$\text{and} \quad V^+ = \{ a, b, aa, ab, \ldots \} \ .$$

A **language** is a set of sentences over an alphabet; i.e., a language over alphabet V is a finite countably infinite subset of V^*. For example, the set

$$L = \{ x | x \text{ is a single } b \text{ or is a finite string of } a\text{'s followed by a single } b \}$$

is a language over $V = \{ a,b \}$.Here the standard notation "$L = \{ x|P \}$" means L is the set of all x having property P. For symbol a in V, let a^n denote the string of n concatenated a's, $n \geq 0$, with a^0 being interpreted as the empty string, λ; then the language above may be defined as

$$L = \{ x | x = a^n b, \ n \geq 0 \}.$$

A basic system studied in formal language theory is one that gives a finite set of rules for generating exactly the set of strings in a specific language. These rules of syntax are embodied in a **grammar**, defined formally as a four-tuple $G = (V_N, V_T, P, S)$ where

V_N is a finite set of **nonterminals** or variables

V_T is a finite set of **terminals** or constants

P is a finite set of **production** or rewriting rules, and

S in V_N is the **starting** symbol.

It is required that V_N and V_T be disjoint sets; that is $V_N \cap V_T = \phi$, the null set. The alphabet V of grammar is the set $V_N \cup V_T$. The set P of productions consisting of rewriting rules of the form $\alpha \rightarrow \beta$ where α is in $V^* V_N V^*$, with the physical interpretation that string α may be written as, or replaced by, β. α must contain at least one nonterminal.

In the remainder of the chapter nonterminals will be denoted by capital letters : $A,B,\ldots,S,\ldots$. Lowercase letters at the begining of the alphabet will be used for terminals : $a,b,c,d\ldots$. String of terminals will be denoted by lowercase letters toward the end of the alphabet : v,w,x,y,z. Strings of mixed terminals and nonterminals will be represented by lowercase Greek letters : $\alpha,\beta,\theta,\rho,\ldots$.

A grammar is often presented by simply listing its productions. If $A \rightarrow \alpha_1, A \rightarrow \alpha_2, \ldots, A \rightarrow \alpha_n$ are the productions for the variable A of some grammar, then these may be expressed by the notation $A \rightarrow \alpha_1 | \alpha_2 | \ldots | \alpha_n$, where the vertical line is read "or".

Given a grammar G, let ρ and δ be strings in V^* and $\alpha \rightarrow \beta$ be a production in P. The notation $\rho\alpha\delta \underset{G}{\Rightarrow} \rho\beta\delta$ to indicate that string $\rho\beta\delta$ is derivable from string $\rho\alpha\delta$ by a single application of a production $\alpha \rightarrow \beta$. The symbol $\underset{G}{\Rightarrow}$ represents the **derivation relation** of grammar G. The symbol $\overset{*}{\underset{G}{\Rightarrow}}$ and $\overset{+}{\underset{G}{\Rightarrow}}$ are used to indicate zero or more uses and one or more uses of the

relation $\underset{G}{\Rightarrow}$. Thus, the notation $\omega \overset{*}{\underset{G}{\Rightarrow}} \theta$ indicates that string θ can be derived from ω by applying zero or more productions from G., while $\omega \overset{+}{\underset{G}{\Rightarrow}} \theta$ indicates that it is necessary to apply one or more productions in order to derive θ from ω.

It is common practice to omit the subscript G when it is clear which grammar is involved in the process. Also since $\overset{*}{\underset{G}{\Rightarrow}}$ includes $\overset{+}{\underset{G}{\Rightarrow}}$, the former symbol is normally used in general expression. For example, the notation $S \overset{*}{\underset{G}{\Rightarrow}} x$, or $S \overset{*}{\Rightarrow} x$ for short may be used to indicate that the terminal string x is derivable from starting symbol S.

The language generated by G, denoted by $L(G)$, is that subset of V_T^* obtained by starting with the nonterminals and using a finite number of productions. In set notation,

$$L(G) = \{ x \mid x \text{ in } V_T^*,\ S \overset{*}{\underset{G}{\Rightarrow}} x \}.$$

A **sentential form** of grammar G is a string over the alphabet V derivable from the starting symbol S. The language $L(G)$ is then the set of sentential forms that consist of terminals only.

Example 3.1 :

Let us consider the grammar $G = (V_N, V_T, P, S)$ with nonterminal set $\{ S \}$, terminal set $\{ a, b \}$, and productions $\{ S \rightarrow aS \mid b \}$. Deriving strings in $L(G)$ in order of increasing length, it can be found

$S \Rightarrow b$ [using production $S \rightarrow b$]

$S \Rightarrow aS \Rightarrow ab$ [using $S \rightarrow aS$, then $S \rightarrow b$]

$S \Rightarrow aS \Rightarrow aaS \Rightarrow aab$ [using $S \rightarrow aS$ twice, then $S \rightarrow b$]

$\vdots$

Information inspection of this derivation process shows that the language generated by the grammar G is

$$L(G) = \{ x \mid x \text{ in } \{ a,b \}^*, S \overset{*}{\underset{G}{\Rightarrow}} x \}$$
$$= \{ b, ab, aab, aaab, \ldots \}$$
$$= \{ x \mid x = a^n b, n \geq 0 \}$$

Suppose that the terminals a and b in this grammar are identified with the pattern primitives in Fig. 3.1 (a), which are used to form R-L-C (resistor-inductor-capacitor) electrical networks. When concatenation of terminals in the sentences of $L(G)$ is interpreted as a physical connection at the junction points of the corresponding primitives, the language represents a class of networks in which any finite number of L-C sections may appear, but there is always exactly one resistor R. Several sample patterns and their representations as sentences in the language are shown in Fig. 3.1(b).

3.3 Chomsky hierarchy

A grammar with no constraints on the form of its productions (other than the general specification of a finite set of string-rewriting rules) is **unrestricted.** Grammars, however, fall into a convenient hierarchy as restrictions on production are imposed. This **Chomsky** hierarchy is defined as follows:

1. A **Context-sensitive grammar** has productions of the form $\theta A \delta \rightarrow \theta \rho \delta$ for θ and δ in V^*, ρ in V^+, and A in V_N. The term **context sensitive** describes the fact that nonterminal A can be rewritten as ρ only when A appears

in the context of substrings θ and δ. An equivalent definition is that for any production $\alpha \rightarrow \beta$ the total number of symbols (nonterminals and terminals) in β must not be less than the number in α; that is, $|\alpha| \geq |\beta|$.

2. A **context-free grammar** has productions of the form $A \rightarrow \alpha$ for A in V_N and α in V^+. The term **context free** arises from the fact that the nonterminal A may be rewritten as string α regardles of the context in which A appears.

3. A **regular grammar** has productions of the forms $A \rightarrow \alpha B$ or $A \rightarrow \alpha$ for A and B in V_N and α in V_T.

These three categories of grammars are also called **types** 1,2, and 3, respectively; unrestricted grammars are classified as **type** 0. It is important to note that all regular grammars are context-free, all context-free grammars are context-sensitive, and all context-sensitive grammars are unrestricted. Generally speaking, unrestricted and context-sensitive grammars provide important results in computation theory. Properties of context-free and regular grammars are important in practical applications as well as in language theory, and many grammars used in syntactic pattern analysis are type 2 or type 3.

It is noted that the definitions of context-sensitive, context-free, and regular grammars given here do not permit productions that can be used to derive the empty string λ ; therefore, λ cannot be in $L(G)$. If in a specific instance it is required that λ belong to the language, then a new starting symbol $S_1 \rightarrow S$ and $S_1 \rightarrow \lambda$, may be introduced, with the second production used only in the derivation of the empty sentence. However, this

is rarely used in syntactic pattern processing because the sentences must provide structural descriptions of patterns, and the languages are usually subsets of V_T^+ rathar than V_T^*.

Given a language L in alphabet V_T, its **complementary language** is

$$\overline{L} = V_T^* - L$$

It can be shown that context-sensitive languages are **recursive** : given context-sensitive grammar G and string in V_T^*, there is an algorithm with inputs G and x that classifies x as a member of $L(G)$ or of $\overline{L}(G)$. [An **effective procedure** is a finite sequence of unambiguous, executable instructions for computing a relationship between input(s) and output(s); and effective procedure that terminates in finite time with a finite number of steps having been executed for any imput(s) is and **algorithm.**] Basically the idea of this algorithm is to enumerate the finite set containing each string θ in V^* such that :

(i) $S \overset{*}{\underset{G}{=\!=\!>}} \theta$ (θ is a sentential form of grammar G), and

(ii) $|\theta| \leq |x|$ (the length of θ does not exceed the length of x).

String x must appear in this enumeration if it belongs to $L(G)$; otherwise x belongs to $\overline{L}(G)$. This result holds for context-free and regular grammars, but it does not hold for the unrestricted case. A **leftmost derivation** is produced by rewriting only the leftmost nonterminal in each step. On the contrary, **rightmost** derivation is produced by rewriting only the rightmost nonterminal in each step. Leftmost derivations are generally taken as the standard way of deriving strings. This in no way restricts the language generated by a grammar, rathar, it simply forces derivations to proceed in an orderly fashion.

It may be the case, for a given grammar G, that there is at least one sentence in $L(G)$ that has two or more distinct leftmost derivations. If this is the case, G is said to be **ambiguous** because each derivation of a string denotes a different syntactic structure and there is no way to determine which structure should be associated with a given occurance of the string. It is not possible to develop an algorithm capable of examining an arbitrary context-free grammar and deciding wheher it is ambiguous or not; however, most of the grammars used in syntactic pattern processing can be individually analyzed for ambiguity, and many are unambiguous.

All finite languages are regular because a finite set of regular productions can always be developed to derive exactly the required strings. Every regular language is a special case of context-free language, but there exist context-free languages that cannot be generated by regular grammars. This increased power of context-free grammars as generators arises from the property of **self-embedding.** A grammar G is self-embedding if there is in V_N at least one nonterminal A such that :

(i) for some α,β in V^*, $S \overset{*}{\underset{G}{\Rightarrow}} \alpha A \beta$; and

(ii) for some θ,δ in V^+, $A \overset{*}{\underset{G}{\Rightarrow}} \theta A \delta$.

In other words, A must be able to embeded itself in nonemply strings and must appear in at least one string in V^+ derivable from the starting symbol S. If the derivation include $S \overset{*}{\underset{G}{\Rightarrow}} uAy$, $A \overset{*}{\underset{G}{\Rightarrow}} vAx$, and $A \overset{*}{\underset{G}{\Rightarrow}} w$ for u,v,w,x,y in V_T^*, then the language $L(G)$ contains at a minimum all strings of the form $uv^i wx^i y$, $i \geq 0$, where v^i denotes i repeitions of substring v and x^i denotes the same numbers of occurances of x.

Example 3.2

Let us consider the terminals a,b,c,d defining the pattern primitives in Fig. 3.2 (a). These directed line segments are the terminals in grammar $G = (\{S,A\}, \{a,b,c,d\}, P, S)$ with productions $\{S \rightarrow cA,\ A \rightarrow aAb|d\}$. The language generated by G is

$$L(G) = \{x|x = ca^n db^n,\ n \geq 0\}.$$
$$= \{cd, cadb, caadbb, \ldots\}.$$

Some sample patterns from this class are given in Fig. 3.2 (b). The self-embedding nonterminal is A, for which $S \overset{*}{\Longrightarrow} cA$, $A \overset{*}{\Longrightarrow} a^n Ab^n$, and $A \overset{*}{\Longrightarrow} d$.

3.4 Equivalent context-free grammars

Two grammars G_1 and G_2 are **equivalent** if $L(G_1) = L(G_2)$. It is sometimes necessary or desirable to replace a given context-free grammar G_1 by an equivalent context-free grammar G_2 that has certain specific properties. For instance, G_2 might have all its productions in some standard form, have no cycles in its derivations, or have no useless productions and nonterminals.

A number of important transformation techniques for context-free grammars are described next. In each case, one begins with G_1 and obtains an equivalent G_2 that is guaranted to have the required characteristics. In the following discussion it is assumed that the context-free grammar $G = (V_N, V_T, P, S)$ is given.

3.5. Cycle-free grammars

A **cycle** is a derivation of the form $A \overset{+}{\Longrightarrow} A$ for A in

V_N. A grammar is **cycle** free if there exists no derivation $A \overset{+}{\Longrightarrow} A$ for any nonterminal A. The existence of cycles merely wastes derivation steps and does not add any power to a grammar. A cycle can occur iff there is a set of productions, $A \rightarrow B_1$, $B_1 \rightarrow B_2$, ..., $B_{n-1} \rightarrow B_n, B_n \rightarrow A, n > 0$, involving single nonterminals only; thus, cycles will be eliminated if all productions of the form $A \rightarrow B$ are removed. Each production of the form $A \rightarrow B$ is known as **unit** production. All other productions including those of the form $A \rightarrow \alpha$ are nonunit productions.

The transformation removing all productions in which a nonterminal is rewritten as a single nonterminal proceeds in the following way.

A new set of productions P' from P is constructed by first including all nonunit productions of P. Then, if $A \overset{*}{\underset{G}{\Longrightarrow}} B$, for A and B in V_N all productions of the form $A \rightarrow \alpha$, where $B \rightarrow \alpha$ is a nonunit production of P, are added to P'.The proof is available in [75].

Example 3.3

Let us consider the grammar $G = (\{ S,A,B \},\{ a,b \},P,S)$ with productions $\{ S \rightarrow aSA,\ S \rightarrow A,\ A \rightarrow BAb,\ A \rightarrow B,$

$$B \rightarrow aS,\ B \rightarrow b \}.$$

i) The new set of productions P' contains all the nonunit production of P.

ii) In the grammar G, $S \overset{*}{\Longrightarrow} A, A \overset{*}{\Longrightarrow} B$, and $S \overset{*}{\Longrightarrow} B$. Thus all the following productions must be included in P'

$S \rightarrow BAb,\ A \rightarrow aS,\ A \rightarrow b$

$S \rightarrow aS, S \rightarrow b$

The equivalent grammar is $G' = (V_N, V_T, P', S)$ with productions

$$S \rightarrow aSA,\ S \rightarrow aS,\ S \rightarrow b,\ S \rightarrow BAb, A \rightarrow BAb,$$

$$A \rightarrow aS,\ A \rightarrow b,\ B \rightarrow aS,\ B \rightarrow b.$$

3.6 Grammars with no useless symbols or productions

It is possible to eliminate from a given context-free grammar G those nonterminals and associated productions that are useless in derivations of terminal strings. A nonterminal A is **useless** if it satisfyes the following two conditions :

1. there is no terminal string x such that $A \overset{*}{\underset{G}{\Rightarrow}} x$; or
2. there is no sentential form $\alpha A \beta$ such that $S \overset{*}{\underset{G}{\Rightarrow}} \alpha A \beta$.

In the first case, A derives no terminal string; in the second case, A is not in any sentential form derived from starting symbol S. If A is useless, all productions of the forms $A \rightarrow \theta$ for any θ or $B \rightarrow \omega A \delta$ for any nonterminal B may be eliminated without affecting $L(G)$.

Thus the useless symbols or productions can be eliminated by checking the two conditions as stated above. The formal proof may be found in [75].

Example 3.4

Let us consider the following grammar

$$S \rightarrow AB \mid a$$

$$A \rightarrow a$$

Applying condition 1, it is found that no terminal string is derivable from B. Thus the nonterminal B and the production $S \longrightarrow AB$ are eliminated. Applying condition 2 to the grammar

$$S \rightarrow a$$

$$A \rightarrow a$$

it is found that only S and a appear in sentential form. So ({ S }, { a }, { $S \rightarrow a$ }, S) is an equivalent grammar with no useless symbol.

It would also be desirable to reduce the size of a grammar by merging sets of nonterminals that, when used as the starting symbol, derive exactly the same set of terminal strings; that is, it is required to test wheather two nonterminals, A and B, are equivalent in the sense of deriving the same set of strings in V_T. Unfortunately, it is not possible to develop an algorithm that will perform this test for arbitrary nonterminals in context-free grammars, although it may be done by careful inspection in some specific cases.

3.7 Chomsky normal form

Various normal forms are available to represent a context-free grammar. Of these, the Chomsky normal form is used extensively in this dissertation.

A context-free grammar is in **Chomsky normal form (CNF)** if each of its production is either of the form $A \rightarrow BC$ for A,B,C in V_N or of the form $A \rightarrow a$ for A in V_N and a in V_T. Given a grammar G_1; an equivalent grammar $G_2 = (\hat{V}_N, V_T, \hat{P}, S)$ in Chomsky normal form is obtained in the following way.

First, the production in P are examined, and all productions already of the form $A \rightarrow BC$ or $A \rightarrow a$ are placed in $\hat{P}$.

The remaining productions in P must be converted to Chomsky normal form. Each of these remaining productions is of the

general form $A \longrightarrow \theta_1\ \theta_2 \ldots \theta_n$, in which each θ_i, $1 \leq i \leq n$, is either a single terminal or a single nonterminal. Each production $A \rightarrow \theta_1 \ldots \theta_n$ must be replaced by a set of productions.

$$
\begin{aligned}
A &\rightarrow Y_1 Y_{2\ldots n} \\
Y_{2\ldots n} &\rightarrow Y_2 Y_{3\ldots n} \\
Y_{3\ldots n} &\rightarrow Y_3 Y_{4\ldots n} \\
&\vdots \\
Y_{n-1,n} &\rightarrow Y_{n-1}\ Y_n
\end{aligned}
$$

in which the subscripted Y's are nonterminals. If θ_i is a nonterminal, Y_i is made equal to θ_i ; if θ_i is a terminal, a new nonterminal Y_i is made and a production $Y_i \rightarrow \theta_i$ is introduced.

The new nonterminals defined in this way become elements of $\hat{V}_N$; the new Chomsky normal form productions become elements of $\hat{P}$. The resultant grammar is G_2.

Example 3.5

Let us consider the grammar $G_1 = (\{ S,A,B \},\{ a,b \},P,S)$ with productions $\{ S \rightarrow BA, A \rightarrow a,\ A \rightarrow abABa,\ B \rightarrow b \}$. In the construction of G_2, the productions $S \rightarrow BA, A \rightarrow a,\ B \rightarrow b$ are first placed in $\hat{P}$. The nonterminals S, A, and B are placed in $\hat{V}_N$.

The remaining production, $A \rightarrow abABa$, is in the form $A \rightarrow \theta_1\ \theta_2\ \theta_3\ \theta_4\ \theta_5$, in which $\theta_1 = \theta_5 = a$, $\theta_2 = b$, $\theta_3 = A$, and $\theta_4 = B$. It is replaced by the productions.

$$
\begin{aligned}
A &\rightarrow Y_1 Y_{2345} \\
Y_{2345} &\rightarrow Y_2 Y_{345} \\
Y_{345} &\rightarrow Y_3 Y_{45} \\
Y_{45} &\rightarrow Y_4 Y_5
\end{aligned}
$$

since θ_3 and θ_4 are nonterminals, the last two of these

productions become

$$Y_{345} \rightarrow AY_{45}$$

$$Y_{45} \rightarrow BY_{5}$$

because θ_1, θ_2, and θ_5 are terminals, new productions $Y_1 \rightarrow a$, $Y_2 \rightarrow b$, and $Y_5 \rightarrow a$ are introduced. The set of new nonterminals added to $\hat{V}_N$ is $\{ Y_1, Y_{2345}, Y_2, Y_{345}, Y_{45}, Y_5 \}$. The Chomsky normal form productions added to $\hat{P}$ are

$$A \rightarrow Y_1 Y_{2345}$$

$$Y_1 \rightarrow a$$

$$Y_{2345} \rightarrow Y_2 Y_{345}$$

$$Y_2 \rightarrow b$$

$$Y_{345} \rightarrow AY_{45}$$

$$Y_{45} \rightarrow BY_5$$

$$Y_5 \rightarrow a$$

The G_1 derivation, for example,

$$S \underset{G_1}{\Longrightarrow} BA \underset{G_1}{\Longrightarrow} bA \underset{G_1}{\Longrightarrow} babABa \overset{+}{\underset{G_1}{\Longrightarrow}} bababa$$

becomes

$$S \underset{G_2}{\Longrightarrow} BA \underset{G_2}{\Longrightarrow} BY_1 Y_{2345} \overset{+}{\underset{G_2}{\Longrightarrow}} baY_{2345} \underset{G_2}{\Longrightarrow} baY_2 Y_{345}$$

$$\underset{G_2}{\Longrightarrow} babY_{345} \underset{G_2}{\Longrightarrow} babAY_{45} \underset{G_2}{\Longrightarrow} babaY_{45}$$

$$\underset{G_2}{\Longrightarrow} babaBY_5 \underset{G_2}{\Longrightarrow} bababY_5 \underset{G_2}{\Longrightarrow} bababa$$

in G_2 .

3.8 Parsing

To parse a string x, which is a sentence in a language $L(G)$ with a known regular or context-free grammar $G = (V_N, V_T, P, S)$, means to determine a sequence of productions used to derive x. If G is ambiguous and x has more than one derivation, the environment

in which G is used determines wheather all derivations of x are required. Any string not in $L(G)$, of course, cannot be parsed using G.

There are two general approaches to parsing. First, given G and x in $L(G)$, one may begin with starting symbol S and attempt to derive x by applying productions in P. In this first case, the syntax analysis proceeds **top-down** from S through intermediate sentential forms until x is obtained and a leftmost derivation of x is found. Second, given G and x in $L(G)$, one may begin with x itself and, by applying productions in a reverse fashion, attempt to reduce x to the satrting symbol S. In this second case, the syntax analysis proceeds **bottom-up** from x to S to yield the reverse of the sequence of productions in a rightmost derivation of x.

Various parsers are available to implement top-down / bottom-up parsing technique for a context-free grammar [10,129]. Choice of a suitable parsing algorithm for a specific task, among number of available methods, is quite difficult. Suitability of an algorithm, amidst many other factors, is also dependent on the problems concerned. So one decides the parsing algorithm considering his own problem. Table based **Cocke-Younger-Kasami (C-Y-K)** [33,135,164] bottom up parsing algorithm is suitable for the present work due to the following reasons :

(i) It is easy to implement.

(ii) Information retrieval is simple as it is table-based.

(iii) As will be seen later, important informations regarding type of complexes in the patient's ECG waveform are available in a suitable form to the physicians.

3.9 The Cocke-Younger-Kasami (C-Y-K) parsing algorithm

This algorithm requires that the grammar to be used for parsing, $G = (V_N, V_T, P, S)$, be given in Chomsky normal form (section 3.7). It is assumed, therefore, that each production in P is either of the form $A \rightarrow BC$ or of the form $A \rightarrow a$. No empty string production like $A \rightarrow \lambda$ is allowed.

Given input string $x = a_1 a_2 \ldots a_n$ to be parsed, a triangular table with elements t_{ij}, $1 \leq i \leq n, 1 \leq j \leq (n - i + 1)$, is considered where i and j are the column and row indices, respectively, and the origin ($i = 1, j = 1$) is located at the bottom left corner. Each table entry t_{ij} must be constructed to contain a subset of V_N, with a nonterminal A entered into t_{ij} if the substring of x begining with a_i and extending for j symbols can be derived from A; in other words, if

$$A \overset{*}{\underset{G}{\Longrightarrow}} a_i \ldots a_{i+j-1} .$$

Ultimately, x is in $L(G)$ iff S is in t_{1n} when the table is completed. In the case that S is in t_{1n}, a leftmost derivation can also be extracted from the table.

The table is built up left to right from the lowest row to the upper entry, t_{1n}. The requirement that G be in Chomsky normal form allows construction of the table for an input string of length n to proceed as according to the following algorithm.

Algorithm CYK

Input : Input string x of length n.

output : Parsing table with elements t_{ij}, $1 \leq i \leq n$, $1 \leq j \leq (n-i+1)$ where i and j are the column and

row indices respectively.

Step 1: Set $j = 1$. Compute t_{i1} as i ranges from 1 to n by [illegible] A in t_{i1} exactly when there is a production $A \rightarrow a_i$ in P.

Step 2: Assuming $t_{i,j-1}$ has been formed for $1 \leq i \leq n$, compute t_{ij} where A is placed in t_{ij} when, for any k such that $1 \leq k < j$, there is a production $A \rightarrow BC$ in P with B in t_{ik} and C in $t_{i+k,j-k}$. Observe that this recursive step is based on a decomposition of the substring $a_i \ldots a_{i+j-1}$ into prefix $a_i \ldots a_{i+k-1}$ and suffix $a_{i+k} \ldots a_{j-k}$ such that

$$B \overset{*}{\Longrightarrow} a_i \ldots a_{i+k-1}, \quad C \overset{*}{\Longrightarrow} a_{i+k} \ldots a_{j-k} \text{ and } A \overset{*}{\Longrightarrow} BC .$$

Step 3: Repeat step 2 until the table is completed or until an entire row has only null entries. x is in $L(G)$ iff S is in t_{1n}.

Step 4: Exit.

For parsing a string of length n, the amount of storage required by this algorithm in the worst case is proportional to n^2 and the number of elementary operations (such as assigning a value to a variable or testing two variables for equality) is proportional to n^3.

Example 3.6

Let $G = (\{S, A, B, C\}, \{a, b\}, P, S)$ have productions

$$S \rightarrow AB \mid AC, \; C \rightarrow SB, \; A \rightarrow a, B \rightarrow b$$

This grammar is in Chomsky normal form with no empty string productions. Its language is $\{ x \mid x = a^m b^m, m \geq 1 \}$.

Let $x = aabb$ be the input string to be parsed. Table construction proceeds as follows :

Step 1 : Set $j = 1$ and compute t_{ij} for $1 \le i \le 4$.

For $a_1 = a$, $t_{11} = \{ A \}$, since there is a production $A \rightarrow a$.

For $a_2 = a$, $t_{21} = \{ A \}$.

For $a_3 = b$, $t_{31} = \{ B \}$, since $B \rightarrow b$ is a production in P.

For $a_4 = b$, $t_{41} = \{ B \}$.

(The bottom row of the Table 3.1 is filled in with these entries.)

Step 2:

(i) First iteration : Set $j = 2$ and compute t_{i2} for $1 \le i \le 3$.

For $a_1a_2 = aa$, we find $t_{12} = \phi$ because there is no nonterminal X such that $X \rightarrow YZ$, $Y \rightarrow a, Z \rightarrow a$.

For $a_2a_3 = ab$, $t_{22} = \{ S \}$ since there are productions $S \rightarrow AB$, $A \rightarrow a$ $B \rightarrow b$.

For $a_3a_4 = bb$, $t_{32} = \phi$.

If at this point t_{22} were also null, the algorithm would be terminated and the input string would be rejected; however, t_{22} is not null, so there is another iteration of this step.

(ii) Second iteration : Set $j = 3$, and compute t_{i3} for $1 \le i \le 2$.

For $a_1a_2a_3 = aab$, $t_{13} = \phi$ because for neither decomposition of substring aab, namely as $(a)(ab)$ or as $(aa)(b)$, it is found that there is a nonterminal X such that $X \rightarrow YZ$ where Y derives the prefix and Z derives the suffix.

For $a_2a_3a_4 = abb$, $t_{23} = C$ since $C \rightarrow SB$, $S \overset{*}{\Longrightarrow} ab$, $B \overset{*}{\Longrightarrow} b$.

(iii) Third iteration : Set $j = 4$ and compute t_{14}.

For $a_1\ a_2\ a_3\ a_4 = aabb$, $t_{14} = \{ S \}$ because $S \rightarrow AC$, $A \overset{*}{\Longrightarrow} a$, and $C \overset{*}{\Longrightarrow} abb$.

Step 3: Stop. The table is complete.

It is concluded that sentence $aabb$ is in $L(G)$. Its leftmost derivation is the production sequence $S \rightarrow AC$, $A \rightarrow a$,

$C \rightarrow SB, S \rightarrow AB, A \rightarrow a, B \rightarrow b, B \rightarrow b.$

The parsing table is shown in Table 3.1.

Table 3.1 The Cocke-Younger-Kasami parsing table.

j \ i	1	2	3	4
4	S			
3	ϕ	C		
2	ϕ	S	ϕ	
1	A	A	B	B

3.10 Concluding remarks

As indicated in this chapter, syntactic pattern recognition methods are based on concepts from formal language theory. Some basic definitions and concepts of formal grammars are introduced. Special stress is given on context-free grammar which will be used as a building block for ECG pattern recognition.

At the end, the Cocke-Younger-Kasami parsing algorithm which is used in the present work is described with example.

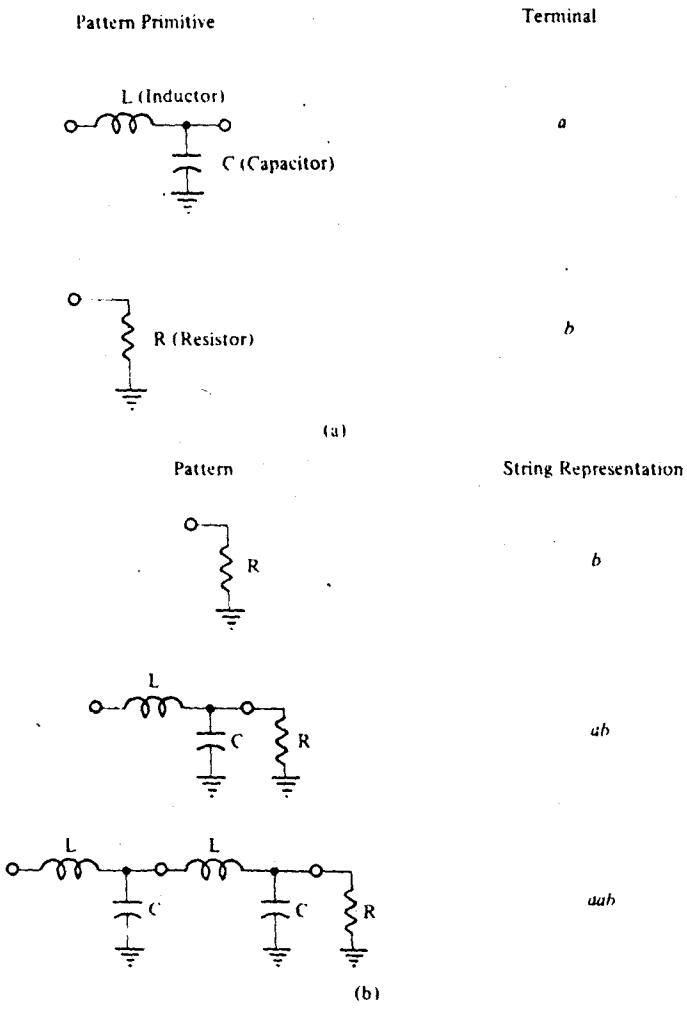

Fig. 3.1 Examples of pattern grammar terminals and sentences. (a) Terminals and their interpretation as pattern primitives. (b) Sample patterns and their representations as terminal strings.

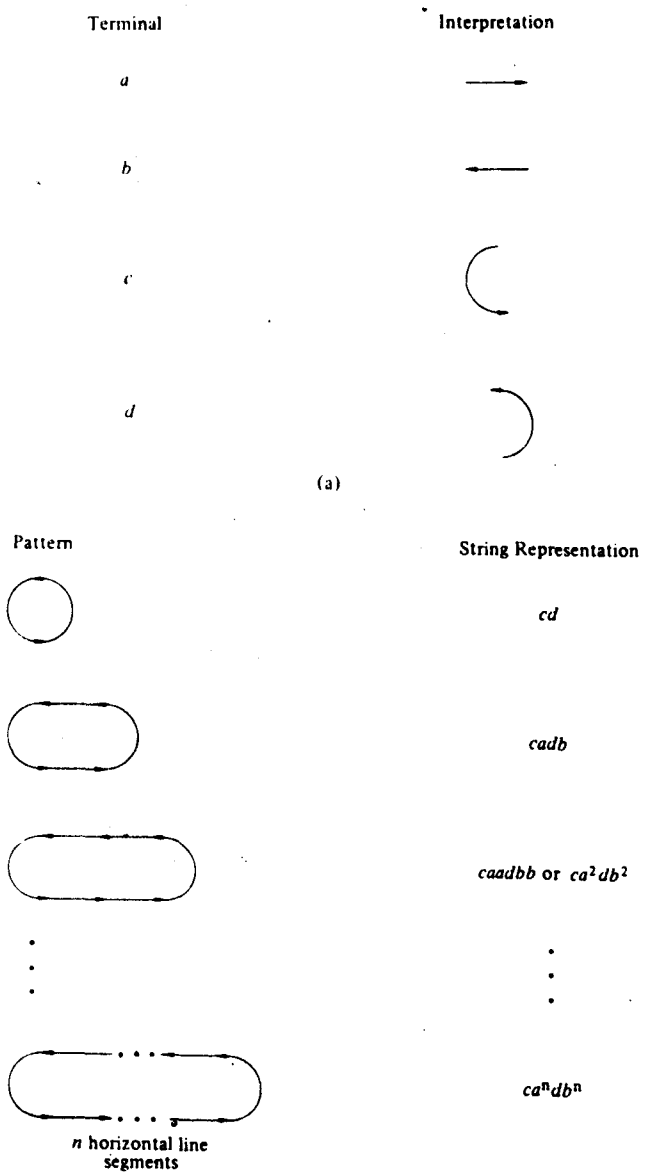

Fig. 3.2 Example of terminals and sentences of a pattern grammar.
(a) Terminals and their pattern primitive interpretations
(b) Sample patterns and their representations as terminal strings.