نظرية الحاسبات
المحاضرة السادسة
الزمن: ساعة

Types of Grammars

Grammars are classified accordingly to the types of productions that define the grammars. Let G be a grammar and let λ define the null string.

(*a*) Let α, α_1 and α_2 denote arbitrary strings on non-terminal symbols and terminal symbols and let A and B denote nonterminal symbols. If the productions in G have the specialised form

$$\alpha_1 A \alpha_2 \rightarrow \alpha_1 \beta \alpha_2$$

where β represents any non-empty string, then G is a, **context sensitive grammar.** The name context sensitive comes from the fact that A can be replaced by β, only when A lies between α_1 and α_2. (Frequently, a grammar is called context sensitive if the productions merely have the form $\alpha \rightarrow \beta$, where $|\alpha| \leq |\beta|$ or of the form $\alpha \rightarrow \lambda$)

(*b*) If the productions in G have the form $A \rightarrow \alpha$, that is, the left side is a single nonterminal and the right side is a word in one or more symbols, *i.e.*, $\alpha \in (V_N \cup U_T)^*$ then G is **context free grammar.** The name context free comes from the fact that a nonterminal symbol that is the left side of a production can be replaced in a string whenever it occurs.

(*c*) A grammar G is said to be **regular** if in each production the left had side is a single nonterminal symbol and the right hand side contains at most one non-terminal symbol which is the right most (or left most) symbol.

The four types of grammars (phrase – structure, context – sensitive, context free and regular) are also known as **type 0, type 1, type 2, type 3** grammars respectively. They form a grammatical hierarchy, called the **chomsky hierarchy.** A type 0 grammar has no restrictions on its productions.

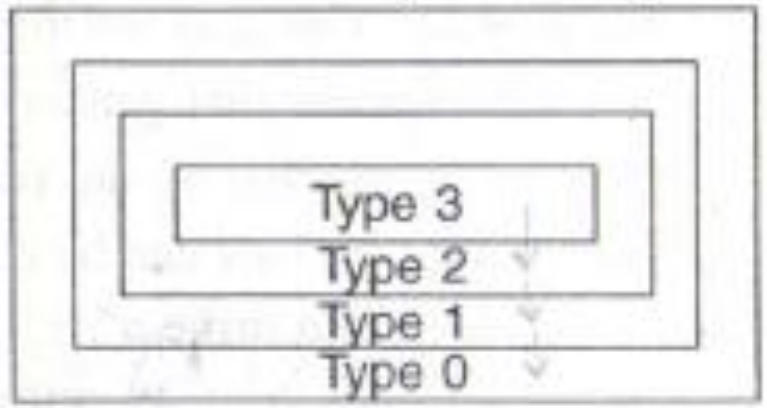
From these definitions we see that every type 3 grammar is a type 2 grammar, every 2 grammar is a type 1 grammar and type 1 grammar is a type 0 grammar. Note the a regular grammar is a context – free grammar and that a context – free grammar with no production of the form $A \rightarrow \lambda$ is a context sensitive grammar.

A language L is context – sensitive (respectively, context-free, regular) if there is a context – sensitive (respectively, context free, regular) grammar G with L = L (G).

To each type of grammar corresponds a kind of machine that produces or accepts a language of the given type : Turning machine (type 0), linear bounded automata (type 1), nondeterministic pushdown automata (type 2) and finite – state machine (type 3)

Class of Grammar, G_i	*Grammatical Characterization*	*Machine Characterization*
Type 0	Unrestricted (or phrase structure)	Turing machine
Type 1	Context-sensitive	Linear-bounded automation
Type 2	Context-free	Pushdown automaton
Type 3	Regular (or right linear)	Finite state machine

A regular grammar is also context free and a context free grammar is also context sensitive. The Venn diagram in Fig 16.1, shows the chomsky heirarchy of the various grammars



Example 19. Determine the type of the grammar G which consists of the production

(*a*) $V_N = \{S, A, B\}$, $V_T = \{a, b\}$, $P = \{S \to aA, A \to aAB, B \to b, A \to a\}$ and starting symbol S.

(*b*) $V_N = \{S, A\}$, $V_T = \{a, b\}$, $P = \{S \to bS, S \to aA, A \to aS, A \to bA, A \to a, S \to b\}$ and starting symbol S.

(*c*) $V_N = \{S, A, B\}$ $V_T = \{a, b, c\}$ with productions $P = \{S \to BAB, S \to ABA, A \to AB, B \to BA, A \to SA, A \to ab, B \to b\}$.

Solution. (*a*) G is a context free or type 2 grammar since each production is of the form $A \to \alpha$, *i.e.*, left side is a single nonterminal symbol and right side is a word n one or more symbols.

(*b*) G is a regular or a type 3 grammar since each production is of the form $A \to \alpha$ or $A \to \alpha B$, that is, the left side is a single nonterminal symbol and the right side is either a single terminal or a terminal followed by a nonterminal.

(*c*) The length of left side of each production does not exceed the length of the right side, hence G is a context sensitive or type 1 grammar.

Example 20. Determine the type of the grammar with the following production

(*a*) $S \rightarrow aAB, A \rightarrow bB, B \rightarrow \lambda$

(*b*) $S \rightarrow ABa, AB \rightarrow a$

(*c*) $S \rightarrow AB, B \rightarrow aAb, aAb \rightarrow b$

Solution. The production $S \to a$ AB and B $\to \lambda$ belong to type – 2 grammar but the production $A \to bB$ belongs to type – 3. Therefore, the grammar with the given productions is of type 2.

(*b*) The production $S \to$ ABa belong to type – 2 grammar but the production $AB \to$ a is a type 0. Therefore, the grammar with the given production is of type 0.

(*c*) The production $S \to$ AB belong to type – 2 grammar while the productions $aAb \to b$ is of type 0 and $B \to Ab$ is of type 1. Hence the grammar with the given production is of type 0.

Since languages are generated by a grammar, therefore corresponding to above four types of grammar, there are four types of languages also. For example,

Language $L = \{a^i b^i : i \geq 1\}$ is a type-2 *language because it can be specified by the type – 2 grammar with productions of the form* $A \to aA\ b$ *and* $A \to ab$.

Example 21. Consider the language $L = \{0^n 1^m : n \neq m\}$, find a context free grammar G which generates L.

Solution. When $n = m$, the grammar G with productions $S \to 0\ 1$, $S \to 0\ S\ 1$ will generate L. In this case G is a context free since each left side is a single nonterminal symbol.

When $n > m$, we first generate a string with equal number of 0's and 1's, then add extra 0's on the left. This is done with

$$S \to AS_1,\ S_1 \to 0\ S_1\ 1,\ S_1 \to \lambda,\ A \to 0\ A,\ A \to 0.$$

When $n < m$, we can use similar reasoning and get the production

$$S \to A\ S_1,\ S \to S_1\ B,\ S_1 \to 0\ S_1\ 1,\ S_1 \to \lambda,\ A \to 0A,\ A \to 0,\ B \to 1\ B,\ B \to 1$$

In all the cases grammar is context free.

Note: there are many other equivalent context free grammars.

Derivation Trees of Context-free Grammar

Given a context free grammar and a string that is derivable from the grammar, it is often useful to know a derivation, because that is what allows us to interpret the string correctly. A natural way of exhibiting the structure of a derivation is to draw a **derivation tree** or **parse tree.** A derivation tree is an ordered tree in which each vertex are labeled with the left sides of a productions and in

which the children of a vertex represents its corresponding right sides. At the root of the tree is the nonterminal symbol with which we begin the derivation. Interior vertex of the tree corresponds to nonterminal symbols that arise in the derivation. The leaves of the tree represents the terminal symbols that arise. If the production $A \to w$ arises in the derivation, where w is a word, the vertex that represents A has children vertices that represent each symbol in w, in order from left to right. In the case of a production $A \to \lambda$, the vertex labeled A has the single child λ. A leaf labeled λ has no siblings that is, a vertex with a child labeled λ can have no other children.

The strings of symbols obtained by reading the leaves of the tree from left to right, omitting any λ's encountered, is said to be the **yield** of the tree.

Example 22. Use the grammar G given as.

$$G = (\{S, A, B\}, \{a, b\}, P, S)$$

where $P = \{(S \to AB), (S \to bA), (A \to a), (A \to aS), (A \to bAA), (B \to b), (B \to bS), (B \to aBB)\}$.

to construct the derivation trees for the strings. (*i*) *aaabbb* (*ii*) *abababba* (*iii*) *aababb*.

Solution. (*i*) The word *aaabbb* can be derived from S as follows

$$S \Rightarrow AB \Rightarrow aB \Rightarrow aaBB \Rightarrow aa\,(aBB)$$
$$b \Rightarrow aaabbb$$

The derivation tree of the word is indicated by Fig. 16.2.

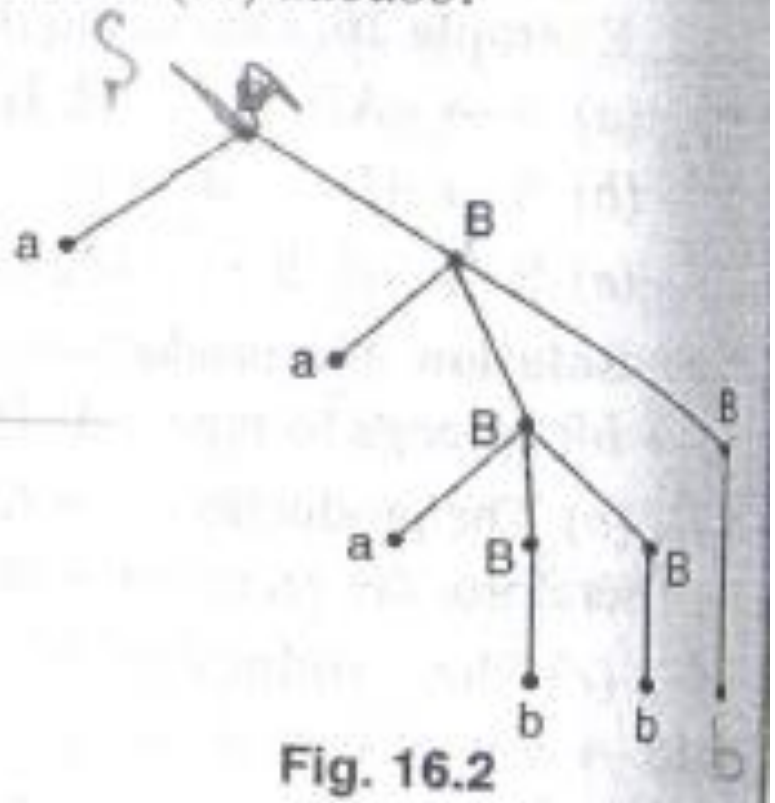
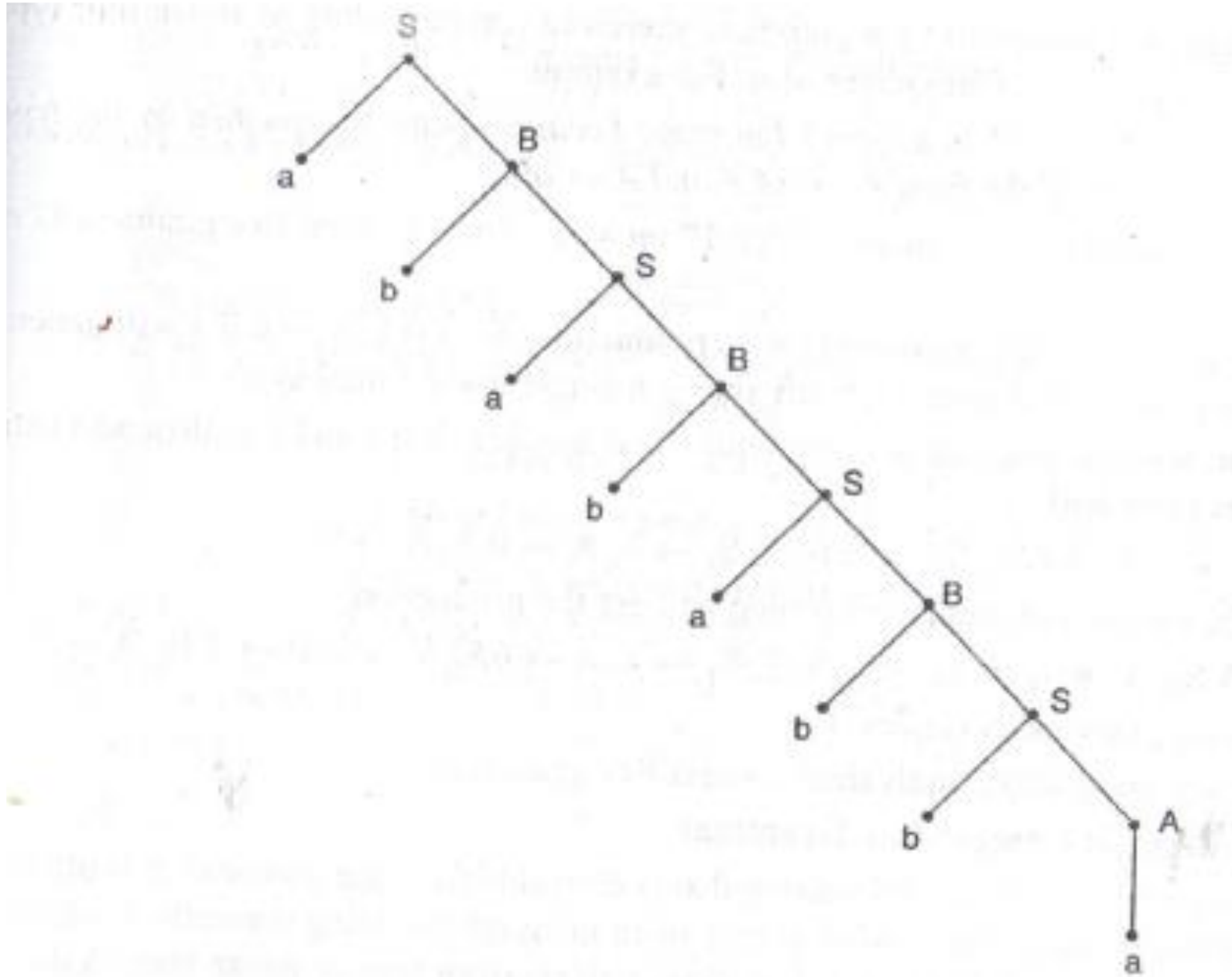


Fig. 16.2

(*ii*) The string abababba can be derived from S as follows

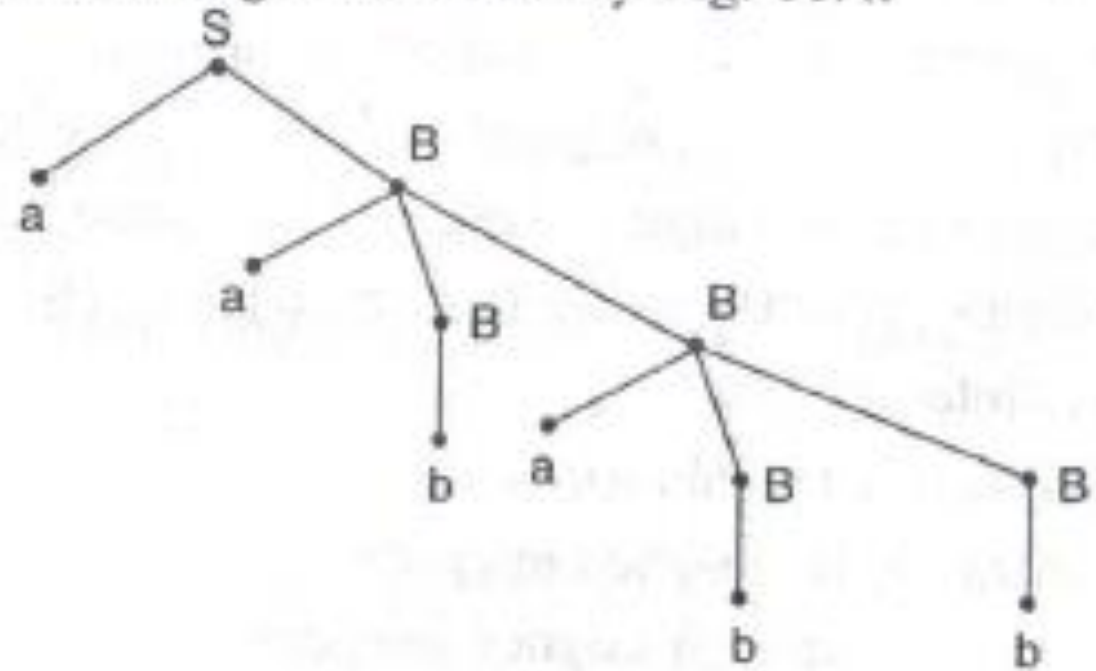
$$S \Rightarrow AB \Rightarrow a\,(bS) \Rightarrow ab\,(AB) \Rightarrow ab\,(a\,bS)$$
$$\Rightarrow abab\,(AB) \Rightarrow abab\,(abS) \Rightarrow ababab\,(bA)$$
$$\Rightarrow abababba$$

The derivation tree of the string is indicated by Fig 16.3.


S
a
B
b
S
a
B
b
S
a
B
b
S
b
A
a

(*iii*) The string aababb can be derived from S as follows $S \Rightarrow AB \Rightarrow a\,(aBB) \Rightarrow aab\,aBB \Rightarrow aababb$

The derivation tree of the string is indicated by Fig. 16.4.



Example 23. Fig 16.5 is the derivation tree of a string ω in a language L of a context-free grammar G (*a*) Find the yield (*b*) which nonterminals, terminals and production must belong to G ?

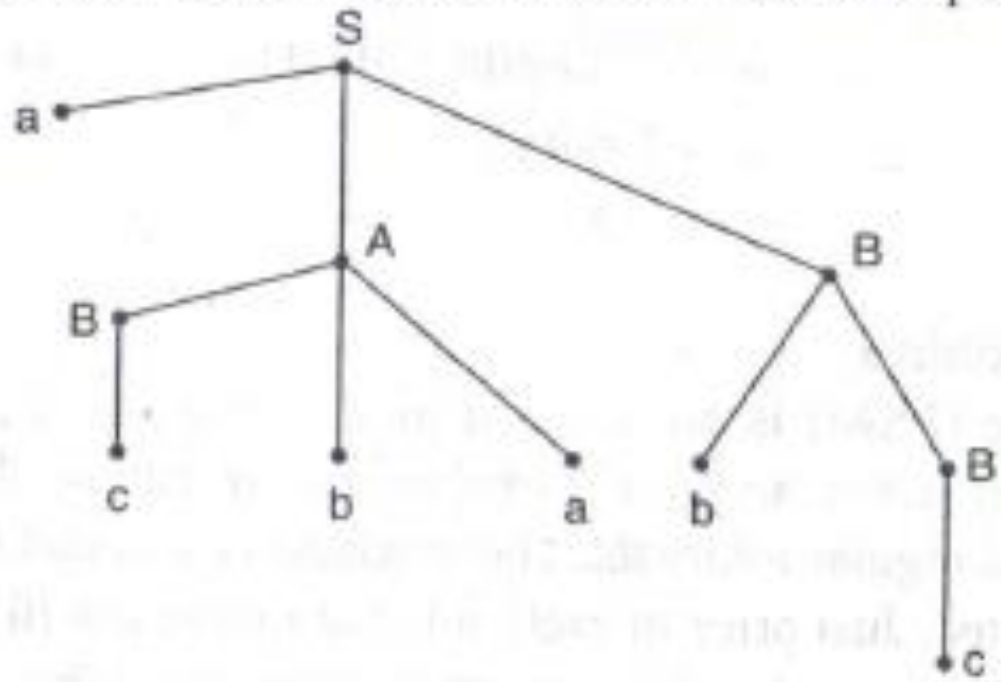


Fig. 16.5

Solution. (*a*) The yield of the tree is the string of symbols which is obtained by reading the leaves of the tree from left to right. Hence the yield of the derivation tree is acbabc.

(*b*) The nonterminal symbol $V_N = \{S, A, B\}$, the terminal symbol $V_T = \{a, b, c\}$, The production $P = \{S \rightarrow aAB, A \rightarrow Bba, B \rightarrow bB, B \rightarrow c\}$ belong to G.

Backus – Naur Form

There is another notation, called the Backus-Naur form (BNF) which is sometimes used for describing the production of context free grammars. This form is used to specify the syntactic rules of many computer languages, including java. In BNF

(*i*) Every nonterminal symbols is enclosed in angle brackets < >.

(*ii*) The terminal symbols are written without any special making.

(*iii*) The symbol : : = is used instead of → and should be read "is defined as".

(*iv*) All the productions with the same nonterminal left-hand side are combined into one statement with all the right hand sides listed on the right of : : =, separated by vertical bars. For instance, the production A → B is written as

$$<A> :: = < B>$$

Productions of the form $<A> :: = <B_1>$, $<A> :: = <B_2>$, ... $<A> = <B_n>$ in BNF may be combined as

$$<A> :: = <B_1> | < B_2> | ..... | <B_n>$$

For instance, production $A \rightarrow A\ a$, $A \rightarrow a$ and $A \rightarrow AB$ can be combined in BNF form as

$$<A> :: = <A>\ a \mid a \mid <A>\ <B>$$

Example 24. Give the Backus – naur form for the production of all integers and show that the derivative of – 32 1 in BNF.

Solution. The following grammar generates all integers.

integer → Signed integer
integer → unsigned integer

<integer> : : = <signed integer> | <unsigned integer>
<signed integer> : : = + <unsigned integer> | – <unsigned integer>
<unsigned integer> : : = <digit> | <digit> | <unsigned integer>
<digit> : : = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 .

The starting symbol is <integer>

The derivative of – 321 in B N F is shown below.

<integer> ⇒ <signed integer>
⇒ – <unsigned integer>
⇒ – <digit> <unsigned integer>
⇒ – <digit> <digit> <unsigned integer>
⇒ – <digit> <digit> <digit>
⇒ – 3 <digit> < digit>
⇒ – 3 2 <digit>
⇒ – 3 2 1.