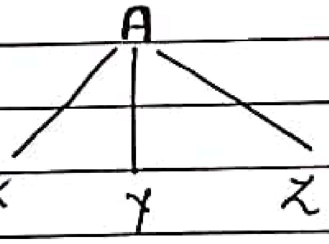
• parse tree -: we can create a graphical representation for derivations that filters out the choice regarding replacement order this representation is called the parse tree, and it has the important purpose of making explicit the hierarchical syntatic structure of sentences that is implied by grammer.
each interior node of the parse tree is labelled by some non-terminal A, and the children of the node are labelled, from left to right, by the symbols in the right-side of the production by which this A was replaced in the derivation.
for example -: if A → xyz is a production used at some step of a derivation then the parse tree for that derivation will have the subtree



The leaves of the parse tree are labelled by non-terminals or terminals and read from left to right. They constitute a sentential form called the yield or frontier of the tree.
for example -: The parse tree for -(id + id) implied by the derivation of the last example is shown as below.

→

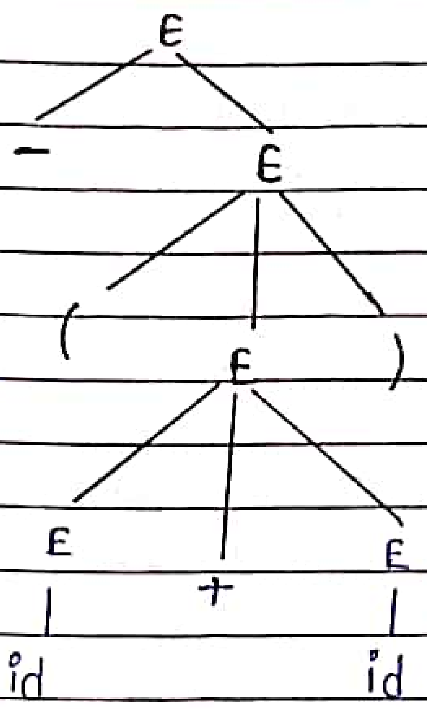


fig: pause tree of $-(id+id)$

For a precise description of pause tree construction, consider a derivation $a_1 \Rightarrow a_2 \Rightarrow \cdots \Rightarrow a_n$, where a_1 is a single non-terminal A.
we construct a pause tree whose yield is a_i for each sentential form a_i, in this derivation.
The process is an induction on i. for the basis. the tree for $a_1 = A$ is a single node labeled A.
To do the induction, suppose we have already constructed the parse tree whose yield is $a_{i-1} = X_1X_2 \ldots X_k$ (Recalling our conventions, each X, is either a nonterminal or a terminal)
Suppose a_i is derived from a_{i-1} by replacing X_j a nonterminal, by $\beta = Y_1Y_2 \cdots Y_r$. That is, at the ith step of the derivation, production $X_j \rightarrow \beta$ is applied to a_{i-1} to derive $a_i = X_1X_2 \cdots X_{j-1} \beta X_{j+1} \cdots X_k$.

To model this step of the derivation, we find the jth leaf from the left in the current pause tree. This leaf is labeled X_j. we give this leaf r chil-dren, labeled $Y_1, Y_2 \ldots Y_r$, from the left. As a special case. if $r=0$ i.e. $\beta = \epsilon$, then we give the jth leaf one child labeled ϵ.

Example -:

Consider the derivation of example. The sequence of pause tree constructed from this derivation is shown in fig 4.2 In the step of the derivation, $E \Rightarrow -E$. To model this step in the creation of the pause tree. we add two children, labeled - and E. to the rootE of the initial tree to create the second tree.

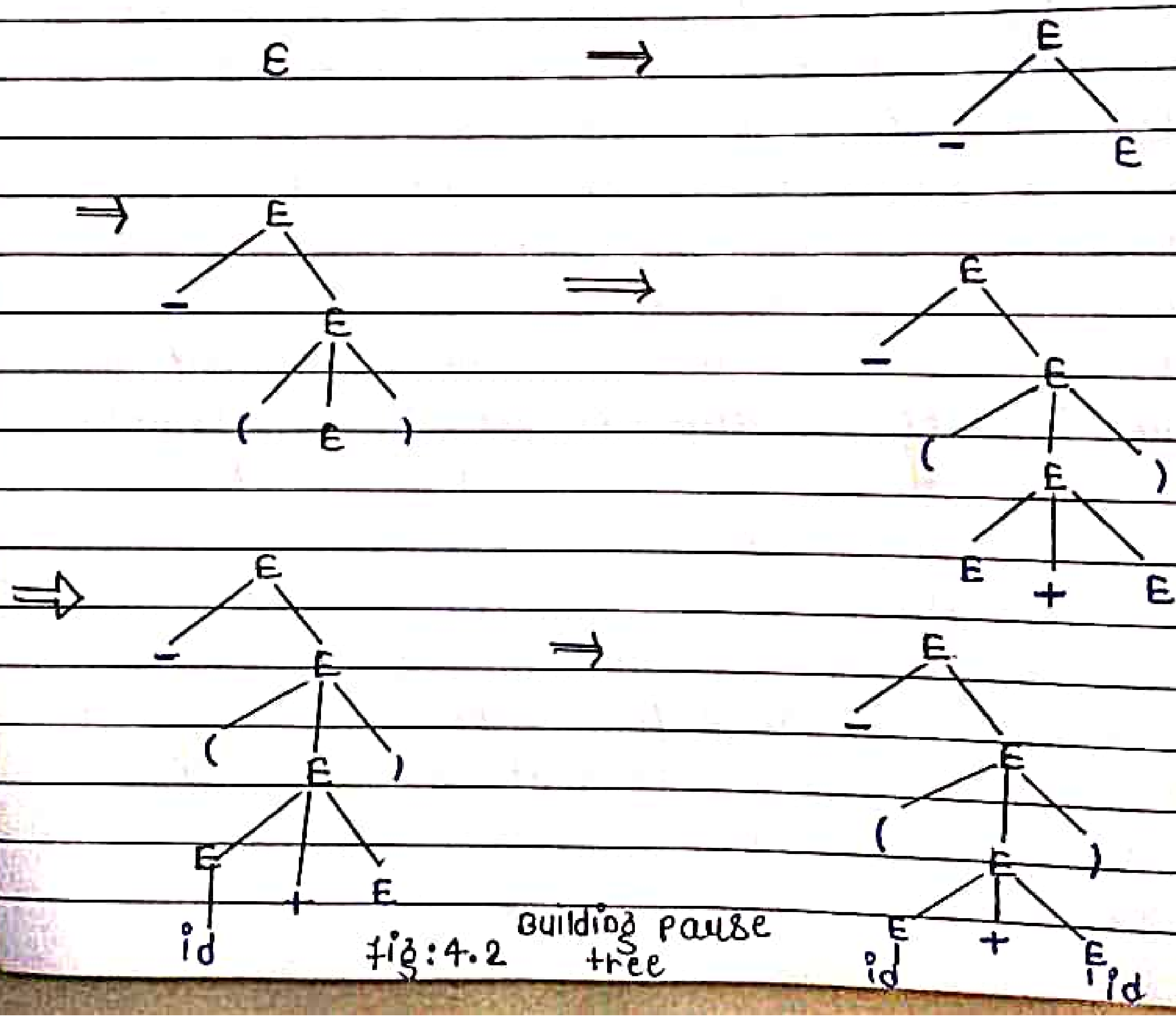


fig:4.2 Building pause tree

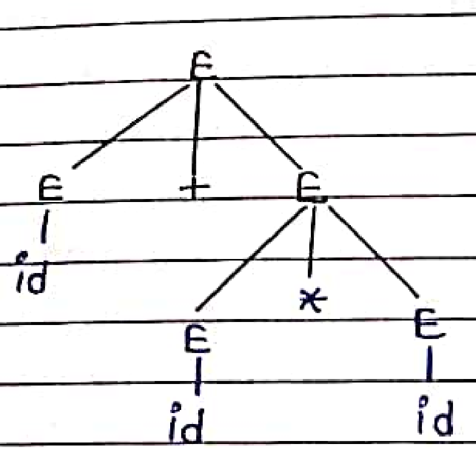
In the second step of the derivation, $-E \Rightarrow -(E)$ consequently we add three children, labeled (, E, and), to the leaf labeled E of the second tree to obtain the third tree with yield -(E). continuing in this fashion we obtain the complete parse tree as the sixth tree.

As we have mentioned, the parse tree ignores variation in the order in which symbols are replaced for example if the derivation of example 4.5 were continued as in line (4.8), the same final parse tree of fig: 4.2 would result. These variations in the order in which productions are applied can also be eliminated by considering only leftmost (or rightmost) derivation it is not hard to see that every parse tree has associated with it a unique leftmost and a unique rightmost derivation. However, we should not assume that every sentence necessarily has only one parse tree or only one leftmost or rightmost derivation.

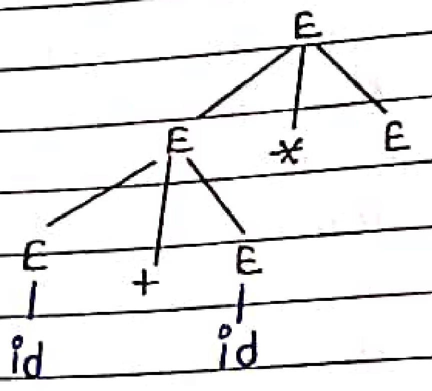
Example 4.5 let us again consider the arithmetic expression grammer (4.7) with which we have been dealing the sentence id + id * id has the two distinct leftmost deriavations.

$E \Rightarrow E + E$	$E \Rightarrow E * E$
$\Rightarrow id + E$	$\Rightarrow E + E * E$
$\Rightarrow id + E * E$	$\Rightarrow id + E * E$
$\Rightarrow id + id * E$	$\Rightarrow id + id * E$
$\Rightarrow id + id * id$	$\Rightarrow id + id * id$

with the two corresponding parse tree shown in fig 4.3



(a)



(b)

fig: 4.3 two parse trees for id + id * id

Note that, in some sense, the parse tree of fig 4.3(a) is "correct" in that it reflects the commonly assumed "precedence" of + and *. while the tree of fig 4.3(b) does not. it is customary to treat operator * as having higher precedence than +. so fig 4.3 (a) which groups the operands of * before those of +, corresponds to the structure we would normally attribute to an expression like a+b*c.

Ambiguity -: A grammer that produces more than one parse tree for some sentence is said to be ambiguise put another way, an ambiguise grammer is one that produces more than one left most and more than one right most for the sentences.

example -:

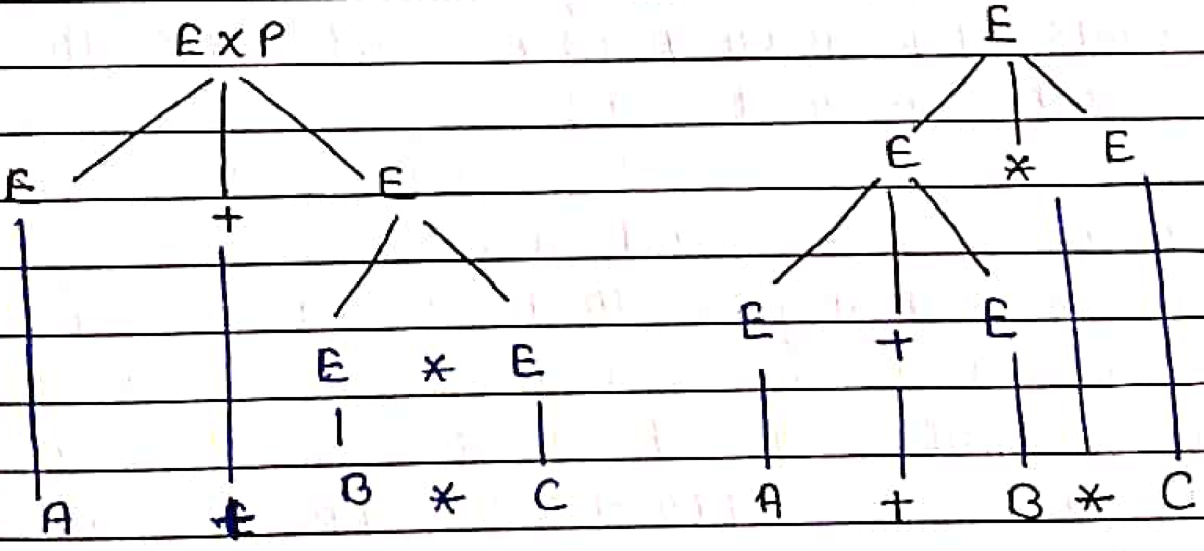
consider the following grammer for arithmetic expression.

E → E+E | E−E | E*E | E/E | E↑E | (E) | −E | id.

This grammer is ambiguise

example -:

A + B * C



• Basic parsing techniques -:

parsers -: A parser for grammer G is a program that takes as input a string w and produces as output either a parse tree for w, if w is a sentence of G, or an error message indicating that w is not a sentence of G. often the parse tree is produced only in a figurative sense, in reality, the parse tree exist only as a sentence of actions made by stepping through the tree construction process.

=) There are two basic types of parsers for CFG.

1) Bottom-up parser
2) Top-down parser

As indicated by their name, bottom-up parser build parse trees from the bottom (leaves) to the top (root), while top down parser start with the root and work down to the leaves in both cases the input to the parser is being scanned from left to right, one symbol at a time.

The bottom-up parsing method we discuss is called "shift-Reduced" parsing because in consists of shifting input symbol on to a stack until the right side of a production appears on top of the stack.

The right side may then be replaced by (Reduced to) the symbol on the left side of production, and the process repeated.

- Representation of a parse tree -:
  There are two basic types of representation-:

1) Implicit
2) Explicit.

The sequence of production used in some derivati--on is an example of an implicit representation. A linked list structure for the parse tree is an explicit representation.

◎ Left-Most derivation -: A derivation in which the left most non-terminal is replaced at every step is said to be left most derivation.

$\alpha \Rightarrow \beta$ by a step in which the left most non-terminal in α is replace, we write $\alpha \underset{lm}{\Rightarrow} \beta$

if $S \overset{*}{\underset{lm}{\Longrightarrow}} \alpha$ then we say α is left sentential form of the grammer at hand.

◎ Right-Most derivation -: In right most derivation right most non terminal is replaced with every step. right most non-terminal is replaced some time called canonical derivation.

example -:

Consider the grammer,

i) $S \rightarrow ictS$
ii) $S \rightarrow ictSeS$
iii) $S \rightarrow a$
iv) $c \rightarrow b$

Here i, t, e stands for if, then and else, C, S
for condition and statements.

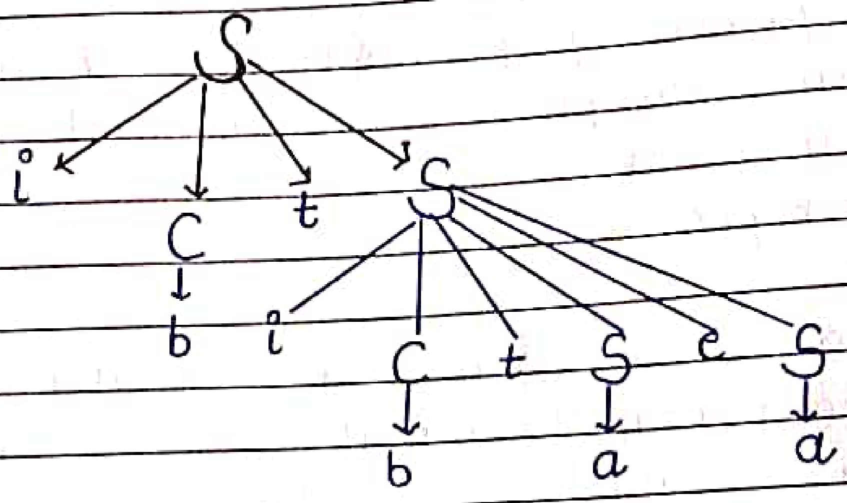


fig: parse tree of the above grammer.

we shall construct a left most derivation for the sentence w = ibitibtaca

The left most derivation corresponding to this parse tree is -:

$S \underset{lm}{\Rightarrow} ictS$

$\underset{lm}{\Rightarrow} ibtS$

$\underset{lm}{\Rightarrow} ibtictSeS$

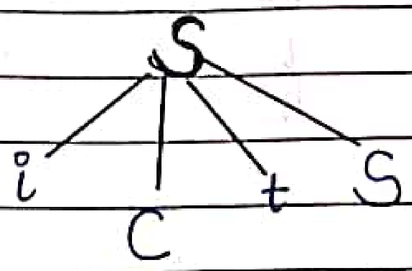
$\underset{lm}{\Rightarrow} ibtibtSeS$

$\underset{lm}{\Rightarrow} ibtibtaeS$

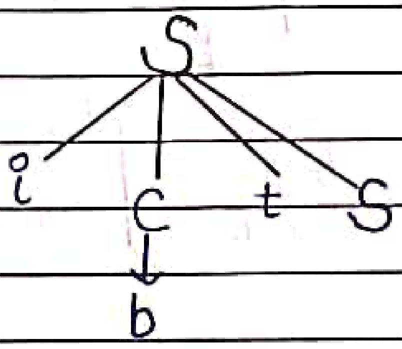
$\underset{lm}{\Rightarrow}$ ibtibtaea

The portions of the parse tree corresponding to the above steps of the derivation are shown below.

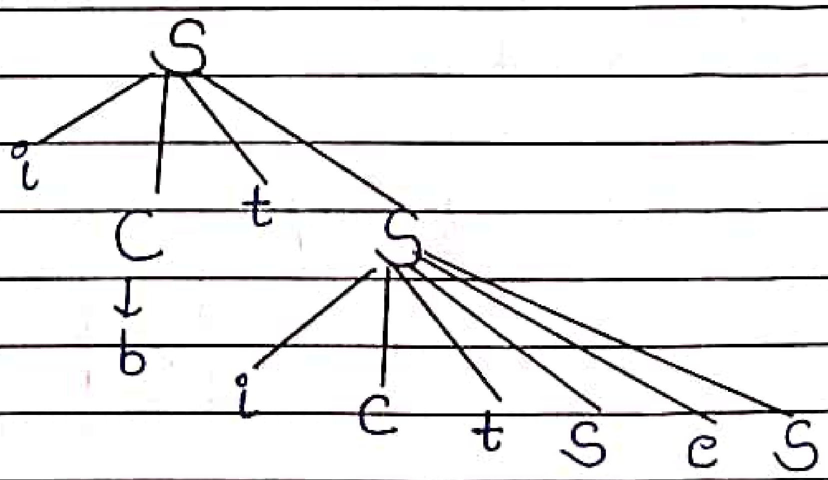
$\rightarrow$



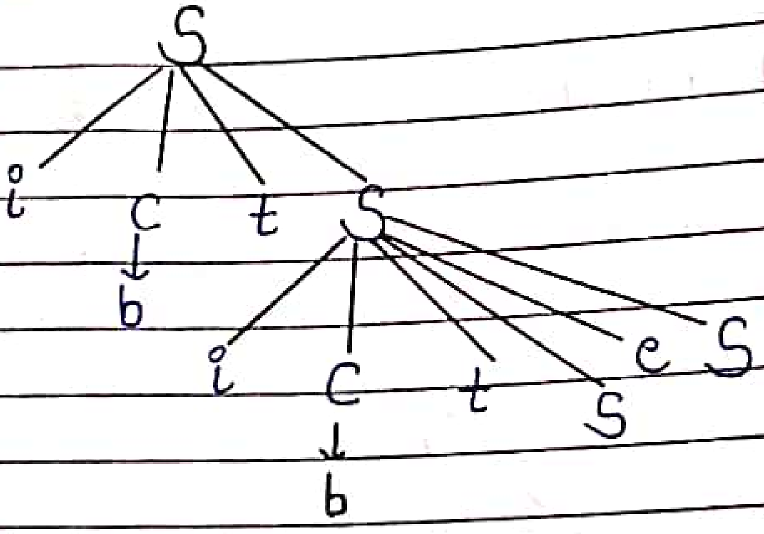
(a) $S \underset{lm}{\Longrightarrow} ictS$



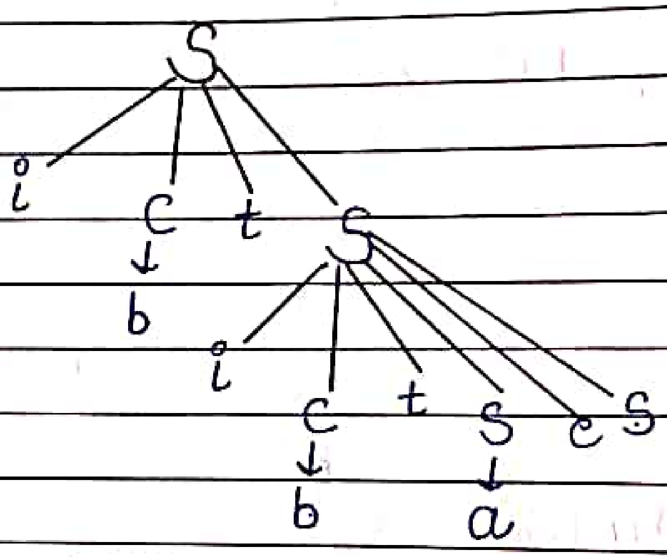
(b) $\underset{lm}{\Longrightarrow} ibtS$



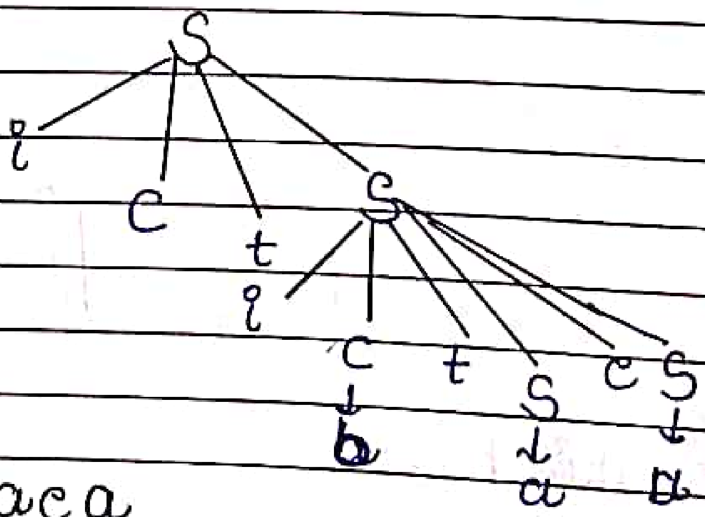
(c) $\underset{lm}{\Longrightarrow} ibtictSeS$



(d) $\xRightarrow[lm]{}$ ibtibtSeS



(e) $\xRightarrow[lm]{}$ ibtibtaeS



(f) $\xRightarrow[lm]{}$ ibtibtaea

• Right most derivation of w= ibtibtaca

$S \underset{Rm}{\Longrightarrow} ictS$

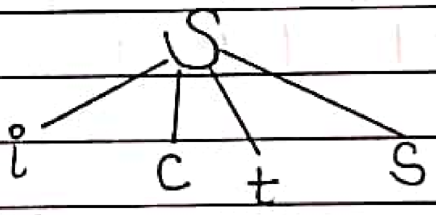
$\underset{Rm}{\Longrightarrow} ictictSeS$

$\underset{Rm}{\Longrightarrow} ictictSea$

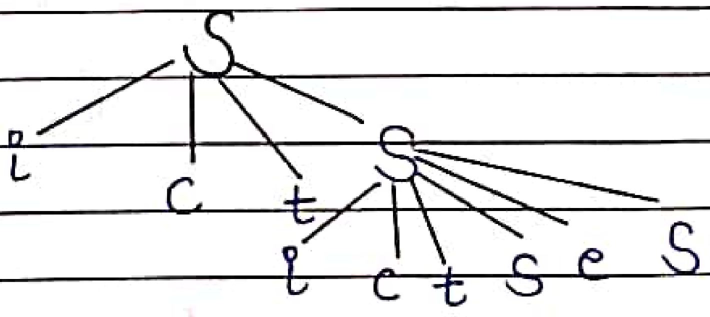
$\underset{Rm}{\Longrightarrow} icticta ea$

$\underset{Rm}{\Longrightarrow} ictibtaea$

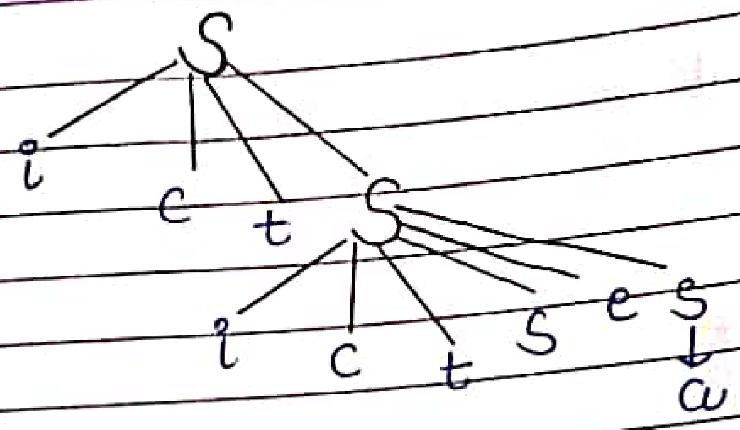
$\underset{Rm}{\Longrightarrow} ibtibtaea$



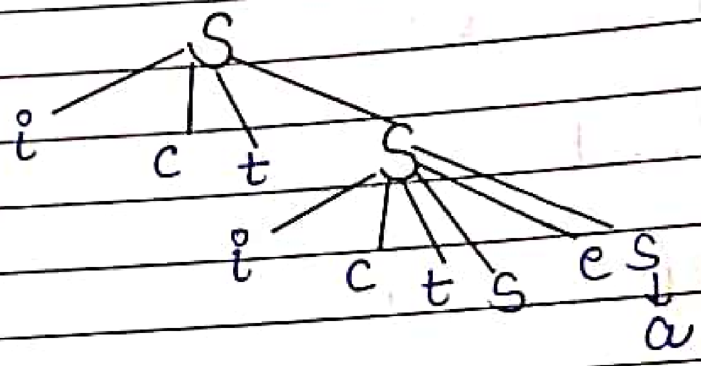
(a) $S \underset{Rm}{\Longrightarrow} ictS$



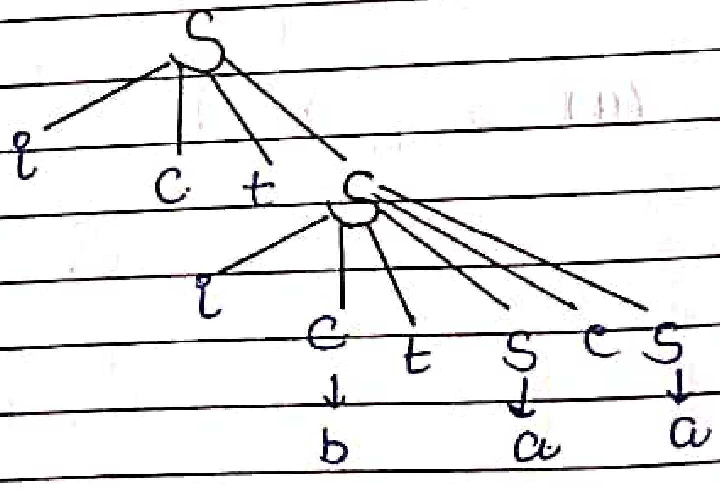
(b) $\underset{Rm}{\Longrightarrow} ictictSeS$



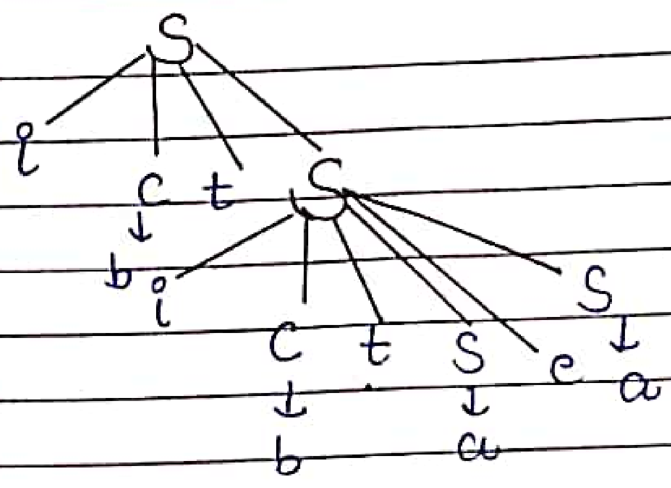
(c) $\xrightarrow[Rm]{}$ ictictsea



(d) $\xrightarrow[Rm]{}$ icticta ea



(e) $\xrightarrow[Rm]{}$ ictibt aea.



(7.) $\underset{Rm}{\Longrightarrow}$ ibtibtaea