

Course: Automata Theory

Lecture 12: Grammar Ambiguity

Lecturer: Martha Gichuki

Course description

- The course begins with an introduction to logic and formal grammar where learners will do a recap on sets, logic and truth tables, sequences, relations and functions
- A coverage of finite state machines, Push Down automata and Turing Machines (The Church's thesis) will culminate the study of various models of computation.
- **Formal language and grammar** will then follow to enable learners differentiate regular and context free languages.
- An evaluation of the computability and complexity of practical computational problems which are the foundations of automata theory will then be done and the outcome will be problem description.

Learning outcomes:

Lecture 12: Grammar Ambiguity

At the end of the lecture the learner will be able to:

- i. Describe ambiguous grammar.
- ii. Derive a string using derivation trees
- iii. Differentiate ambiguous and unambiguous grammar

Context Free Grammar (CFG)

- Recall that PDA recognizes Context Free Languages and these are **built upon** Context Free Grammar.
- In a natural language like English, understanding a sentence begins with understanding the grammatical structure.
- This means knowing how the language is **derived from the grammar rules of the language.**

Context Free Grammar (CFG)...

- Given a Context Free Grammar e.g. one that specifies the syntax of a programming language;
- If a string is derivable from the grammar it is often useful to know a **derivation** because it allows us to interpret the strings correctly.
- A natural way to represent these derivations is by use of **derivation trees**.

Formal Definition of a Context Free Grammar (CFG)

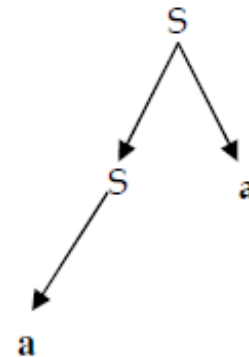
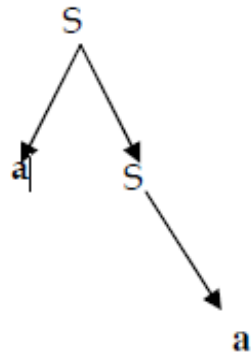
A Context Free Grammar is a 4 – Tuple $\{V, T, S, P\}$, where: -

- V – is a finite set of Variables
- T – is a finite set of Terminals
- S – is the Start Variable
- P – is a finite set of production rules

- A natural way of exhibiting any structure is by deriving it using a **derivation tree**.
- Another name for a derivation tree is a **parse tree**.
- The symbol \rightarrow is used to mean “**Could take the value**”.
- Example, the production rule
 $(S \rightarrow a/b)$ means S could take the value a or b

Derivation trees

- The root of the tree represents the variable with which we begin the derivation (**Start Variable**).
- Each interior node of the tree corresponds to the variables of the grammar.
- The children correspond to the symbols in the entire string appearing on the right side of the production rules



Derivation of Languages:

- This refers to the sequence of rules that produce the finished string of terminals from the start symbol.
- This is called **derivation or production**.
- The language of CFG is the set of terminal symbols which we can derive using specified production rules.

Example 1:

- Given the terminal – a
- Non-terminal – S
- Production rules $S \rightarrow aS$, $S \rightarrow \Lambda$ where Λ (empty string), the derivation for **aaaa** could be given as

$S \rightarrow aS$

$S \rightarrow aaS$

$S \rightarrow aaaS$

$S \rightarrow aaaaS$

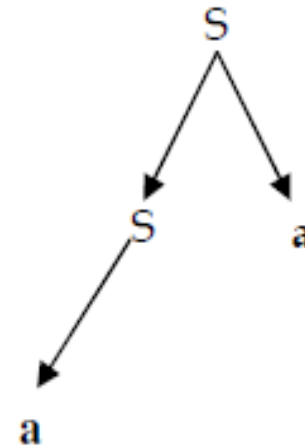
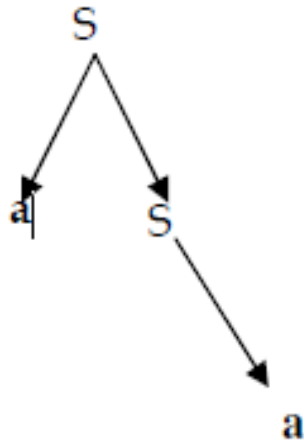
$S \rightarrow aaaaa\Lambda \rightarrow aaaaa$

Example 2:

- Given the terminal a;
- Non – terminal S
- Productions: $S \rightarrow aS / Sa / a$
- Derive the Tree for the string “aa”

Solution

- The word “aa” can be generated using two parse trees: -



Example 3:

- Consider a context free grammar that defines the language **pal of all palindromes** over the alphabet $\Sigma \{a, b\}$ using the following production rules:-
 - i. $(S \rightarrow \Lambda)$ - S could take the null value
 - ii. $(S \rightarrow a/b)$ - S could take the value a or b
 - iii. $(S \rightarrow aSa/bSb)$ - S could take the form aSa or bSb

Example 3: ...

- From these rules we can write $S \rightarrow aSa \rightarrow abSba \rightarrow ab\wedge ba \rightarrow abba$ (which are all palindromes!)
- In an expression such as aSa or bSb , the two alternatives are aSa & bSb and not a & b direct.

- To check whether the given **grammar is ambiguous or not-**

$S \rightarrow SS$

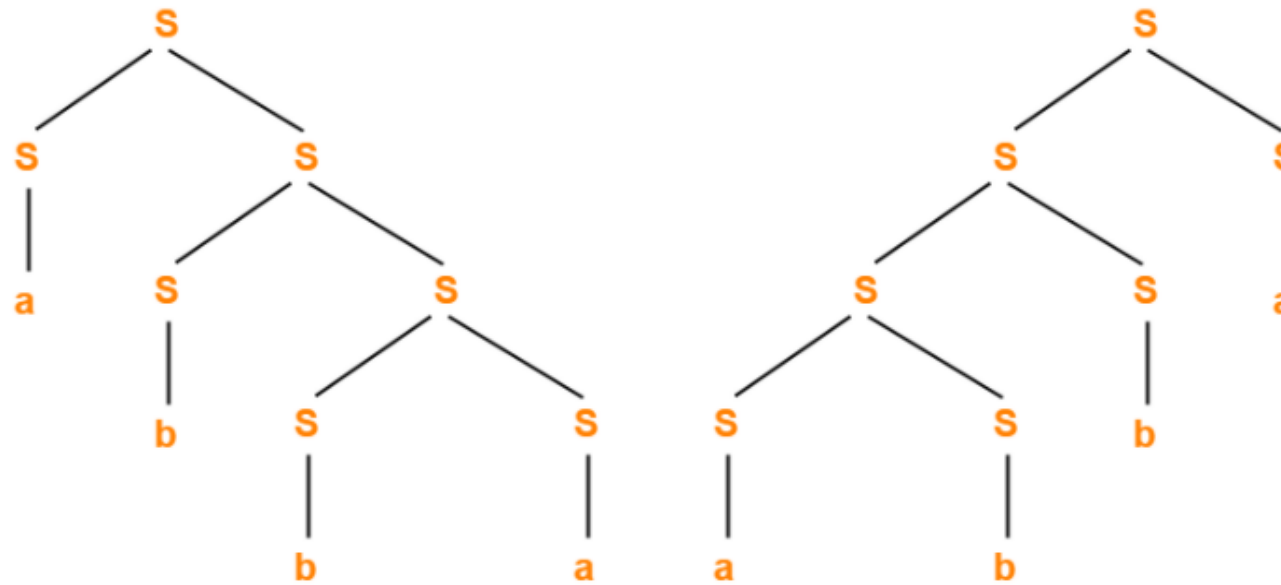
$S \rightarrow a$

$S \rightarrow b$

- Let us consider a string w generated by the given grammar-

$w = abba$.

Since two different parse trees exist for string w , therefore the given grammar is ambiguous.



Parse tree-01

Parse tree-02

Example 4: Left and Right most Derivation

- Given the string “a + a + a” with the following production rules: -
 - (i.) $S \rightarrow S+S$
 - (ii.) $S \rightarrow a$
- The **Left-most derivation tree** for some string is the process that looks at the string from Left to Right following the production rules that are provided.
- The left most derivation takes the format: -
$$S \rightarrow S+S \rightarrow a+S \rightarrow a+S+S \rightarrow a+a+S \rightarrow a+a+a = \mathbf{a+(a+a)}$$

Right-most derivation

- Given the string “a + a + a” with the following production rules: -

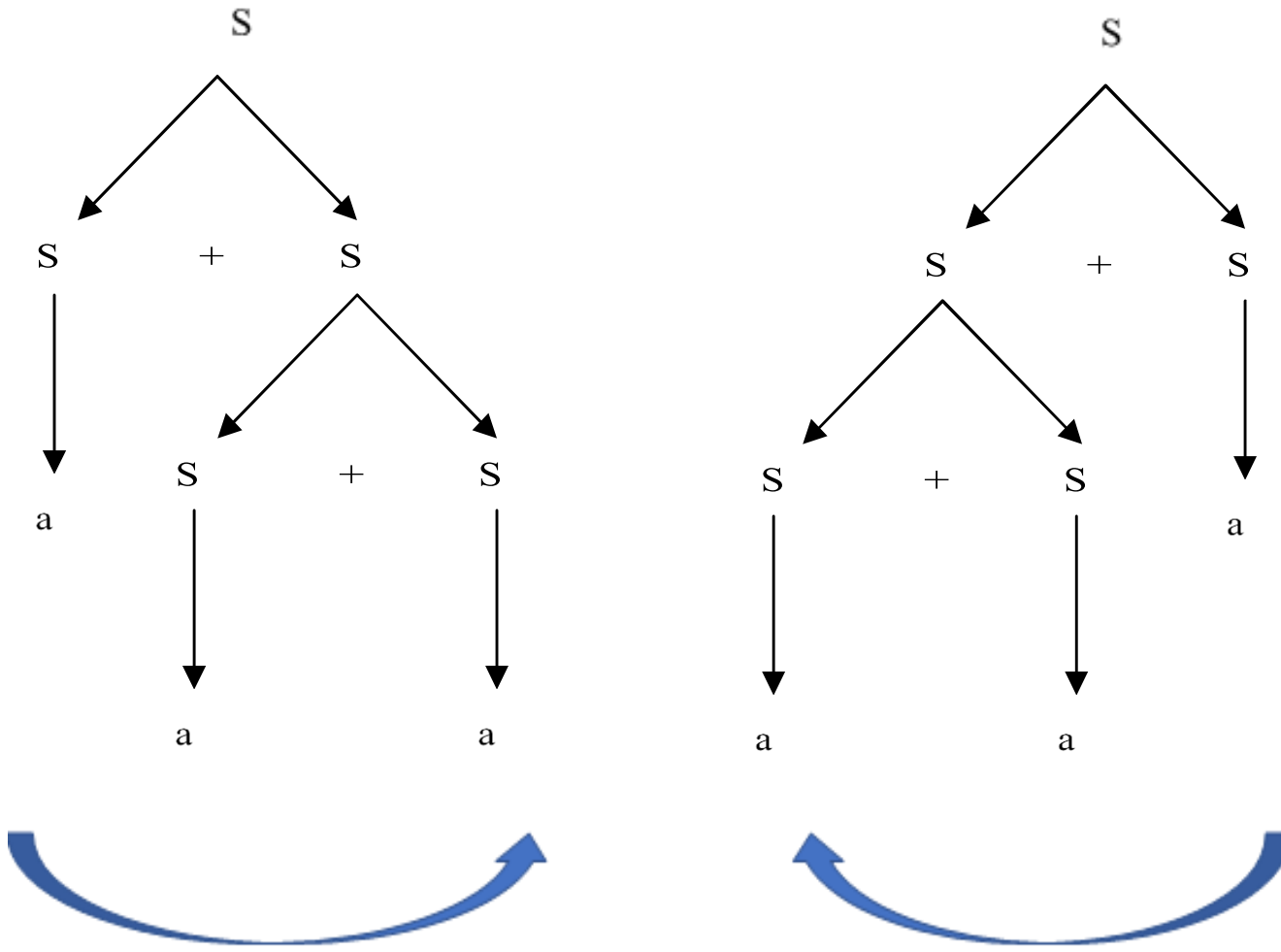
$$(i.) \quad S \rightarrow S+S$$

$$(ii.) \quad S \rightarrow a$$

- Right-most derivation looks at the string from Right to left.

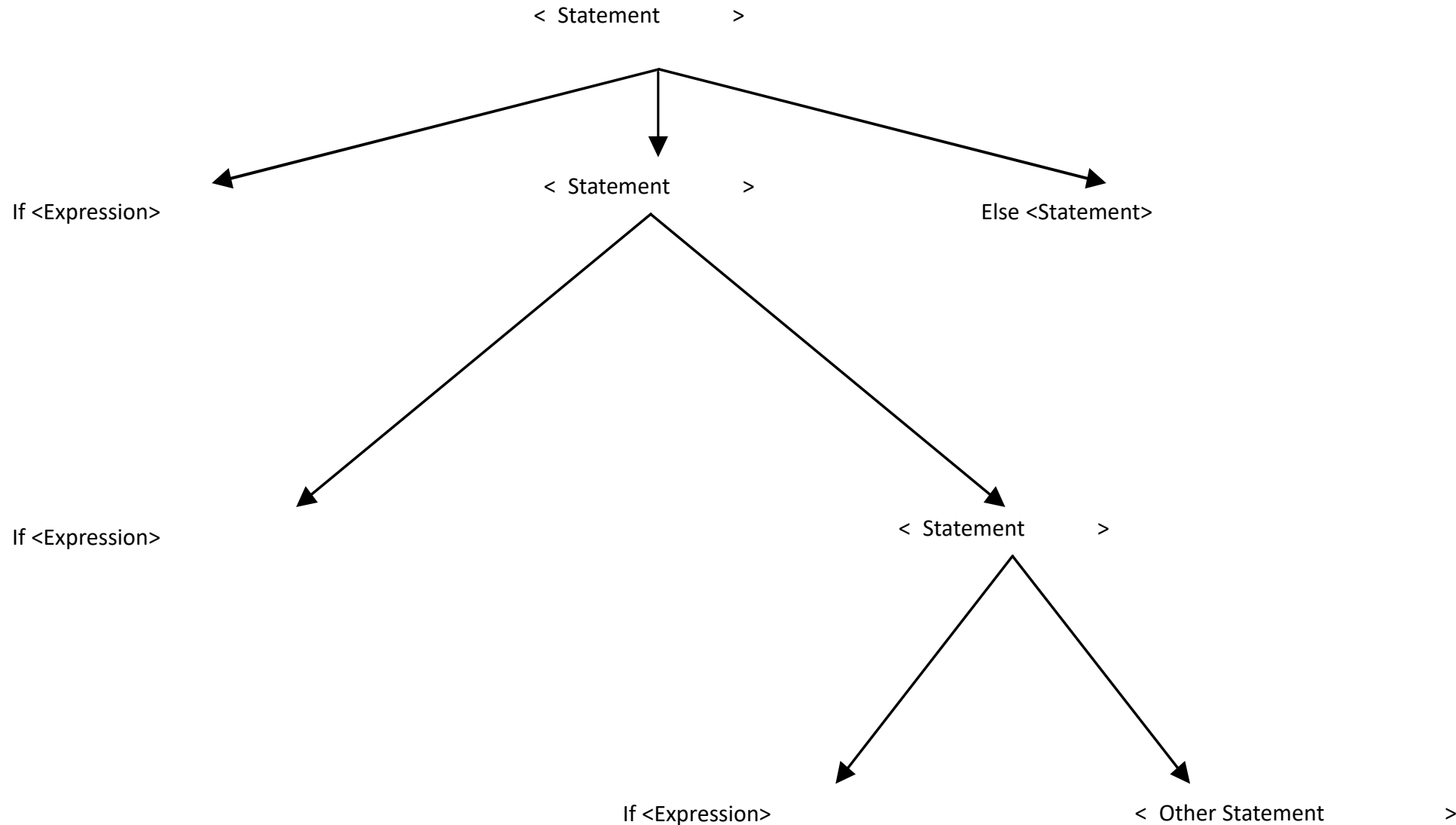
$$S \rightarrow S+S \rightarrow S+S+S \rightarrow a+S+S \rightarrow a+a+S \rightarrow a+a+a = \mathbf{(a+a)+a}$$

- We can exhibit this structure by using left - $a+(a+a)$ and right - $(a+a) + a$ derivation trees as follows: -



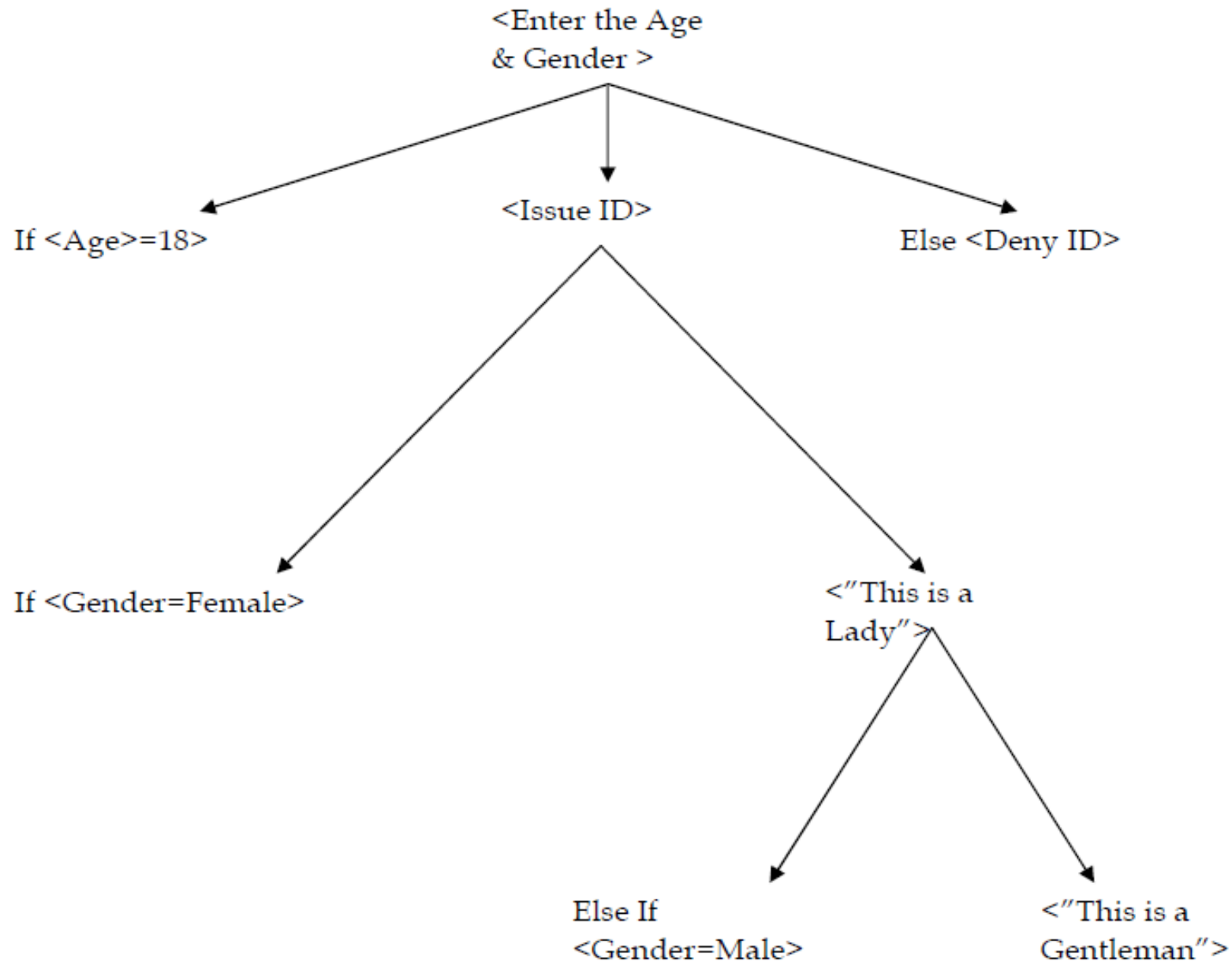
- A CFG is said to be **ambiguous** if there is at least one **single string** having **two or more distinct derivation trees**. The above CFG is ambiguous because it generates two distinct derivation trees. (Note: There could be more).

- A standard example of ambiguity is the If Else Statements in Programming languages.



Consider implementing the following problem in a Visual Basic programming language

- If Gender= M then **male** title is selected and if age ≥ 18 then **Gentleman** Title is selected otherwise **Master** title is picked,
- If Gender = F then **Female** title is selected and if age ≥ 18 then **Lady** Title is selected else **Miss** Title is selected.
- Otherwise Unknown option is selected in which **"??"** title sign is displayed



- We can use a derivation tree to illustrate how a word is derived from a CFG and the trees are known as syntax/parse/derivation trees (parse generations).

Grammar Ambiguity

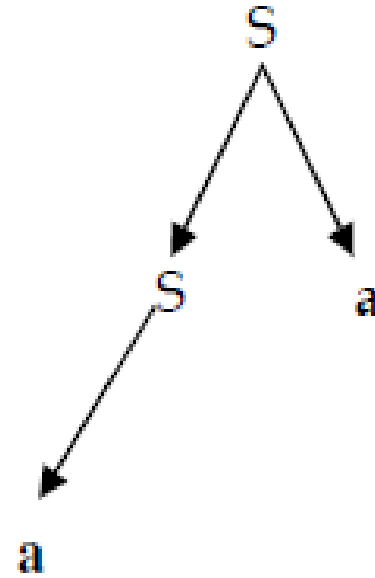
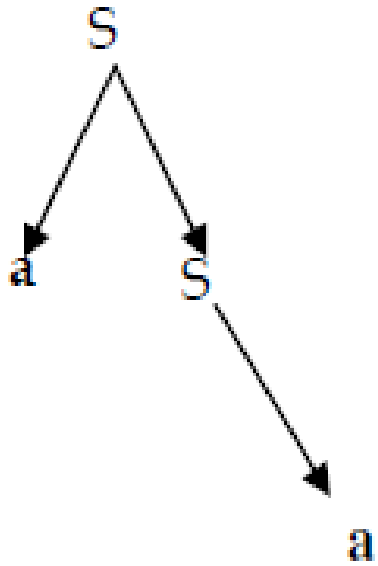
- A CFG is ambiguous if there's at least one word in its CFG with two possible derivations corresponding to two different trees.

Example 6:

Consider our earlier example where given the

- Terminal a ;
- Non – terminal S
- Production rules: $S \rightarrow aS / Sa/a$

To derive the tree for the string “aa” we have two solutions:-



- Therefore, the CFG is ambiguous.
- For any valid derivation tree, reading the leaves from right to left gives one string in the language defined by the grammar.

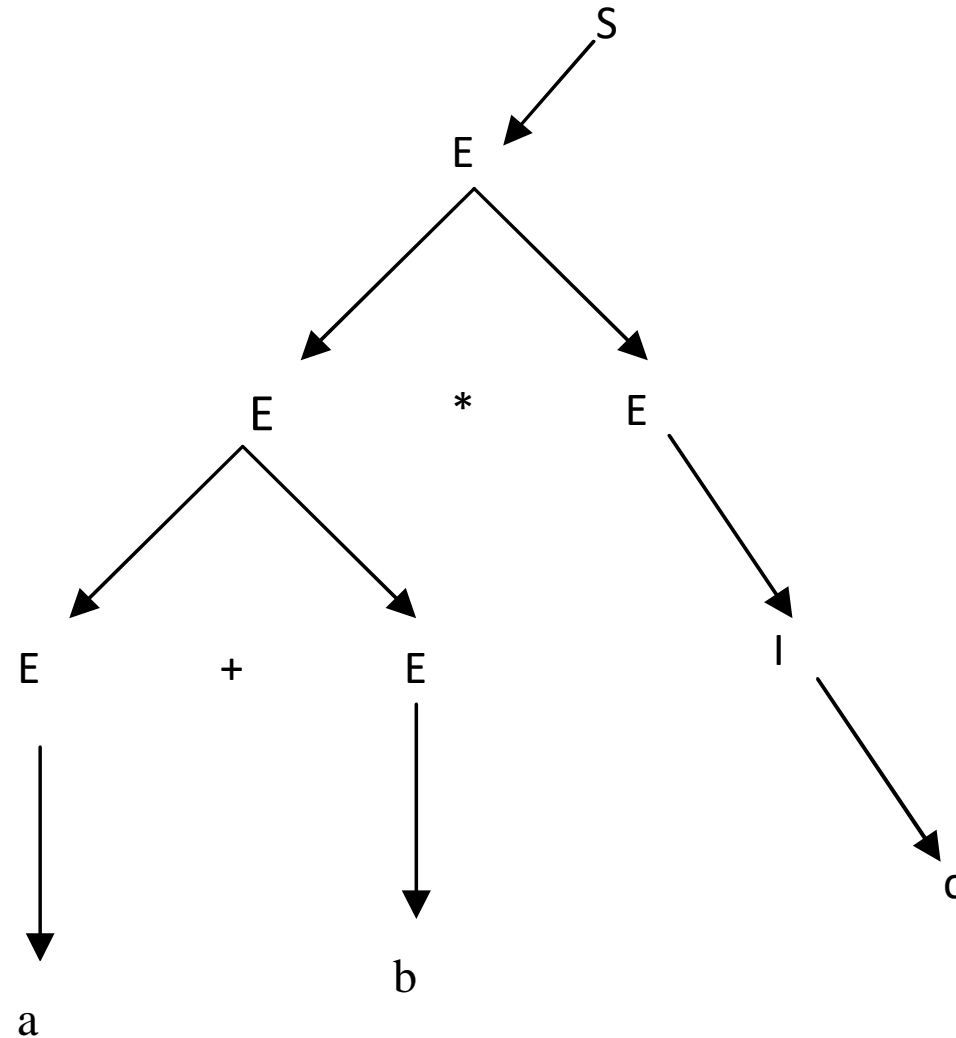
Example 5:

Given a CFG $G = \{V, T, P, S\}$; where

- $V = \{S, E, I\}$
- $T = \{a, b, c\}$
- $S = \text{Start Variable} = S$
- $P = \text{Production rules:}$
 - i. $I \rightarrow a/b/c$
 - ii. $E \rightarrow I/E+E/E^*E$
 - iii. $S \rightarrow E$
- Given the string $a + b * c$, draw a derivation tree for the string.

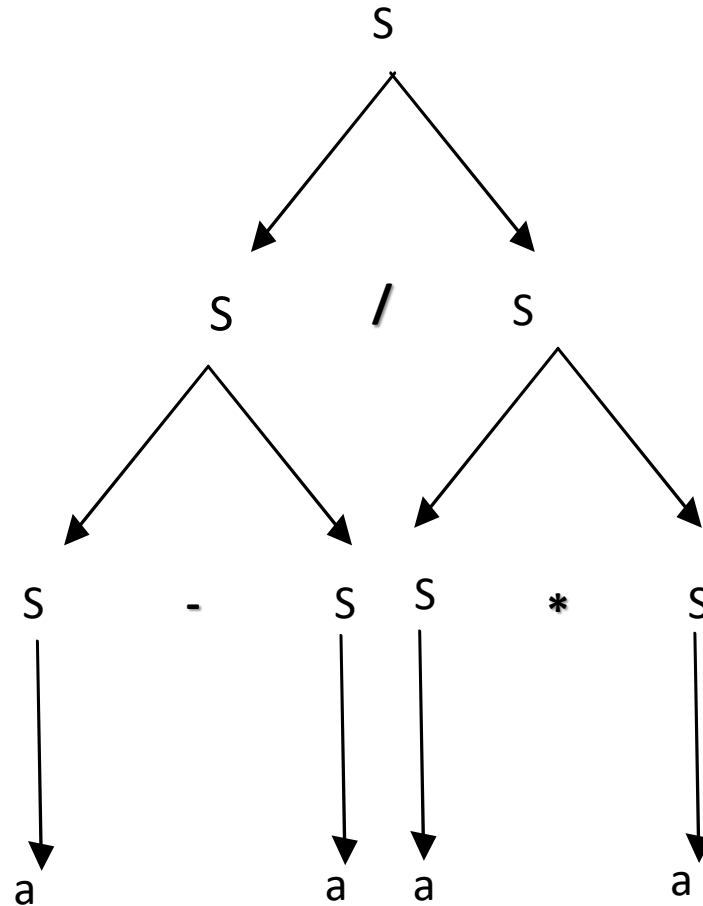
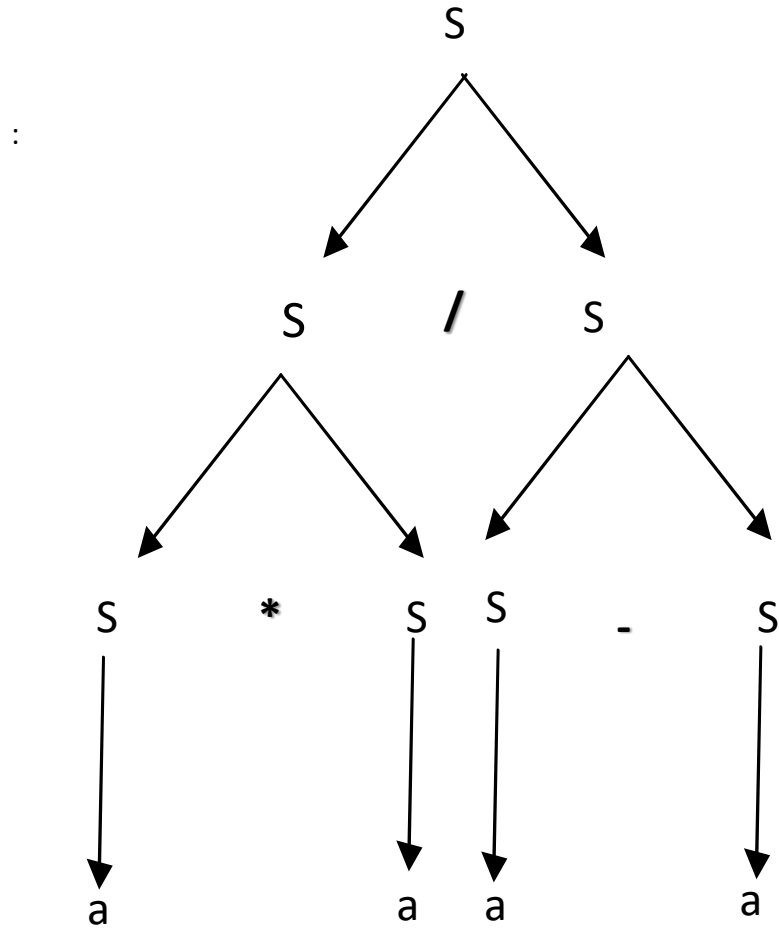
Given the string $a + b * c$, draw a derivation tree for the string.

• **Solution**



Review Question: One can use the following production to obtain a context free grammar: - $S \rightarrow (S + S) / (S - S) / (S * S) / (S / S) / a$.

- Proof that $(a * a) / a - a$; is ambiguous using the same derivation.



- Consider the grammar: $S \rightarrow bB/aA$, $A \rightarrow b/bS/aAA$, $B \rightarrow a/aS/bBB$
- For the string $w = bbaababa$ find leftmost derivation, right most derivation and parse tree

- Leftmost derivation

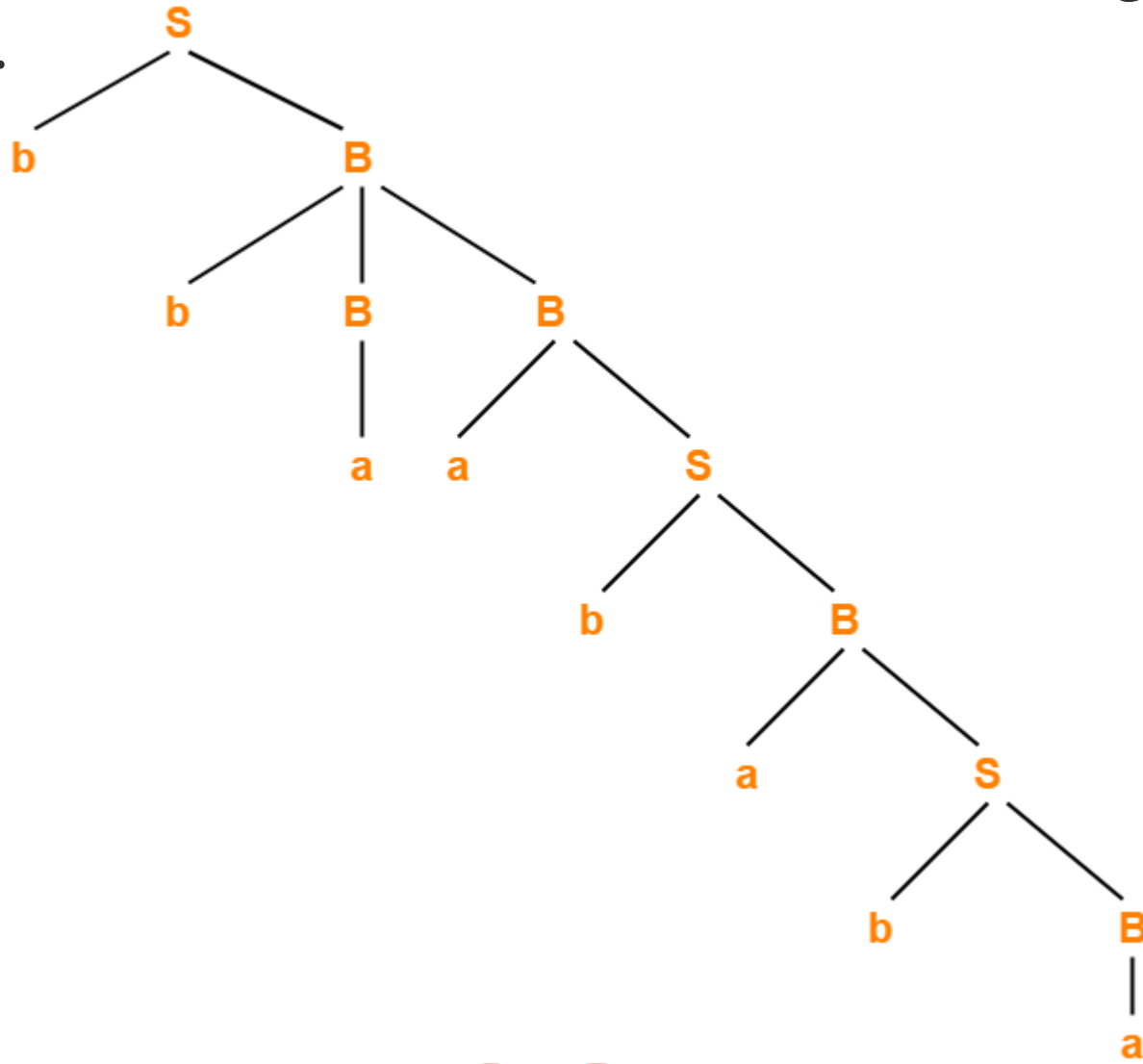
$S \rightarrow bB$
 $\rightarrow bbBB$ (Using $B \rightarrow bBB$)
 $\rightarrow bbaB$ (Using $B \rightarrow a$)
 $\rightarrow bbaaS$ (Using $B \rightarrow aS$)
 $\rightarrow bbaabB$ (Using $S \rightarrow bB$)
 $\rightarrow bbaabaS$ (Using $B \rightarrow aS$)
 $\rightarrow bbaababB$ (Using $S \rightarrow bB$)
 $\rightarrow bbaababa$ (Using $B \rightarrow a$)

- Rightmost derivation

$S \rightarrow bB$
 $\rightarrow bbBB$ (Using $B \rightarrow bBB$)
 $\rightarrow bbBaS$ (Using $B \rightarrow aS$)
 $\rightarrow bbBabB$ (Using $S \rightarrow bB$)
 $\rightarrow bbBabaS$ (Using $B \rightarrow aS$)
 $\rightarrow bbBababB$ (Using $S \rightarrow bB$)
 $\rightarrow bbBababa$ (Using $B \rightarrow a$)
 $\rightarrow bbaababa$ (Using $B \rightarrow a$)

Parse Tree

• Whether we consider the leftmost derivation or rightmost derivation, we get the parse tree shown below. The reason is given grammar is unambiguous.



References

- Rowan G. & John T., (2009), *Discrete Mathematics: Proofs, Structures and Applications*, CRC Press, ISBN: 9781439812808.
- W. D. Wallis (2003), *A Beginners Guide to Discrete Mathematics*, Springer Science & Business Media, ISBN: 978-0817642693.
- Introduction to the theory of computation (3rd ed.), Michael, S. Boston, Cengage Learning. ISBN-13: 978-1133187790, (2012).
- Introduction to languages and the theory of computation (3rd ed.), Martin, J., New York: McGraw-Hill. ISBN-13: 978-0072322002, (2002)