Automata Theory - Lecture 12

Grammar and Ambiguity

Lecturer: Martha Gichuki

Lecture learning outcomes

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

(i) Describe ambiguous grammar.

(ii) Derive a string using derivation trees

(iii) Differentiate ambiguous and unambiguous grammar

Introduction:

An automaton is always anchored on the basic concepts of **symbols, words alphabets and strings.** These are defined briefly as follows: -

Symbol – A character or letter (An arbitrary datum) that has some meaning to or effect on the machine. Symbols are sometimes called letters.

Word – A finite string formed by the concatenation of a number of symbols.

Alphabet – A finite set of symbols. It is frequently denoted byΣ, which is the set of letters in an alphabet.

Language - refers to a set of words formed by symbols in a given alphabet.

Kleene Closure – A language may be thought of as a sub-set of possible words. The set of all possible words may in turn be thought of as the set of all possible concatenations of strings. Formally, this set of all possible strings is called a free monoid. It is denoted as ***** and the super script $^{(*)}$ is called the **Kleene Star**.

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 meaning knowing how it is derived from the grammar rules of the language.

Given a Context Free Grammar e.g. one specifying the syntax of a programming language 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 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 → is used to mean "Could take the value".

Example, the production rule (S → a/b) means S could take the value a or b

- 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 is 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.

Ambiguity

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

Example 1:

- Given the terminal – a
- Non-terminal – S
- Production rules S→aS, S→∧ where ∧ (empty string), the derivation for **aaaa** could be given as

i. S→aS
ii. S→aaS
iii. S→aaaS
iv. S→aaaaS
v. S→aaaa ∧→aaaa

Example 2:

Given the terminal a;

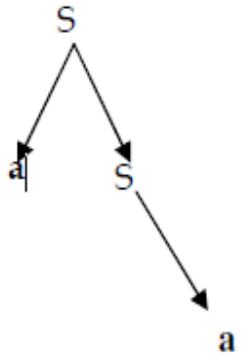
Non – terminal S

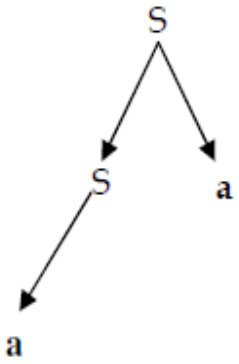
Productions: S → aS / Sa / a

Derive the Tree for the string "aa"

Solution

The word aa can be generated using two trees: -





Example 3:

Consider a context free grammar that defines the language **pal of all palindromes** over the alphabet ∑ {a, b} using the following production rules:-

i. (S→∧) - S could take the null value

ii. (S → a/b) - S could take the value a or b

iii. (S → aSa/bSb) - S could take the form aSa or bSb

From these rules we can write S → aSa→abSba→ab∧ba →abba (which are all palindromes!)

In an expression such as aSa or bSb, the two alternatives are aSa & bSb and not a & b direct.

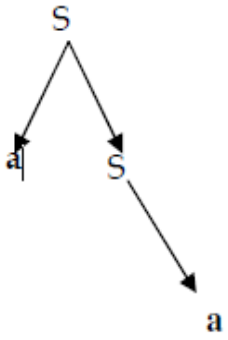
Example 4:

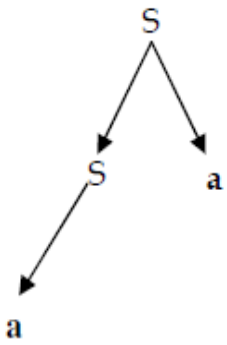
Consider our earlier example where given the terminal a;

Non – terminal S

Production rules: S → 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.

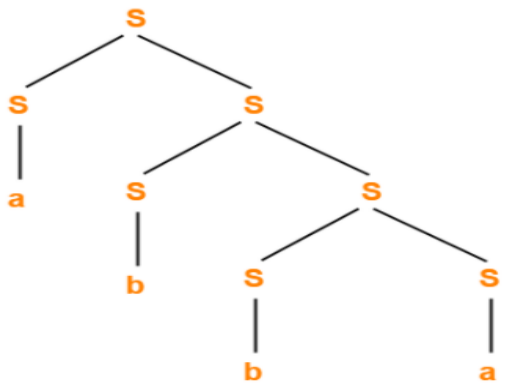
Example 5:

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

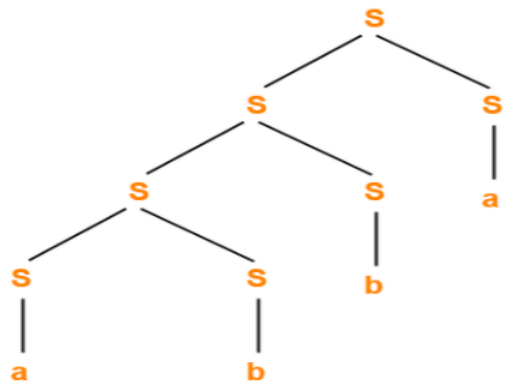
i. $S \rightarrow SS$
ii. $S \rightarrow a$
iii. $S \rightarrow b$

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 6:

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

V= {S, E, I}

T = {a, b, c}

S = Start Variable =S

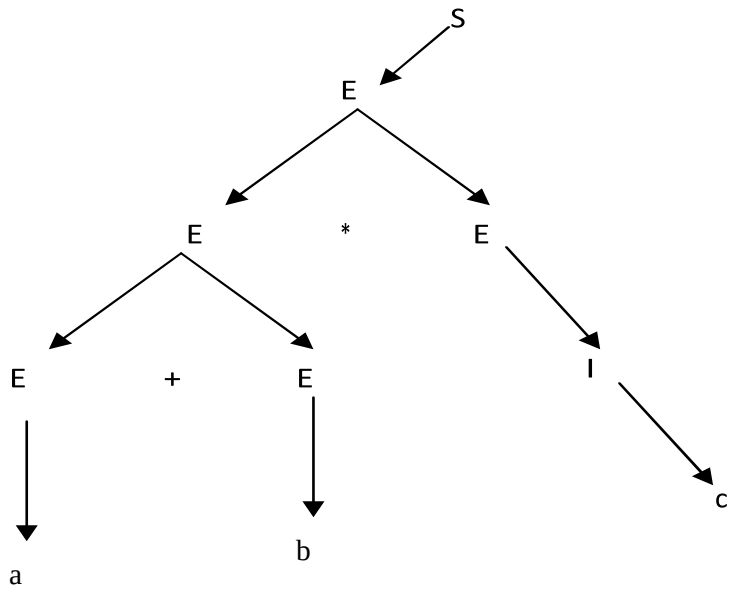
P= Production rules: = $I \rightarrow a/b/c$

$E \rightarrow I/E+E/E*E$

$S \rightarrow E$

Given the string a + b * c, come up with a derivation tree for the string.

:



Example 7:

Consider the grammar: S-bB/aA, A-b/bS/aAA, B-a/aS/bBB

For the string w=bbaababa find leftmost derivation, right most derivation and parse tree

Leftmost derivation

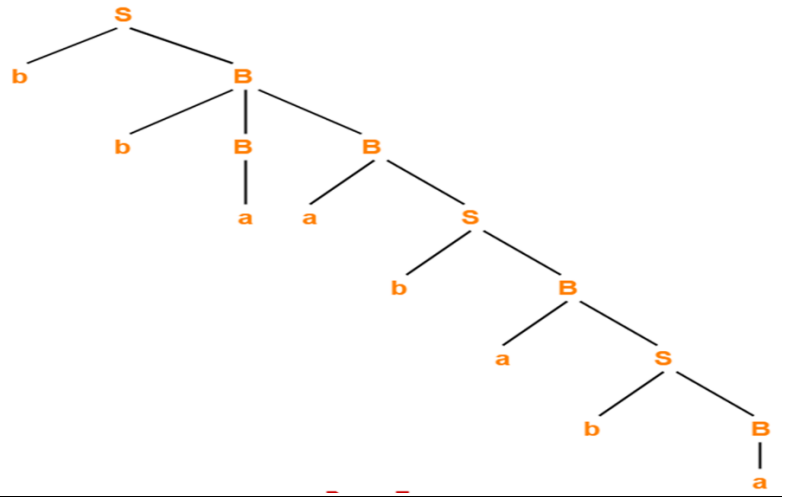
S → bB
→ bbBB (Using B → bBB)
→ bbaB (Using B → a)
→ bbaaS (Using B → aS)
→ bbaabB (Using S → bB)
→ bbaabaS (Using B → aS)
→ bbaababB (Using S → bB)
→ bbaababa (Using B → a)

Rightmost derivation

S → bB
→ bbBB (Using B → bBB)
→ bbBaS (Using B → aS)
→ bbBabB (Using S → bB)
→ bbBabaS (Using B → aS)
→ bbBababB (Using S → bB)
→ bbBababa (Using B → a)
→ bbaababa (Using B → 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.



Example 8: 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: -

$S \rightarrow S+S \rightarrow a+S \rightarrow a+S+S \rightarrow a+a+S \rightarrow a+a+a = a+(a+a)$

Right-most derivation

Given the string "a + a + a" with the following production rules: -

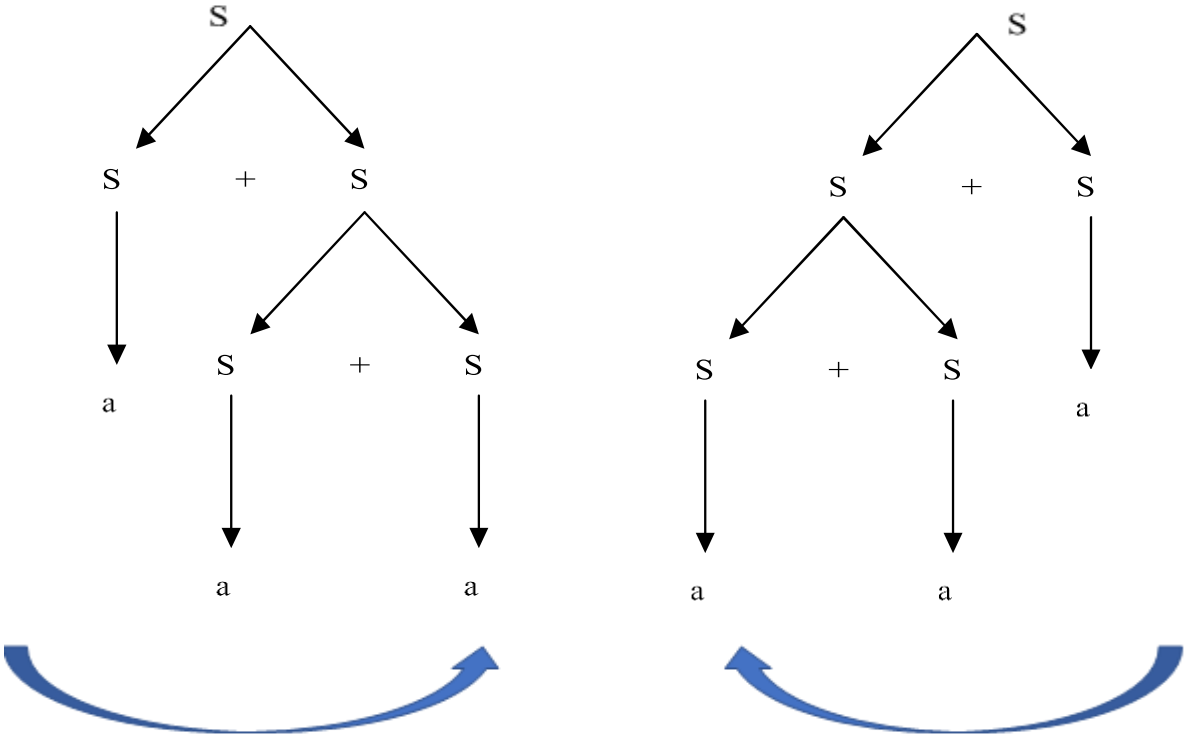
(i.) $S \rightarrow S+S$

(ii.) $S \rightarrow a$

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

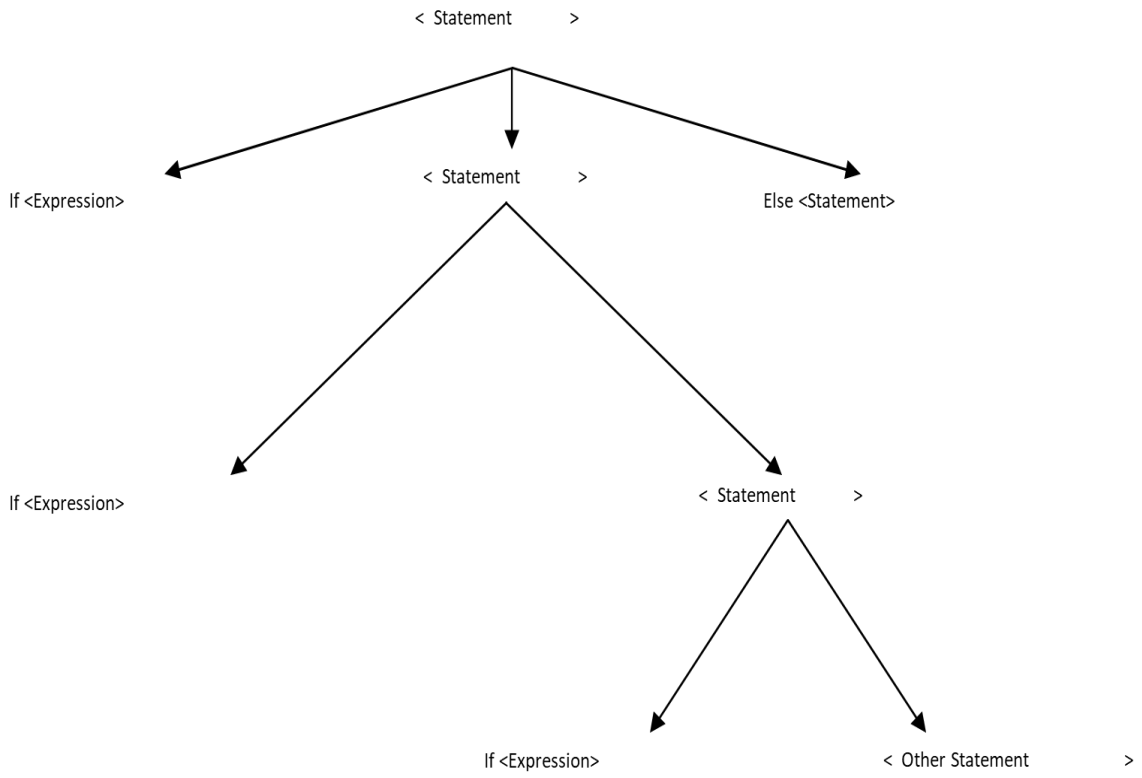
$S \rightarrow S+S \rightarrow S+S+S \rightarrow a+S+S \rightarrow a+a+S \rightarrow a+a+a = (a+a)+a)$

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.

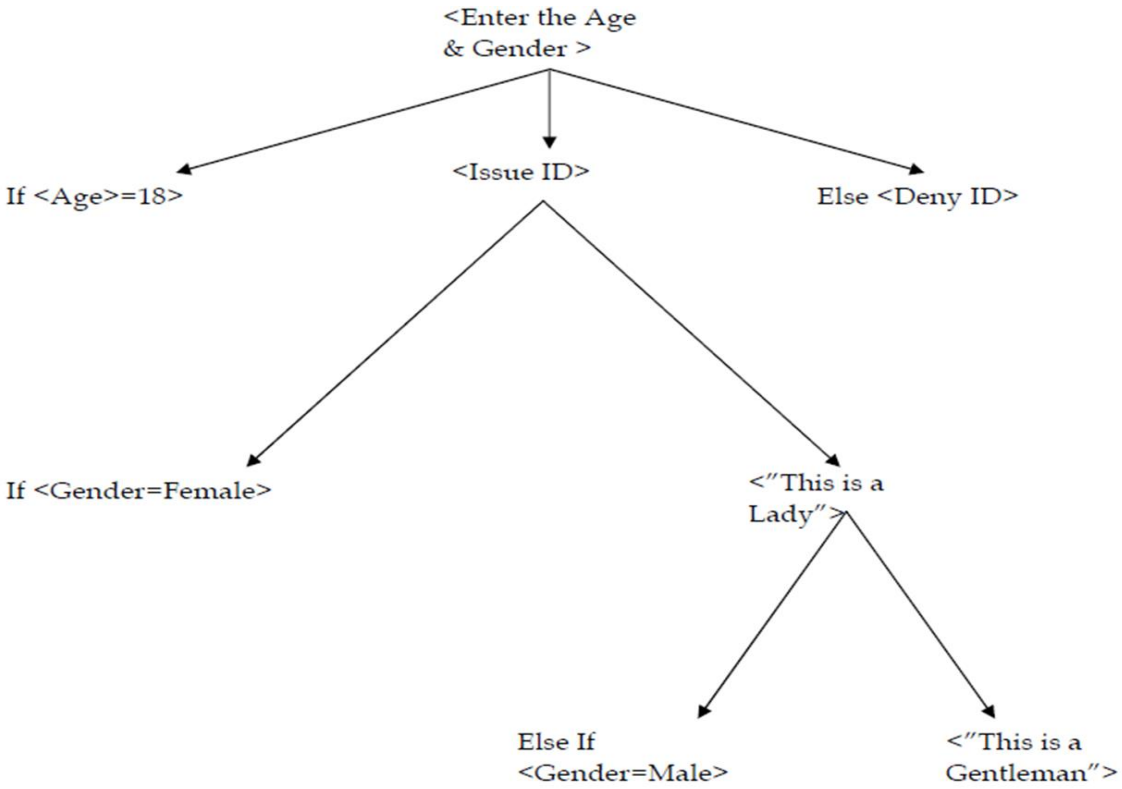


Consider implementing the following problem in a Visual Basic programming language

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



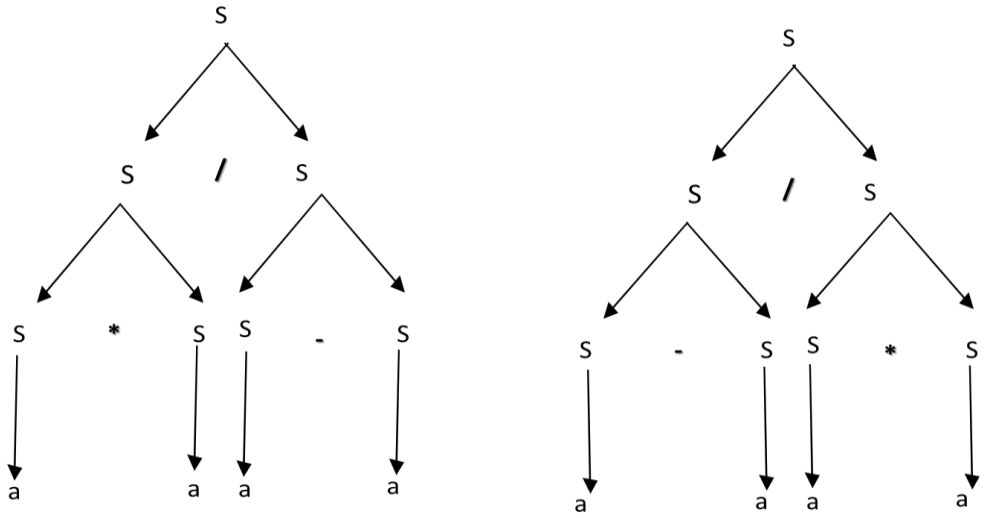
We can use a derivation tree to illustrate how a word is derived from a CFG and the trees are known as syntax/parse/derivation tress (parse generations).

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

Review Question:

One can use the following production to obtain a context free grammar: -

S-> (S + S) / (S – S)/(S*S)/ (S/S)/a. Proof that (a* a)/a – a; is ambiguous using the same derivation.



References

1 Rowan G. & John T., (2009), *Discrete Mathematics: Proofs, Structures and Applications*, CRC Press, ISBN: 9781439812808.

2 W. D. Wallis (2003), *A Beginners Guide to Discrete Mathematics*, Springer Science & Business Media, ISBN: 978-0817642693.

3 Introduction to the theory of computation (3rd ed.), Michael, S. Boston, Cengage Learning. ISBN-13: 978-1133187790, (2012).

4 Introduction to languages and the theory of computation (3rd ed.), Martin, J., New York: McGraw-Hill. ISBN-13: 978-0072322002, (2002)