Theory of Automata Context Free Grammars

Hafiz Tayyeb Javed

Week 10- Lecture 1

Contents

Trees

- Consider the following CFG:

  $S \rightarrow AA$

  $A \rightarrow AAA|bA|Ab|a$

- The derivation of the word *bbaaaab* is as follows:

  $S \Rightarrow AA \Rightarrow bAAAA \Rightarrow bbAaaAb \Rightarrow bbaaaab$

$$S \rightarrow AA$$

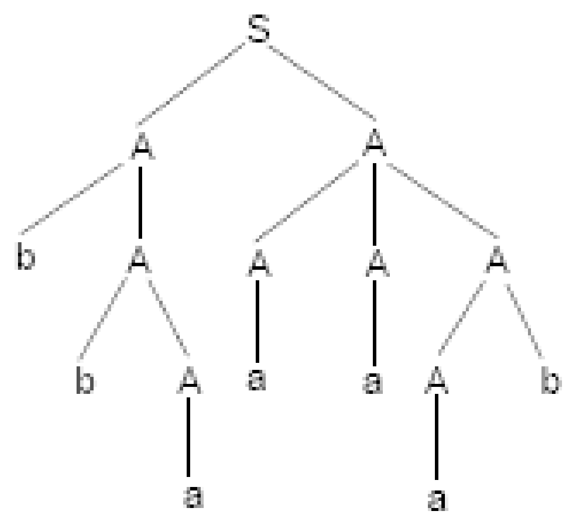
$$A \rightarrow AAA|bA|Ab|a$$

The derivation of the word *bbaaaab* is as follows:

S ➔ AA ➔ bAAAA ➔ bbAaaAb ➔ bbaaaab

- We can use a tree diagram to show that derivation process:

We start with the symbol S**. Every time we use a production to replace a non-terminal by a string, we d[...] the non-terminal to EACH character in the string.**



- Reading from left to right produces the word bbaaaab.
- Tree diagrams are also called **syntax trees**, **parse trees**, **generation trees**, **production trees**, or **derivation trees.**

Lukasiewicz Notation - Example

- Also called the polish prefix notation.
- A parenthesis free notation
- Consider the following CFG for a simplified version of arithmetic expressions:

  S → S + S | S * S | number

  where the only non-terminal is S, and the terminals are number together with the symbols +,* .

- Obviously, the expression 3 + 4 * 5 is a word in the language defined by this CFG; however, it is ambiguous since it is not clear whether it means (3 + 4) * 5 (which is 35), or 3 + (4 * 5) (which is 23).
- To avoid ambiguity, we often need to use parentheses, or adopt the convention of "hierarchy of operators" (i.e., * is to be executed before +).
- We now present a new notation that is unambiguous but does not rely on operator hierarchy or on the use of parentheses.

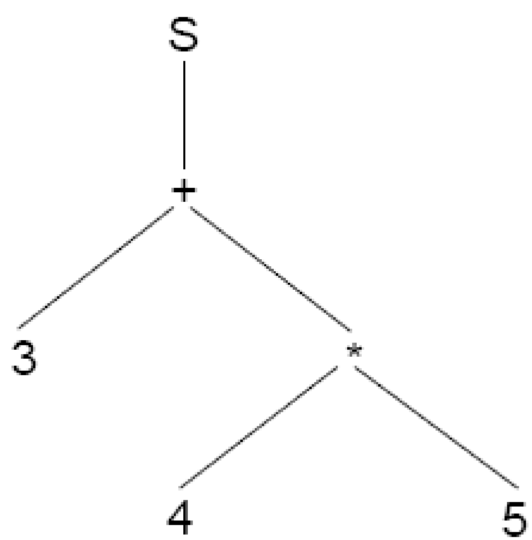
- Let us define a new CFG in which S, $+$, and $*$ are nonterminals and <u>number</u> is the only terminal. The productions are

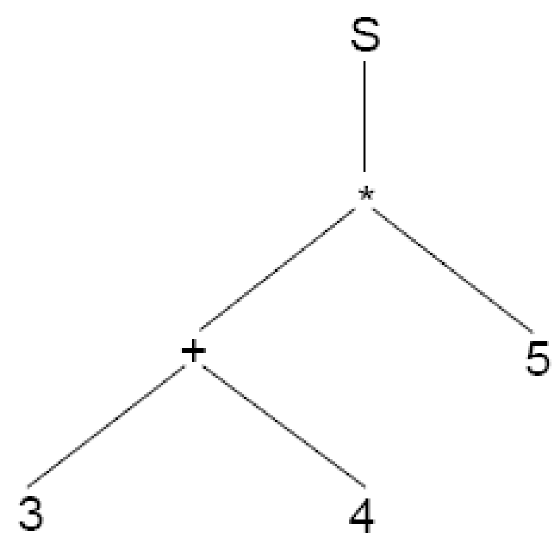
$$S \rightarrow * \mid + \mid \underline{\text{number}}$$

$$+ \rightarrow + + \mid + * \mid + \ \underline{\text{number}} \mid * + \mid * * \mid * \ \underline{\text{number}} \mid \underline{\text{number}} \ + \mid \underline{\text{number}} \ * \mid \underline{\text{number}} \ \underline{\text{number}}$$

$$* \rightarrow + + \mid + * \mid + \ \underline{\text{number}} \mid * + \mid * * \mid * \ \underline{\text{number}} \mid \underline{\text{number}} \ + \mid \underline{\text{number}} \ * \mid \underline{\text{number}} \ \underline{\text{number}}$$

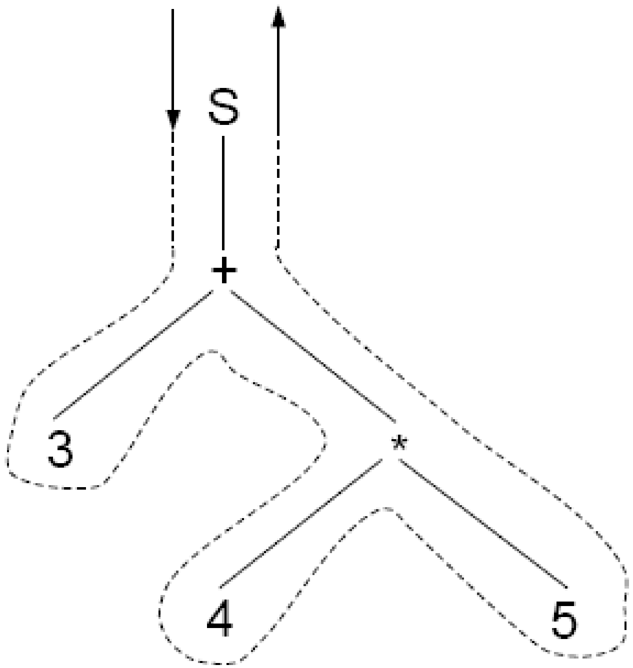
- Let us draw the derivation tree for the expression $3 + (4 * 5)$ and $(3 + 4) * 5$ respectively, using the new CFG above.



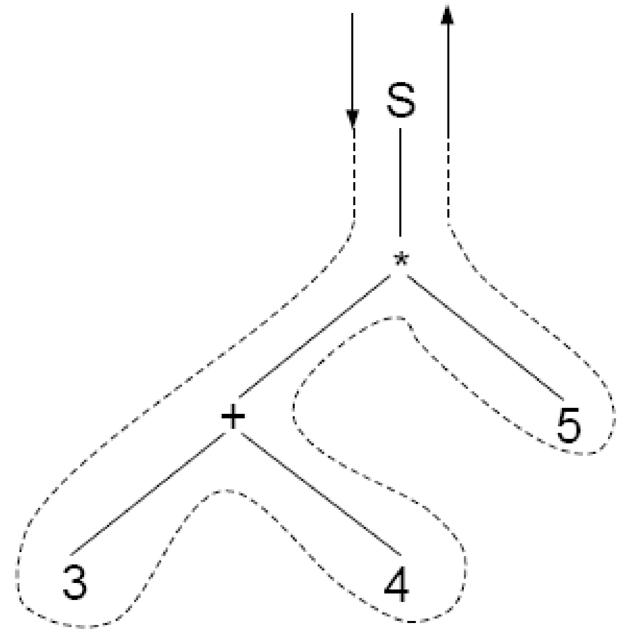


New Notation: Lukasiewicz notation

- We can now construct a **new notation** for arithmetic expressions:
  - We walk around the tree and write down symbols, **once each**, as we encounter them.
  - We begin on the left side of the start symbol S and head south.
  - As we walk around the tree, we always keep our left hand on the tree.



+ 3 * 4 5



* + 3 4 5

- Using the algorithm above, the first derivation tree is converted into the notation: + 3 * 4 5.
- The second derivation tree is converted into * + 3 4 5.

Example

- Consider the expression: + 3 * 4 5:

String	**First o-o-o**
$+ \ 3 \ * \ 4 \ 5$	$* \ 4 \ 5$
$\Downarrow$	
$+ \ 3 \ 20$	$+ \ 3 \ 20$
$\Downarrow$	
23	

- Consider the second expression: * + 3 4 5:

String	**First o-o-o**
$* \ + \ 3 \ 4 \ 5$	$+ \ 3 \ 4$
$\Downarrow$	
$* \ 7 \ 5$	$* \ 7 \ 5$
$\Downarrow$	
35	

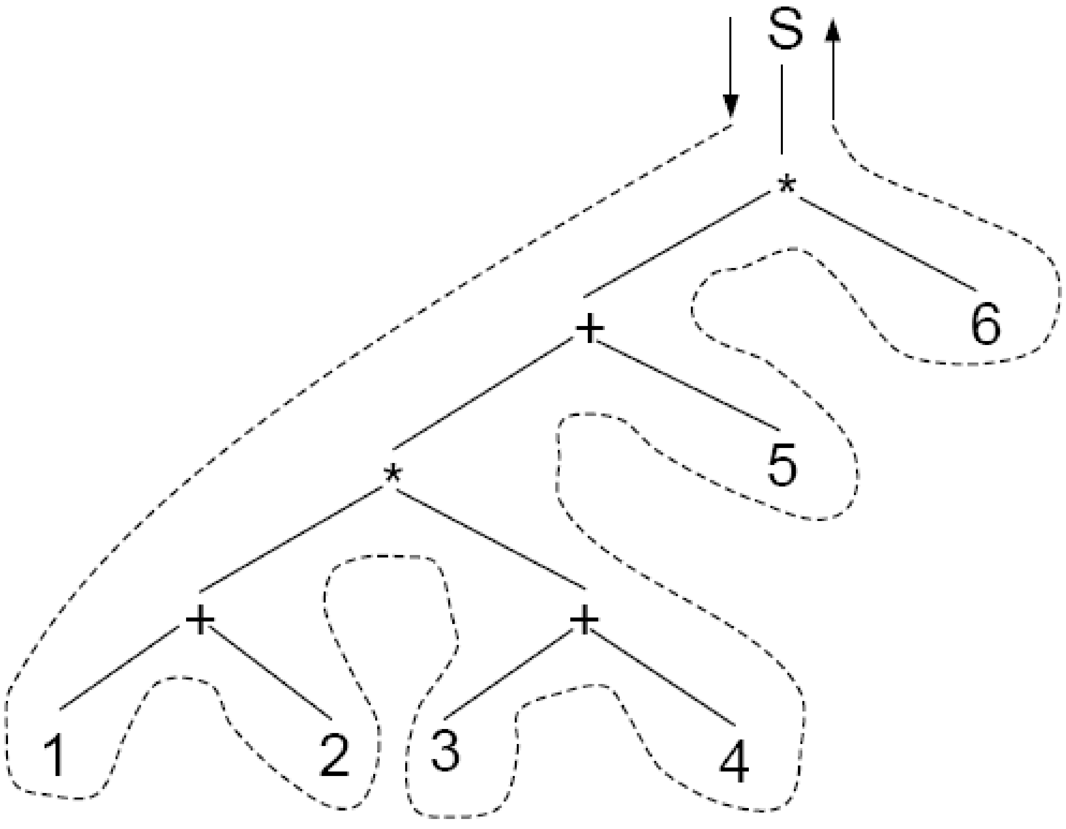
Example

- Convert the following arithmetic expression into operator prefix notation:

  ((1 + 2) * (3 + 4) + 5) * 6.

- This normal notation is called **operator infix notation**, with which we need parentheses to avoid ambiguity.

- Let’s us draw the derivation tree:



- Reading around the tree gives the equivalent prefix notation expression:
- * + * + 1 2 + 3 4 5 6.

Evaluate the String

String	**First o-o-o**
$* \; + \; * \; + \; 1 \; 2 \; + \; 3 \; 4 \; 5 \; 6$	$+ \; 1 \; 2$
$\Downarrow$	
$* \; + \; * \, 3 \; + \; 3 \; 4 \; 5 \; 6$	$+ \; 3 \; 4$
$\Downarrow$	
$* \; + \; * \, 3 \; 7 \; 5 \; 6$	$* \, 3 \; 7$
$\Downarrow$	
$* \; + \; 21 \; 5 \; 6$	$+ \; 21 \; 5$
$\Downarrow$	
$* \, 26 \; 6$	$* \, 26 \; 6$
$\Downarrow$	
156	

- This operator prefix notation was invented by Lukasiewicz (1878 - 1956) and is often called Polish notation.

- There is a similar **operator postfix notation** (also called Polish notation), in which the operation symbols (+, -, ...) come after the operands. This can be derived by tracing around the tree of the other side, keeping our **right** hand on the tree and then reversing the resultant string.

- Both these methods of notation are useful for computer science: Compilers often convert infix to prefix and then to assembler code.

Ambiguity- example

- Consider the language generated by the following CFG:

$S \rightarrow AB$

$A \rightarrow a$

$B \rightarrow b$

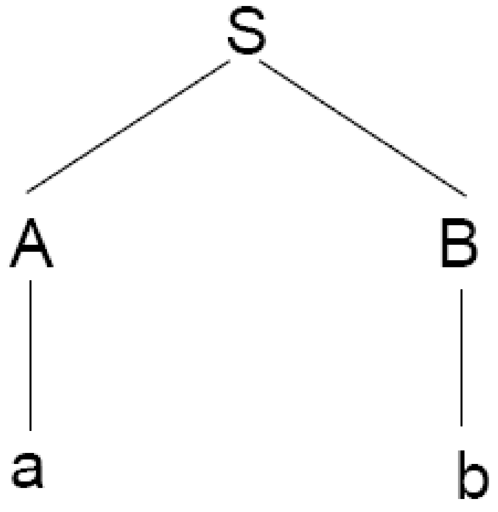
- There are two derivations of the word ab:

$S \Rightarrow AB \Rightarrow aB \Rightarrow ab$

or

$S \Rightarrow AB \Rightarrow Ab \Rightarrow ab$

- However, These two derivations correspond to the same syntax tree:



- The word ab is therefore not ambiguous. In general, when all the possible derivation trees are the same for a given word, then the word is unambiguous.

Ambiguity - Definition

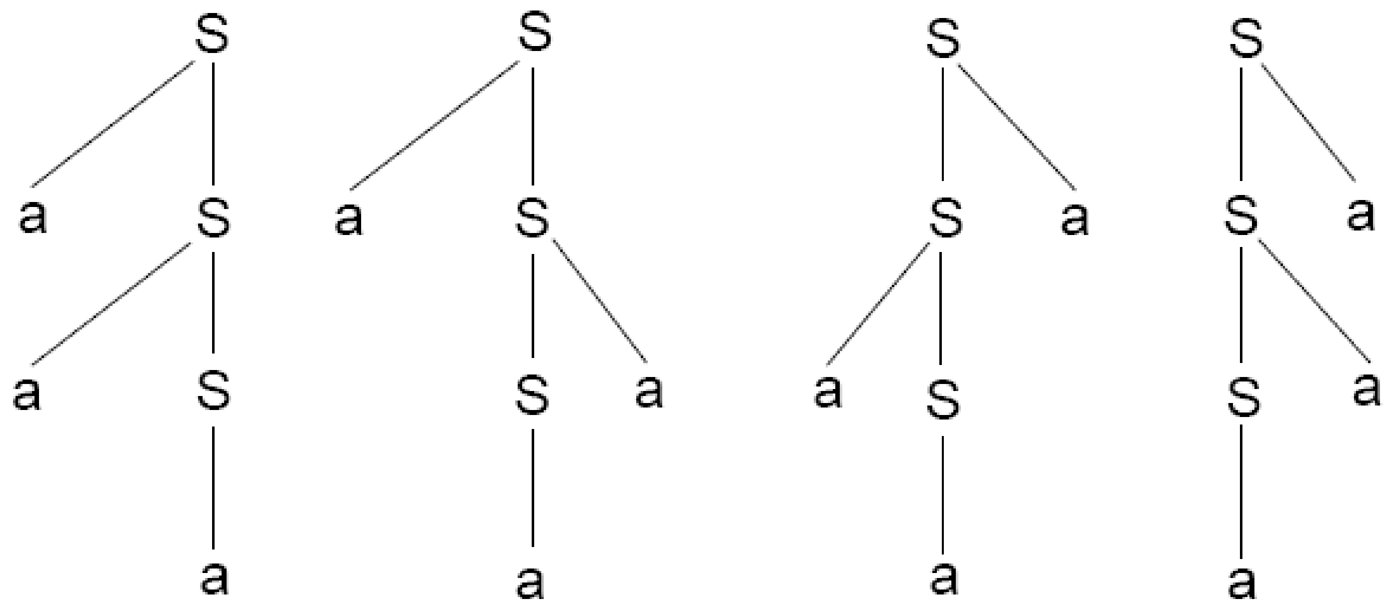
A CFG is called **ambiguous** if for at least one word in the language that it generates, there are two possible derivations of the word that correspond to different syntax trees. If a CFG is not ambiguous, it is called **unambiguous**.

Example

- The following CFG defines the language of all non-null strings of a's:

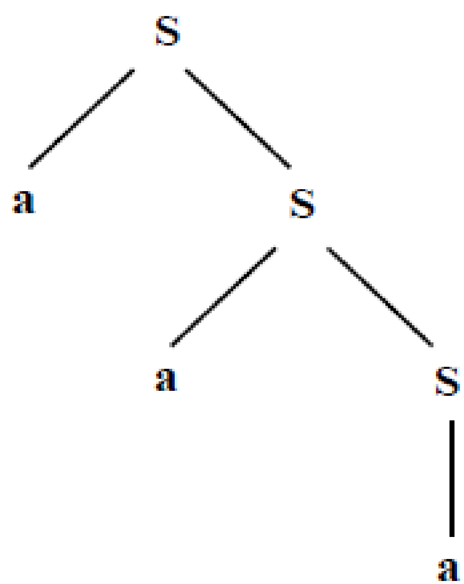
  S → aS| Sa |a

- The word a^3 can be generated by 4 different trees:



Example

- the CFG, $S \rightarrow aS|a$ is not ambiguous as neither the word aaa nor any other word can be derived from more than one production trees. The derivation tree for aaa is as follows:



The Total Language Tree

- It is possible to depict the generation of all the words in the language of a CFG simultaneously in one big (possibly infinite) tree.

Definition:

- For a given CFG, we define a tree with the start symbol S as its root and whose nodes are working strings of terminals and non-terminals. The descendants of each node are all the possible results of applying every applicable production to the working string, one at a time. A string of all terminals is a terminal node in the tree. The resultant tree is called the **total language tree** of the CFG.

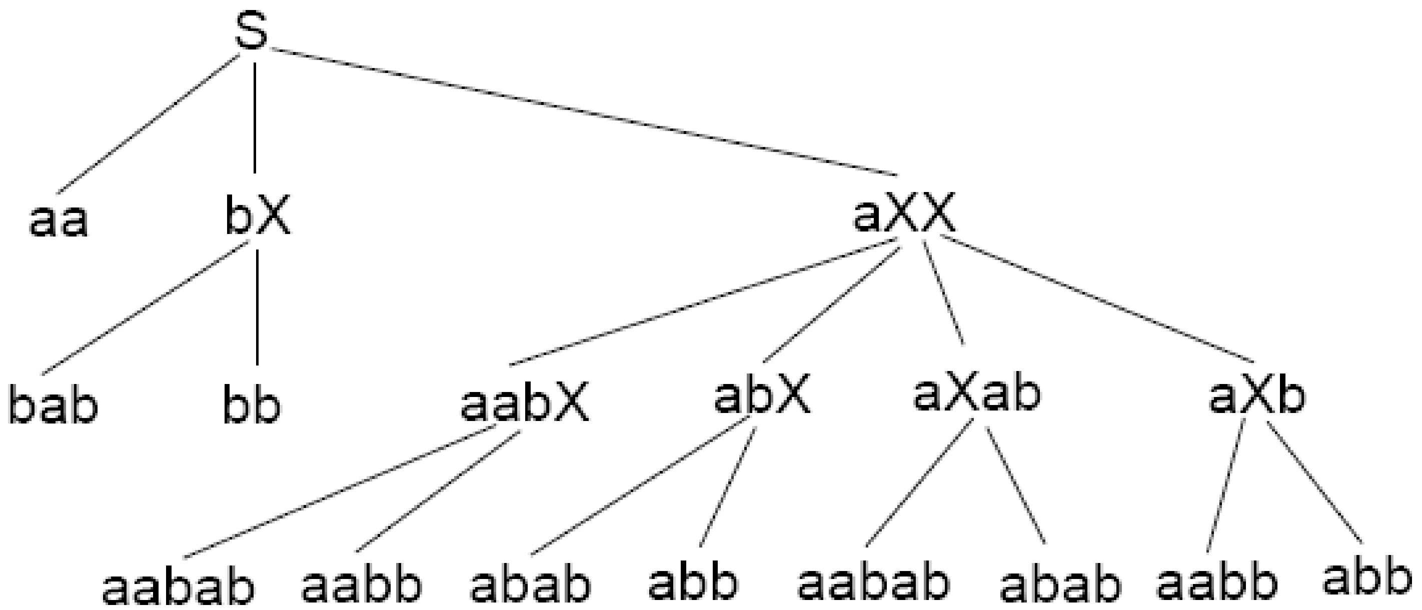
Example

- Consider the CFG:

  $S \rightarrow aa \mid bX \mid aXX$

  $X \rightarrow ab \mid b$

- The total language tree is



- The above total language has only 7 different words.
- Four of its words (abb, aabb, abab, aabab) have two different derivations because they appear as terminal nodes in two different places.
- However, these words are NOT generated by two different derivation trees. Hence, the CFG is unambiguous. For example,