NAME: SHOSAN HADIJAT ABIMBOLA

MATRIC NO:17/SCI01/076

1. Grammar: A grammar in compiler construction usually consists of at least two parts and sometimes three, less often one. I will start with the most common case first. The grammar has two parts, a lexical (lexer) specification, and a syntactic (parser) specification. A grammar is a set of rules that define what strings constitute a language.

It describes via rules what sentences the (often infinite) language contains (and can be used to generate a parser).It can also be used to generate (syntactically) valid sentences of the language.Most importantly, it relates the sentences of the language to their structure and categorizes parts of the sentence, which is the feature used by parsers.

Every grammar induces a set of tree-structures where for every sentence of the language at least one tree exists. If there is more than one tree for any sentence, that sentence is called syntactically ambiguous and also the grammar is called ambiguous.

2. Derivation: A derivation is basically a sequence of production rules, in order to get the input string. During parsing, we take two decisions for some sentential form of input: Deciding the non-terminal which is to be replaced. Deciding the production rule, by which, the non-terminal will be replaced.

A derivation is basically a sequence of production rules, in order to get the input string. During parsing, we take two decisions for some sentential form of input:

- Deciding the non-terminal which is to be replaced.
- Deciding the production rule, by which, the non-terminal will be replaced.

To decide which non-terminal to be replaced with production rule, we can have two options.

i. Left-most Derivation

If the sentential form of an input is scanned and replaced from left to right, it is called left-most derivation. The sentential form derived by the left-most derivation is called the left-sentential form.

ii. Right-most Derivation

If we scan and replace the input with production rules, from right to left, it is known as right-most derivation. The sentential form derived from the right-most derivation is called the right-sentential form.

3. Production: The productions of a grammar specify the manner in which the terminals and non-terminals can be combined to form strings. Each production consists of a non-terminal called the left

side of the production, an arrow, and a sequence of tokens and/or on- terminals, called the right side of the production.

Example:

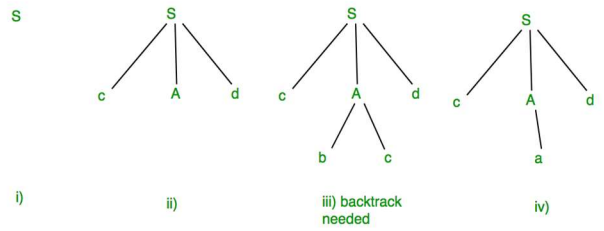
Suppose Production rules for the Grammar of a language are:

*S -> cAd*

*A -> bc|a*

*And the input string is "cad".*

Now the parser attempts to construct syntax tree from this grammar for the given input string. It uses the given production rules and applies those as needed to generate the string. To generate string "cad" it uses the rules as shown in the given diagram:




In the step iii above, the production rule A->bc was not a suitable one to apply (because the string produced is "cbcd" not "cad"), here the parser needs to backtrack, and apply the next production rule available with A which is shown in the step iv, and the string "cad" is produced.

Thus, the given input can be produced by the given grammar, therefore the input is correct in syntax. But backtrack was needed to get the correct syntax tree, which is really a complex process to implement.

4. Sentence: Sentence. A sentence is a sentential form consisting only of terminals such as a + a * a. A sentence can be derived using the following algorithm: Algorithm Derive String String := Start Symbol REPEAT Choose any nonterminal in String. Find a production with this nonterminal on the left-hand side.

A sentence is a sentential form consisting only of terminals such as a + a * a. A sentence can be derived using the following algorithm:

*Algorithm*

*Derive String*

*String := Start Symbol*

*REPEAT*

*Choose any nonterminal in String.*

*Find a production with this nonterminal on the left-hand side.*

*Replace the nonterminal with one of the options on the right-hand*

*side of the production.*

*UNTIL String contains only terminals.*

5. Null symbol: ε it is sometimes useful to specify that a symbol can be replaced by nothing at all. To indicate this, we use the null symbol ε, e.g., A –> B | ε.