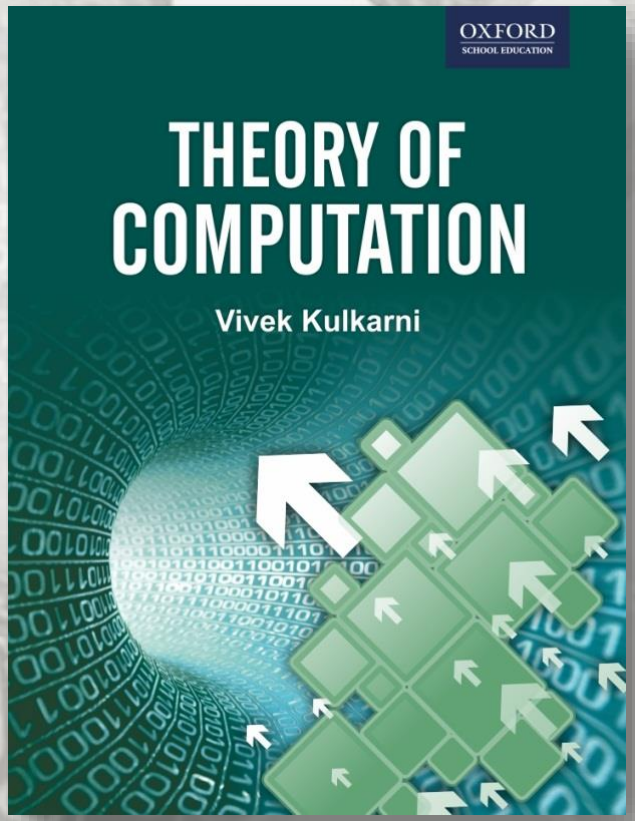
OXFORD
HIGHER EDUCATION

THEORY OF COMPUTATION

Vivek Kulkarni

Slides for Faculty Assistance



Chapter 5

Grammars

Author: Vivek Kulkarni

vivek_Kulkarni@yahoo.com

Outline

- Following topics are covered in the slides:
  - Concept of context-free grammars (CFGs)
  - Formalism of CFGs and context-free languages (CFLs)
  - Leftmost/rightmost derivations and derivation tree
  - Concept of ambiguous grammar and removal of ambiguity
  - Simplification of CFG
  - Chomsky normal form (CNF) and Greibach normal form (GNF)
  - Chomsky hierarchy
  - Equivalence of regular grammars and finite automata
  - pumping lemma for CFLs
  - Concept of derivation graph
  - Applications of CFG
  - Backus-Naur form (BNF)
  - Kuroda normal form, Dyck language,, and ogden's lemma

Grammar Concepts

- Grammar, for a particular language, is defined as a finite set of formal rules for generating syntactically correct sentences.
- Grammar consists of two types of symbols, namely:
  - **Terminals**: are part of a generated sentence
  - **Non-terminals** (also called variables or auxiliary symbols): take part in the formation (or generation) of the statement, but are not part of the generated statement
- For example, if we want to generate an English statement 'dog runs', we may use the following rules:

  < Sentence > → < noun > < verb >

  < Noun > → dog

  < Verb > → runs

  Here, 'dog' and 'runs' are the terminal symbols, while 'Sentence', 'Noun', and 'Verb' are non-terminals

Formal Definition of a Grammar

- A phrase structure grammar is denoted by a quadruple of the form:

  $G = \{V, T, P, S\}$,

  where,

  V: Finite set of non-terminals (or variables); sometimes this is also denoted by N instead of V

  T: Finite set of terminals

  S: S is a non-terminal, which is a member of N; it usually denotes the starting symbol

  P: Finite set of productions (or syntactical rules)

- Productions have the generic form as, '$\alpha \rightarrow \beta$', where $\alpha, \beta \subset (V \cup T)^*$, and $\alpha \neq \in$. As we know, $V \cap T = \phi$. Note that α and β may consist of any number of terminals as well as non-terminals and they are usually termed as *sentential forms.*
- If the production rules have the form: '$A \rightarrow \alpha$', where A is any non-terminal and α is any sentential form, then such grammar is called **context-free grammar (CFG)**.

Leftmost Derivation

- If at each step in a derivation, a production is applied to the leftmost variable (or non-terminal), then the derivation is called **leftmost derivation**.
- For example, if the grammar G consists of: {(E), (+, *, a), P, E},

  where P consists of the following productions:

  (1) $E \rightarrow E + E$

  (2) $E \rightarrow E * E$

  (3) $E \rightarrow a$
- Using this grammar, let us generate the string '$a + a$'.

  $E \Rightarrow \underline{E} + E$, using production (1)

  $\Rightarrow a + \underline{E}$, using production (3) applied to the leftmost non-terminal E

  $\Rightarrow a + a$, using production (3) applied to E, which is the only non-terminal
- Observe that in each step of the derivation, the leftmost non-terminal symbol is replaced by the right-hand side of the production applicable.

Rightmost Derivation

- If at each step in the derivation of a string, a production is always applied to the rightmost non-terminal, then the derivation is called **rightmost derivation**. It is also termed as *canonical derivation*.
- For example, if the grammar G consists of: {(E), (+, *, a), P, E},

  where P consists of the following productions:

  (1) $E \rightarrow E + E$

  (2) $E \rightarrow E * E$

  (3) $E \rightarrow a$
- Using this grammar, let us generate the string '$a + a * a$'.

$E \Rightarrow E + \underline{E}$, using production (1)

$\Rightarrow E + E * \underline{E}$, using production (2), applied to the rightmost E

$\Rightarrow E + \underline{E} * a$, using production (3), applied to the rightmost E

$\Rightarrow \underline{E} + a * a$, using production (3), applied to the rightmost E

$\Rightarrow a + a * a$, using production (3), applied to the only E left

Derivation Tree

- Derivation tree is a graphical representation, or description, of how the sentence (or string) has been derived, or generated, using a grammar. For every string derivable from the start symbol, there is an associated derivation tree. Derivation tree is also known as *rule tree*, *parse tree*, or *syntax tree*.
- Consider the grammar,

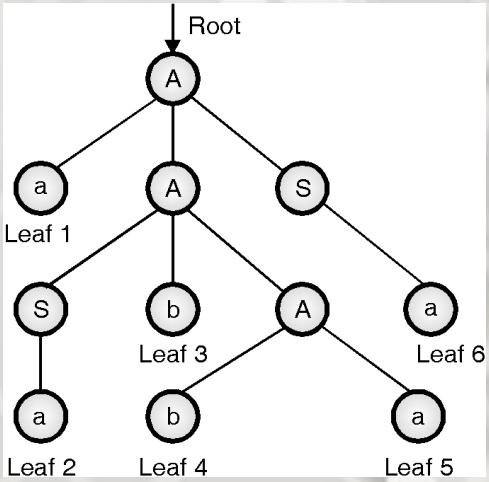
  $G = \{(S, A), (a, b), P, S\}$,

  where P consists of:

  $S \rightarrow a\ A\ S \mid a$

  $A \rightarrow S\ b\ A \mid S\ S \mid b\ a$

  The figure depicts the derivation tree for the string '*aabbaa*'.



Context-Free Languages (CFL)

- A **context-free language** (**CFL**) is the language generated by a context-free grammar G. This can be described as:

  $L(G) = \{w \mid w \subset T^*$, and is derivable from the start symbol $S\}$

- Examples:
  - *CFG with production rules,* $S \rightarrow a\ S\ b \mid a\ b$ *generates,*

    $L(G) = \{ab, aabb, aaabbb, aaaabbbb, \ldots\} = \{a^n b^n \mid n \geq 1\}$

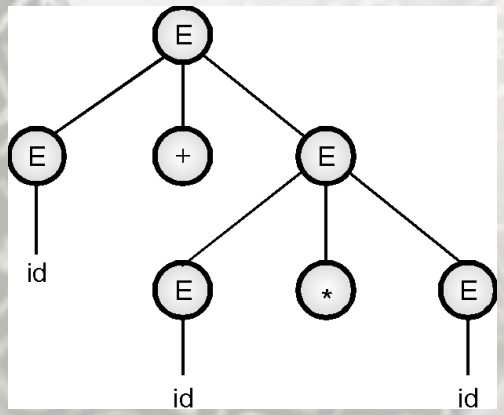
  - CFG with production rules, $S \rightarrow aS$, $S \rightarrow \in$ *generates all* strings over $\Sigma = \{a\}$, containing any (zero or more) number of a's, i.e., a regular language denoted by regular expression a^*

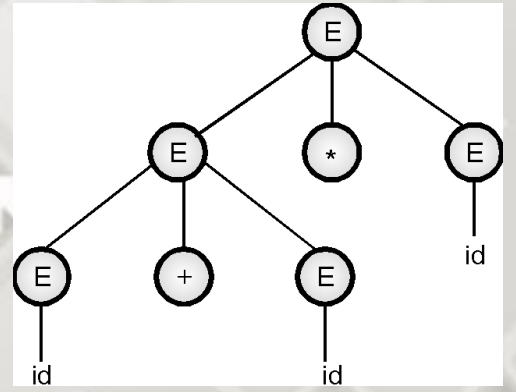
Ambiguous Context-Free Grammar

- A CFG for a language is said to be *ambiguous*, if there exists at least one string, which can be generated (or derived) in more than one way.
- For example, consider the following grammar:

  $E \rightarrow E + E \mid E * E \mid id$

  The string ' id + id * id' can be generated in two different ways as shown in the derivation trees below:





Removal of Ambiguity

- There is no algorithm for removing ambiguity in any given CFG. Every ambiguous grammar has a different cause for the ambiguity, and hence, the remedy for each would be different. Removing the ambiguity for a specific grammar is feasible; but creating a generic solution to remove the ambiguity of any ambiguous grammar is an *unsolvable problem*.

- For example, ambiguous grammar,

  $E \rightarrow E + E \mid E * E \mid id$

  can be re-written removing the ambiguity as,

  $E \rightarrow E + T \mid T$

  $T \rightarrow T * F \mid F$

  $F \rightarrow id$

Simplification of CFG

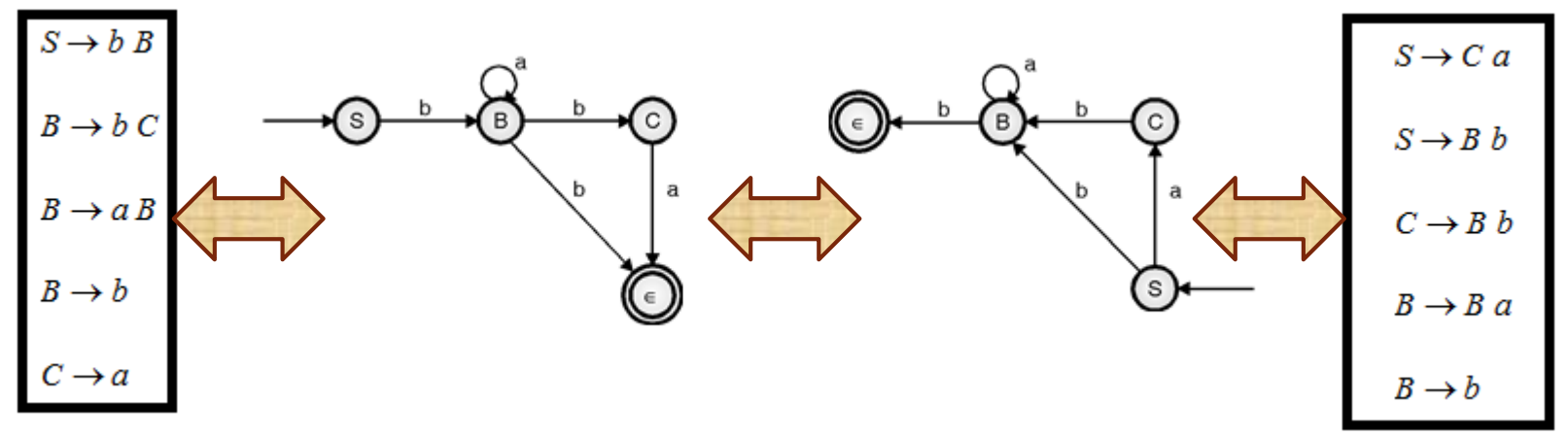
- A context-free grammar G can be simplified using following simple techniques:
  - Removal of Useless Symbols
  - Removal of Unit Productions ('$A \rightarrow B$')
  - Elimination of $\in$-Productions ('$A \rightarrow \in$')
  - Normalization:
    - **Chomsky normal form** (**CNF**): Productions can be expressed in only two ways: '$A \rightarrow BC$', and '$A \rightarrow a$'
    - **Greibach normal form** (**GNF**): every production is of the form, $A \rightarrow a\alpha$, where A is a non-terminal, a is a terminal, and α is a (possibly empty) string containing only non-terminals.

Chomsky Hierarchy

- Four different classes of phrase structure grammars, exists:
  - **Type-0 (unrestricted grammar)**
    - Productions of the form '$\alpha \rightarrow \beta$'; $\alpha \neq \in$, where $\alpha, \beta \subset (V \cup T)^*$
  - **Type-1 (context-sensitive grammar)**
    - Productions of the form $\alpha 1$ A $\alpha 2 \rightarrow \alpha 1$ β $\alpha 2$; ($\beta \neq \in$), where, the replacement of a non-terminal A by β is allowed only if $\alpha 1$ precedes A and $\alpha 2$ succeeds A; $\alpha 1$ and $\alpha 2$ may or may not be empty.
  - **Type-2 (context-free grammar)**
    - Productions of the form '$A \rightarrow \alpha$'; where $\alpha \subset (V \cup T)^*$
  - **Type-3 (regular grammar)**
    - **Left-linear grammar** (G_L): Productions of the form, $A \rightarrow B\ w$ or $A \rightarrow w$, where, A and B are non-terminals, and w is a string of terminals
    - **Right-linear grammar** (G_R): Productions of the form, $A \rightarrow w$ B or $A \rightarrow w$, where, A and B are non-terminals, and w is a string of terminals

Equivalence of Right-linear and Left-linear Grammars

- Inter-conversion steps:
  - Represent the given regular grammar by a transition diagram with vertices labelled by the symbols from $\{V \cup (\in)\}$ and transitions labelled by the symbols from $\{T \cup (\in)\}$.
  - Interchange the positions of the initial and final states.
  - Reverse the directions of all the transitions, keeping the positions of all intermediate states unchanged.
  - Rewrite the grammar from this new transition graph into another format.



Pumping Lemma for Context-Free Languages

- Let L be a CFL. Then, there exists a constant n such that if z is any string in L such that its length is at least n, i.e., $| z | \geq n$, then we can write $z = uvwxy$, where:
  - $| vwx | \leq n$; that is, the middle portion of the string is not too long.
  - $| vx | \geq 1$; that is, $vx \neq \in$. Since, v and x are the substrings that get pumped, it is required that at least one of them should be non-empty..
  - For all values of i; $i \geq 0$, uv^iwx^iy is in L; that is, the two substrings v and x can be pumped as many times as required and the resultant string obtained will be a member of L.

- Just as in the case of pumping lemma for regular sets, the pumping lemma for CFLs suggest finding some pattern near the middle of the string that can be pumped (or repeated) in order to describe the CFL. If such a property can be found for a given language L, then L is considered as a CFL; otherwise L is not a CFL.

Pumping Lemma - Example

- Show that $L = \{a^p \mid p \text{ is prime}\}$ is not a context-free language.
- **Solution**:
  - Step 1: Let us assume that the language L is a CFL.
  - Step 2: Let us choose a sufficiently large string z such that $z = a^l$ for some large l, which is prime. Since we assumed that L is a CFL and an infinite language; pumping lemma can be applied now. This means that we should be able to write a string: $z = uvwxy$.
  - Step 3: As per pumping lemma, every string 'uv^iwx^iy', for all $i \geq 0$ is in L.
- That is, if we consider $i = 0$, then $uv^0wx^0y = uwy$ is in L. Let us assume $| uwy |$ is a prime number, say k.
- As per pumping lemma, let us have $| vx | = r \geq 1$; and let $i = k$. Then, as per pumping lemma, 'uv^kwx^ky' should be in L. However, $| uv^kwx^ky | = | uwy | + k * | vx | = k + k * r = k (1 + r)$, which is not a prime number as it is divisible by k. Thus, 'uv^kwx^ky' is not in L. This contradicts our assumption that L is a CFL. Hence, L is not a CFL.

Ogden's Lemma

- This is a stronger version of pumping lemma for CFLs.
- It differs from the pumping lemma by allowing us to focus on any n distinguished (or marked) positions of a string z, and guaranteeing that the strings to be pumped have distinguished (or marked) positions between 1 and n.
- **Lemma statement:** If L is a CFL, then there exists a constant n, such that:

  If z is any string in language L having length at least n, and in which we select at least n positions to be distinguished (or marked), then we can write $z = uvwxy$ such that:

  - vwx has at most n distinguished (or marked) positions.
  - vx has at least one distinguished (or marked) position.
  - For all i, uv^iwx^iy is in L.

The Kuroda Normal Form

- The Kuroda normal form is for context-sensitive languages.
- A context-sensitive grammar is said to be in Kuroda normal form, if all the production rules are of the form:

  $AB \rightarrow CD$, or $A \rightarrow BC$, or $A \rightarrow B$, or $A \rightarrow a$,

  where A, B, C, and D are all non-terminal symbols and a is a terminal symbol.
- Every grammar in Kuroda normal form is monotonic or non-contracting. That is, none of the rules decreases the size of the string that is being generated.

Dyck Language

- Dyck language is the language consisting of well-formed parentheses. For such a language, the alphabet set can be written as = { (,) }; and the grammar for such a language can be written as: $S \rightarrow S\ (\ S\ )\ |\ \in$

- Given any word x over , if:

  $D\ (x)$ = (number of left parentheses in x) – (number of right parentheses in x), then x is a member of Dyck language if and only if:

  - $D\ (x) = 0$; and
  - $D\ (y) \geq 0$, for any prefix y of x.

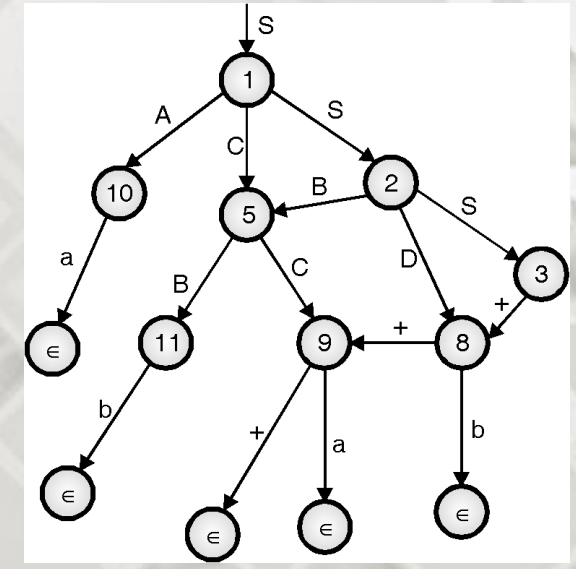
- Dyck language is also called *parenthesis language* and is a CFL.

Derivation Graph

- Derivation graph is required because,
  - Derivation tree does not give any information about the production rule that has been applied at each particular stage in the derivation.
  - Derivations of any string from the languages generated using type-0 and type-1 grammars are not supported by derivation trees.

1: $S \rightarrow ACS$	6: $DA \rightarrow AD$
2: $S \rightarrow BDS$	7: $DB \rightarrow BD$
3: $S \rightarrow +$	8: $D+ \rightarrow +b$
4: $CA \rightarrow AC$	9: $C+ \rightarrow +a$
5: $CB \rightarrow BC$	10: $A \rightarrow a$
	11: $B \rightarrow b$

$S \Rightarrow AC\underline{S}$,	(rule 1)
$\Rightarrow ACBD\underline{S}$,	(rule 2)
$\Rightarrow ACB\underline{D+}$,	(rule 3)
$\Rightarrow A\underline{CB}+b$,	(rule 8)
$\Rightarrow AB\underline{C}+b$,	(rule 5)
$\Rightarrow \underline{A}B+ab$,	(rule 9)
$\Rightarrow a\underline{B}+ab$,	(rule 10)
$\Rightarrow ab+ab$,	(rule 11)



Applications of CFGs and BNF form

- A parser basically uses the context-free grammar to group the tokens together and form sentences. If it is able to form a sentence from a sequence of tokens delivered to it by the lexical analyser, the sentence is considered to be syntactically correct.

- Backus-Naur form (or Backus-normal Form; abbreviated as BNF) is a notational technique used for context-free grammars.
  - In BNF notation, non-terminals can be represented by any word—similar to JAVA identifiers delimited by angular brackets on both the ends. For example, '<ifStatement>', '<program>', and '<postalAddress>' are valid non-terminals .
  - terminals are represented by words consisting of capital letters. For example, 'NEWLINE', 'ID', and 'NUMBER' are valid terminals.