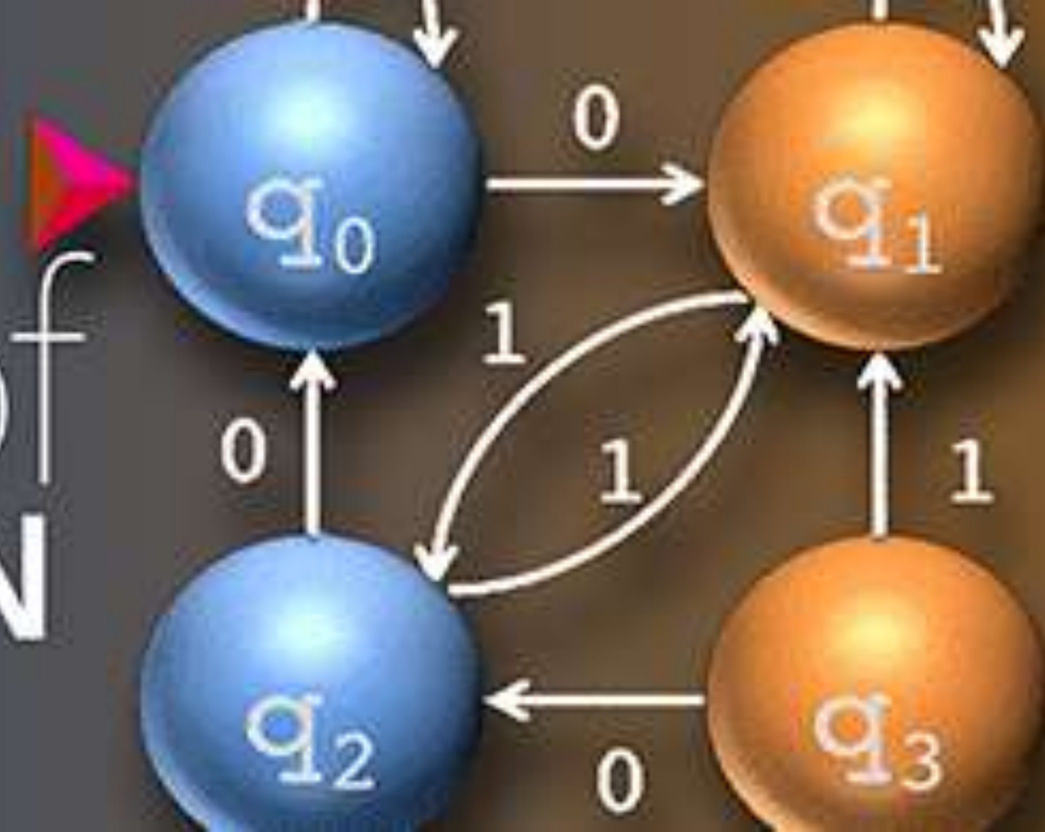
CSE 305

Theory of COMPUTATION



Lecture 23

Context-Free Grammars (1)

Md. Mijanur Rahman, Prof. Dr.

Dept. of Computer Science and Engineering, Jatiya Kabi Kazi Nazrul Islam University, Bangladesh.

www.mijanrahman.com

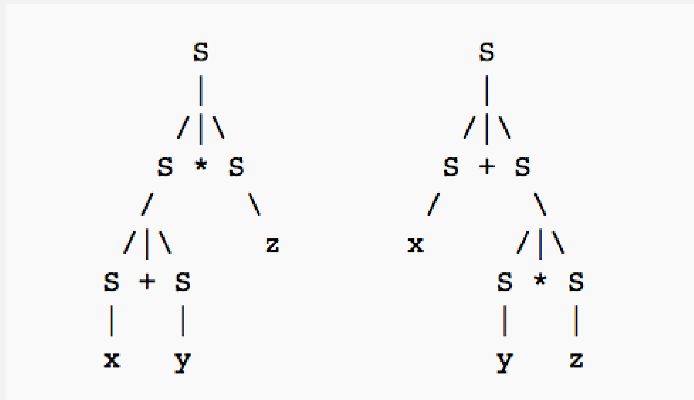
Contents

Context-Free Grammars

- ❑ Introduction to Context-Free Grammars (CFG)
- ❑ Formal Definition of Context-Free Grammars (CFG)
- ❑ Parse Trees
- ❑ Capabilities of CFG
- ❑ Relationship with other Computation Models
- ❑ Types of Context-Free Grammars
- ❑ Derivations Using Grammars
- ❑ Removal of Ambiguity
- ❑ Removal of Left Recursion
- ❑ Left Factoring
- ❑ Simplification of CFG
- ❑ Chomsky Normal Form (CNF)

Context-Free Grammars

- **Context-free grammars** (CFGs) are used to describe context-free languages. A context-free grammar is a set of recursive rules used to generate patterns of strings.
- **A context-free grammar can describe all regular languages and more, but they cannot describe *all* possible languages.**
- Context-free grammars are studied in fields of theoretical computer science, compiler design, and linguistics. CFG's are used to describe programming languages and parser programs in compilers can be generated automatically from context-free grammars.



- *Two parse trees that describe CFGs that generate the string "x + y * z".*

Context-Free Grammars

- Context-free grammars can generate context-free languages. **They do this by taking a set of variables which are defined recursively, in terms of one another, by a set of production rules.**

- Context-free grammars are named as such because any of the production rules in the grammar can be applied regardless of context—**it does not depend on any other symbols that may or may not be around a given symbol that is having a rule applied to it.**

Context-Free Grammars

- **Context-free grammars have the following components:**
  1. A set of **terminal symbols** which are the characters that appear in the language/strings generated by the grammar. Terminal symbols never appear on the left-hand side of the production rule and are always on the right-hand side. For example: a, b, c,…, 0, 1,…..
  2. A set of **nonterminal symbols** (or **variables**) which are placeholders for patterns of terminal symbols that can be generated by the nonterminal symbols. These are the symbols that will always appear on the left-hand side of the production rules, though they can be included on the right-hand side. The strings that a CFG produces will contain only symbols from the set of nonterminal symbols. For example: A, B, …, X, Y.
  3. A set of **production rules** which are the rules for replacing nonterminal symbols. Production rules have the following form: variable → string of variables and terminals. For example: $X \rightarrow A$, $A \rightarrow a$
  4. A **start symbol** which is a special nonterminal symbol that appears in the initial string generated by the grammar. For example: S.

Context-Free Grammars

- **To create a string from a context-free grammar, follow these steps:**
  1. Begin the string with a start symbol.
  2. Apply one of the production rules to the start symbol on the left-hand side by replacing the start symbol with the right-hand side of the production.
  3. Repeat the process of selecting nonterminal symbols in the string, and replacing them with the right-hand side of some corresponding production, until all nonterminals have been replaced by terminal symbols. Note, it could be that not all production rules are used.

- For comparison, a **context-sensitive grammar** can have production rules where both the left-hand and right-hand sides may be surrounded by a **context** of terminal and nonterminal symbols.

Formal Definition: **Context-Free Grammars**

- A context-free grammar can be described by a four-element tuple (V, T, P, S), where
  - V is a finite set of variables (which are non-terminal);
  - T is a finite set (disjoint from V) of terminal symbols;
  - P is a set of production rules where each production rule maps a variable to a string $s \in (V \cup T)^*$;
  - S (which is in V) which is a start symbol.

Example: Context-Free Grammars

- Example:

  Come up with a grammar that will generate the context-free (and also regular) language that contains all strings with matched parentheses.

- There are many grammars that can do this task. This solution is one way to do it, but should give you a good idea of if your (possibly different) solution works too.

  Starting symbol: S
  Production rules:

  $S \to (\ )$

  $S \to SS$

  $S \to (S)$

  $S \to \epsilon$ where ϵ is an empty string

  and translate them into a single line: $S \to (\ ) \mid SS \mid (S) \mid \epsilon$.

Example: Context-Free Grammars

Example: Here is a context-free grammar that generates arithmetic expressions (subtraction, addition, division, and multiplication).

- Start symbol = <expression>
- Terminal symbols = $\{+, -, *, /, (,), \text{number}\}$, where “number” is any number
- Production rules:
  1. <expression> → number
  2. <expression> → (<expression>)
  3. <expression> → <expression> + <expression>
  4. <expression> → <expression> - <expression>
  5. <expression> → <expression> * <expression>
  6. <expression> → <expression> / <expression>

  - This allows us to construct whatever expressions using multiplication, addition, division, and subtraction we want. What these production rules tell us is that the result of any operation, for example, multiplication, is also an expression (denoted, <expression>).

Example: Context-Free Grammars

Example: Using the example above, show the steps of deriving the following expression: $(4+5)*(2-6)$. Note, there are many ways to do this, but the solution below should give you enough guidance to check if your derivation works.

- <expression> → 4 (using rule 1)
  <expression> → 5 (using rule 1)
  <expression> → 4 + 5 (using rule 3)
  <expression> → (4 + 5) (using rule 2)
  <expression> → 2 (using rule 1)
  <expression> → 6 (using rule 1)
  <expression> → 2 - 6 (using rule 4)
  <expression> → (2 - 6) (using rule 2)
  <expression> → (4 + 5) * (2 - 6) (using rule 5)

- Production rules:
  1. <expression> → number
  2. <expression> → (<expression>)
  3. <expression> → <expression> + <expression>
  4. <expression> → <expression> - <expression>
  5. <expression> → <expression> * <expression>
  6. <expression> → <expression> / <expression>

Example: Context-Free Grammars

Example: Consider the following language

L= {wcw^R | w € (a, b)*}

and

Production rules:

S → aSa

S → bSb

S → c

- Now check that **abbcbba** string can be derived from the given CFG.

S ⇒ aSa

S ⇒ abSba

S ⇒ abbSbba

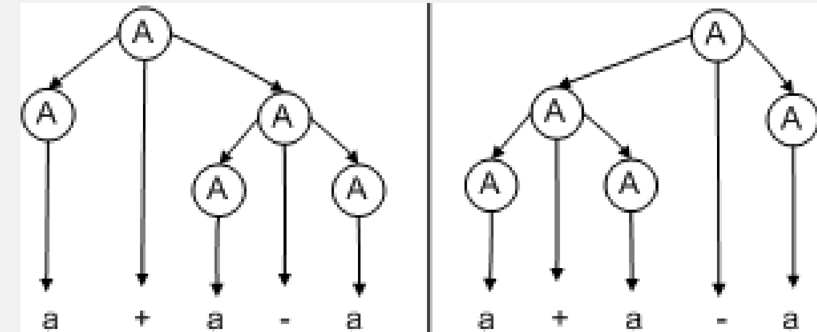
S ⇒ abbcbba

- By applying the production S → aSa, S → bSb recursively and finally applying the production S → c, we get the string **abbcbba**.

Parse Trees

- Context-free grammars can be modeled as **parse trees**.
- **In a parse tree**, the nodes of the tree represent the symbols and the edges represent the use of production rules. The leaves of the tree are the end result (terminal symbols) that make up the string the grammar is generating with that particular sequence of symbols and production rules.
- **Example:**
    - The parse trees below represent two ways to generate the string "a + a - a" with the grammar

      $A \rightarrow A{+}A \mid A{-}A \mid a.$



- Because this grammar can be implemented with multiple parse trees to get the same resulting string, this is said to be **ambiguous**.

Capabilities of CFG

- There are the various capabilities of CFG:
  - Context free grammar is useful to describe most of the programming languages.
  - If the grammar is properly designed then an efficient parser can be constructed automatically.
  - Using the features of associatively & precedence information, suitable grammars for expressions can be constructed.
  - Context free grammar is capable of describing nested structures like: balanced parentheses, matching begin-end, corresponding if-then-else's & so on.

Relationship with other Computation Models

- A context-free grammar can be generated by pushdown automata just as regular languages can be generated by finite state machines.
- Since all regular languages can be generated by CFGs, all regular languages can too be generated by pushdown automata.
- Any language that can be generated using regular expressions can be generated by a context-free grammar.
- The way to do this is to take the regular language, determine its finite state machine and write production rules that follow the transition functions.

Types of Context-Free Grammars

- **C**ontext **F**ree **G**rammars (CFG) can be classified on the basis of following two properties:

1. Based on number of strings it generates.

  - If CFG is generating finite number of strings, then CFG is **Non-Recursive** (or the grammar is said to be Non-recursive grammar)
  - If CFG can generate infinite number of strings then the grammar is said to be **Recursive** grammar

- During Compilation, the parser uses the grammar of the language to make a parse tree (or derivation tree) out of the source code.
- The grammar used must be unambiguous. An ambiguous grammar must not be used for parsing.

Types of Context-Free Grammars

2. Based on number of derivation trees.

- If there is only 1 derivation tree then the CFG is unambiguous.
- If there are more than 1 derivation tree, then the CFG is ambiguous.

Types of Context-Free Grammars

- **Examples of Recursive Grammars:**

1) $S \rightarrow SaS$

   $S \rightarrow b$

- The language (set of strings) generated by the above grammar is: {b, bab, babab,…}, which is infinite.

2) $S \rightarrow Aa$

   $A \rightarrow Ab|c$

- The language generated by the above grammar is :{ca, cba, cbba …}, which is infinite.
- **Note:** A recursive context-free grammar that contains no useless rules necessarily produces an infinite language.

Types of Context-Free Grammars

- **Examples of Non-Recursive Grammars:**

$$S \rightarrow Aa$$

$$A \rightarrow b|c$$

- The language generated by the above grammar is: {ba, ca}, which is finite.

Types of Context-Free Grammars

Types of Recursive Grammars:

- Based on the nature of the recursion in a recursive grammar, a recursive CFG can be again divided into the following:
  - Left Recursive Grammar (having left Recursion)
  - Right Recursive Grammar (having right Recursion)
  - General Recursive Grammar (having general Recursion)

- **Note:** A linear grammar is a context-free grammar that has at most one non-terminal in the right hand side of each of its productions.

Types of Context-Free Grammars

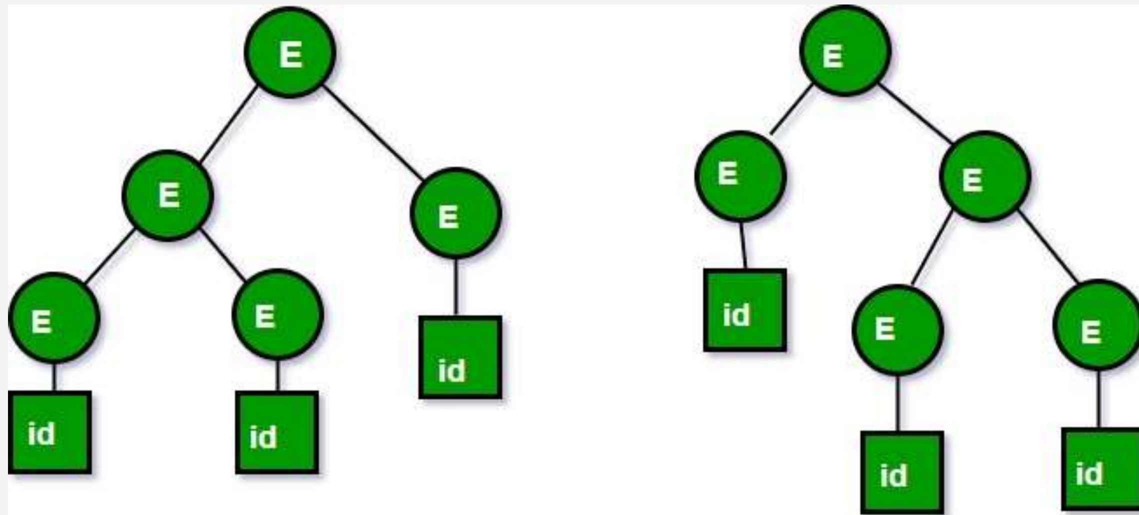
- **Ambiguous Grammars and Unambiguous Grammars:**
- **Ambiguous grammar:**
  A CFG is said to ambiguous if there exists more than one derivation tree for the given input string i.e., more than one **L**eft **M**ost **D**erivation **T**ree (LMDT) or **R**ight **M**ost **D**erivation **T**ree (RMDT).
- **Definition: G = (V,T,P,S) is a CFG is said to be ambiguous if and only if there exist a string in T* that has more than one parse tree.**
  where V is a finite set of variables.
  T is a finite set of terminals.
  P is a finite set of productions of the form, $A \rightarrow \alpha$, where A is a variable and $\alpha \in (V \cup T)^*$ S is a designated variable called the start symbol.

Types of Context-Free Grammars

- **Example of ambiguous grammar:**

1. Let us consider this grammar : **E → E+E | id**

- We can create 2 parse tree from this grammar to obtain a string **id+id+id.** The following are the 2 parse trees generated by left most derivation:



- Both the above parse trees are derived from same grammar rules but both parse trees are different. Hence the grammar is ambiguous.

Types of Context-Free Grammars

- **Example of ambiguous grammar:**

2. Let us now consider the following grammar:

```
Set of alphabets ∑ = {0,…,9, +, *, (, )}
E -> I
E -> E + E
E -> E * E
E -> (E)
I -> ε | 0 | 1 | … | 9
```

- From the above grammar String **3*2+5** can be derived in 2 ways. Hence the grammar is ambiguous.

```
I) First leftmost derivation
        E=>E*E
         =>I*E
         =>3*E+E
         =>3*I+E
         =>3*2+E
         =>3*2+I
         =>3*2+5
```

```
II) Second leftmost derivation
        E=>E+E
         =>E*E+E
         =>I*E+E
         =>3*E+E
         =>3*I+E
         =>3*2+I
         =>3*2+5
```

Types of Context-Free Grammars

- **Ambiguous grammar:**
- **Note :** Ambiguity of grammar is undecidable, i.e. there is no particular algorithm for removing the ambiguity of grammar, but we can remove ambiguity by:

  Disambiguate the grammar i.e., rewriting the grammar such that there is only one derivation or parse tree possible for a string of the language which the grammar represents.

? THE END