SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

CSE211 – Formal Languages and Automata Theory

U2L4_Ambiguity in Grammars & Languages

Dr. P. Saravanan
School of Computing
SASTRA Deemed University

Agenda

- Recap of previous class
- Ambiguous grammar
- Example for ambiguous grammar
- Multiplicity of derivation
- Inherent Ambiguity
- Removing Ambiguity from grammar
- Problems: Checking ambiguity of grammar

Recap of previous class

- Derivation using Grammar
- Types of derivation
- Examples
- Comparison of LM and RM derivation
- Definition of CFL
- Sentential Forms
- Parse Tree
- Examples for parse tree
- Yield of a parse tree

Parse Tree: Example

- Given the grammar:

  S → if (E) S | if (E) S else S

  S → other

  E → expr

- Draw the parse tree for

  if (expr) if (expr) other else other

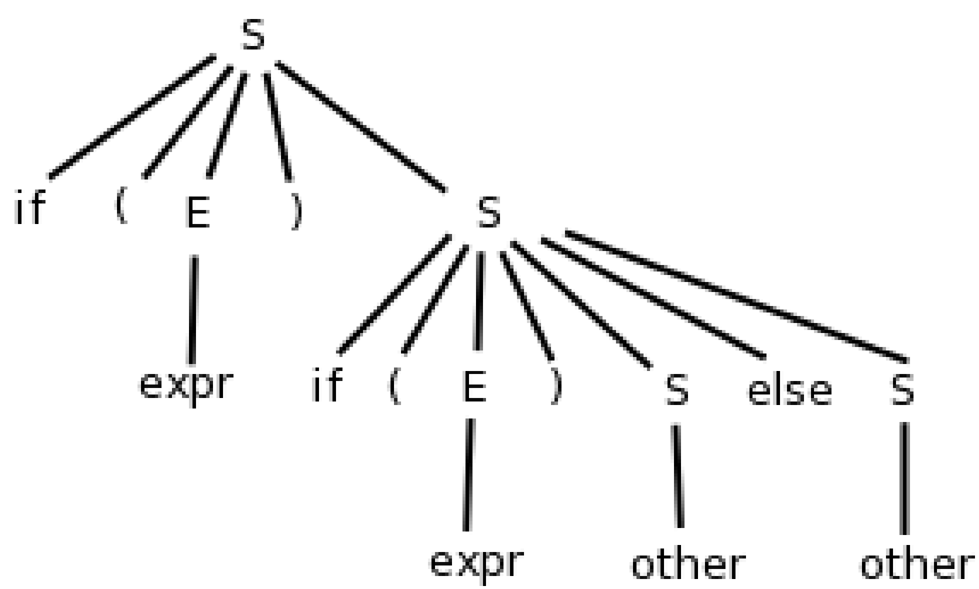
Parse Tree Example

- if (expr) if (expr) other else other

Production Rules

$S \rightarrow$ if (E) S / if (E) S else S
$S \rightarrow$ other
$E \rightarrow$ expr

Parse Tree 1



Derivation 1:

$S \Rightarrow$ if (E) S
$\Rightarrow$ if (E) if (E) S else S
$\Rightarrow$ if (E) if (E) S else other
$\Rightarrow$ if (E) if (E) other else other
$\Rightarrow$ if (E) if (expr) other else other
$\Rightarrow$ if (expr) if (expr) other else other

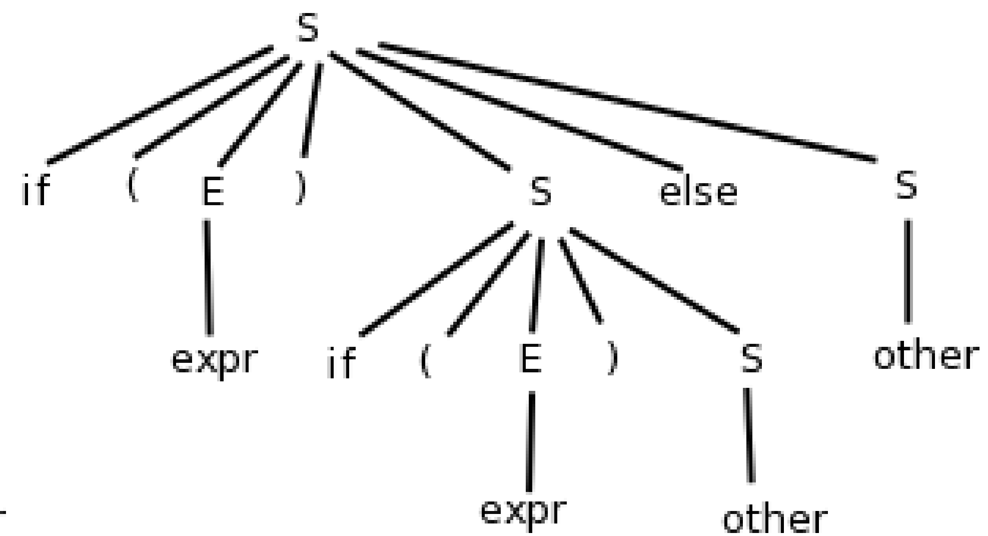
Parse Tree Example

- if (expr) if (expr) other else other

Production Rules

$S \rightarrow$ if (E) S / if (E) S else S
$S \rightarrow$ other
$E \rightarrow$ expr

Parse Tree 2



Derivation 2:

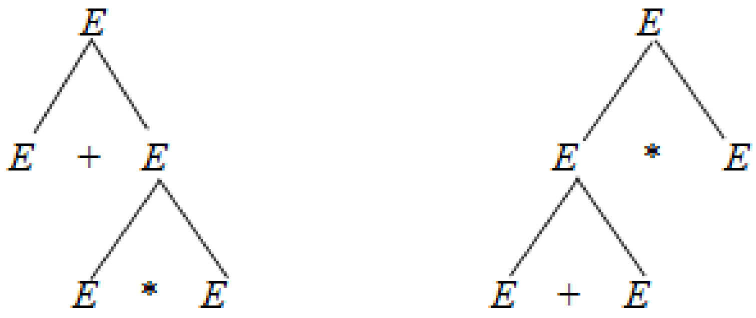
$S \Rightarrow$ if (E) S else S
$\Rightarrow$ if (E) S else other
$\Rightarrow$ if (E) if (E) S else other
$\Rightarrow$ if (E) if (E) other else other
$\Rightarrow^2$ if (expr) if (expr) other else other if (E) S

Ambiguous Grammars

- **Definition:**
  - A CFG $G = (V, T, P, S)$ is *ambiguous* if there exists at least one string w in T^* for which there are two different parse trees, each with root labeled S and yield w *identically*.
- If each string has at most one parse tree in G, then G is *unambiguous*.
- Caused by multiplicity of parse tree
- Ambiguity causes problems in program compiling
- It causes multiple syntactic interpretation of the string

Example

- Given the productions of the arithmetic expression grammar G_r of Example 5.3 as follows,
  - $E \rightarrow I \mid E + E \mid E * E \mid (E)$
  - $I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$
- the parse trees of the following two derivations
  - $E \Rightarrow E + E \Rightarrow E + E * E$ *and* $E \Rightarrow E * E \Rightarrow E + E * E$
- are shown as two trees obviously are *distinct*.



Multiplicity of derivation

- More than one form of derivation for the same string is called multiplicity
- Two derivations of the string $a + b$ are as follows:
  - $E \Rightarrow E + E \Rightarrow I + E \Rightarrow a + E \Rightarrow a + I \Rightarrow a + b$
  - $E \Rightarrow E + E \Rightarrow E + I \Rightarrow I + I \Rightarrow I + b \Rightarrow a + b$
- The parse trees for the above two derivations may be checked easily to be the same
- The ambiguity is caused by a multiplicity of parse trees rather than a multiplicity of derivations

Inherent Ambiguity

- But some CFL's are "*inherently ambiguous*," i.e., every grammar has more than one distinct parse tree for each of *some strings* in the language.
- A CFL *L* is said to be *inherently ambiguous* if all its grammars are ambiguous.
- Even one grammar of L is unambiguous then L is said to be unambiguous
- An example of inherently ambiguous languages

$$L = \{a^n b^n c^m d^m \mid n \geq 1, m \geq 1\} \cup \{a^n b^m c^m d^n \mid n \geq 1, m \geq 1\}$$

Removing Ambiguity from Grammars

- Ambiguity in certain grammars may be removed by redesigning the grammar.
- **Concept:**
  - There is *no* general algorithm that can tell whether a grammar is ambiguous or not
  - That is, testing of grammatical ambiguity is *undecidable*
  - Ambiguity in *inherently ambiguous* grammars is also *irremovable*
  - But elimination of ambiguity in some common programming language structures is possible

Removing Ambiguity: Example

- Remove the ambiguity of the arithmetic expression grammar of Example 5.3 whose productions are repeated below:
  - $E \rightarrow I \mid E + E \mid E * E \mid (E)$
  - $I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$

Removing Ambiguity: Example

- Two distinct causes of ambiguity in the grammar
- The precedence of operators in not respected
  - In conventional arithmetic's, the “×” has a highest precedence than the “+” or “–” operators:

$$a - a * a = a - (a * a)$$

- A sequence of “+” or “–” operators, or a sequence of “×” operators, can group either from the left or from the right.
  - Most operators are left associative in conventional arithmetic:

$$a - a + a - a - a = (((a - a) + a) - a) - 2$$

Removing Ambiguity: Example

- The solution of the problem is to introduce several different variables,
- each of the variable represents those expressions that share a level of "binding strength." Specifically
  1. Any string in the language is a mathematical expression E.
  2. Each expression consists in the sum or difference of terms T (or just in a single term).
  3. Each term consists, in turn, in the product of factors F (or just in a single factor).

Removing Ambiguity: Example

- This hierarchy can be illustrated as follows:

$$\underbrace{\underbrace{\underbrace{a}_{\text{factor}}}_{\text{term}} + \underbrace{\underbrace{a}_{\text{factor}} * \underbrace{a}_{\text{factor}}}_{\text{term}}}_{\text{expression}}$$

- The new unambiguous grammar for

- $$\begin{aligned} &E \rightarrow I \mid E + E \mid E * E \mid (E) \\ &I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \end{aligned}$$

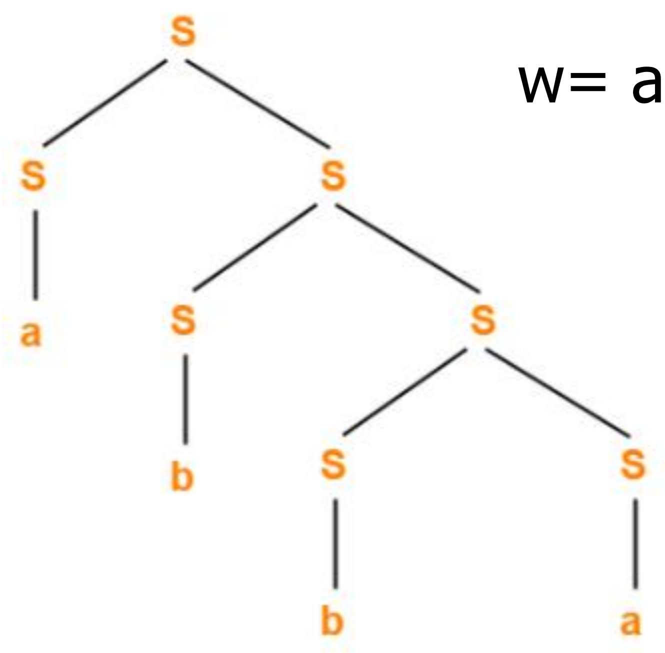
  is

$$\begin{aligned} &E \rightarrow T \mid E + T \\ &T \rightarrow F \mid T * F \\ &F \rightarrow I \mid (E) \\ &I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \end{aligned}$$

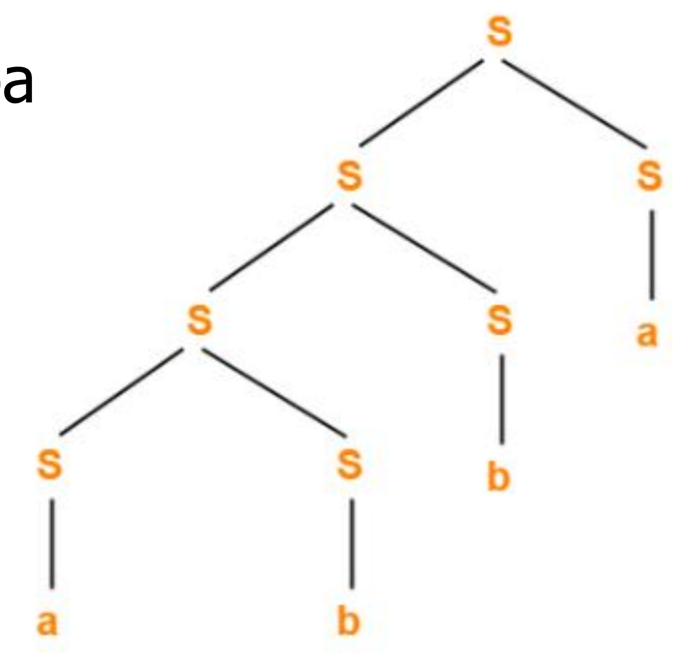
Problems: Checking Ambiguity

1. Check whether the given grammar is ambiguous or not- { S → SS, S → a, S → b }

w= abba



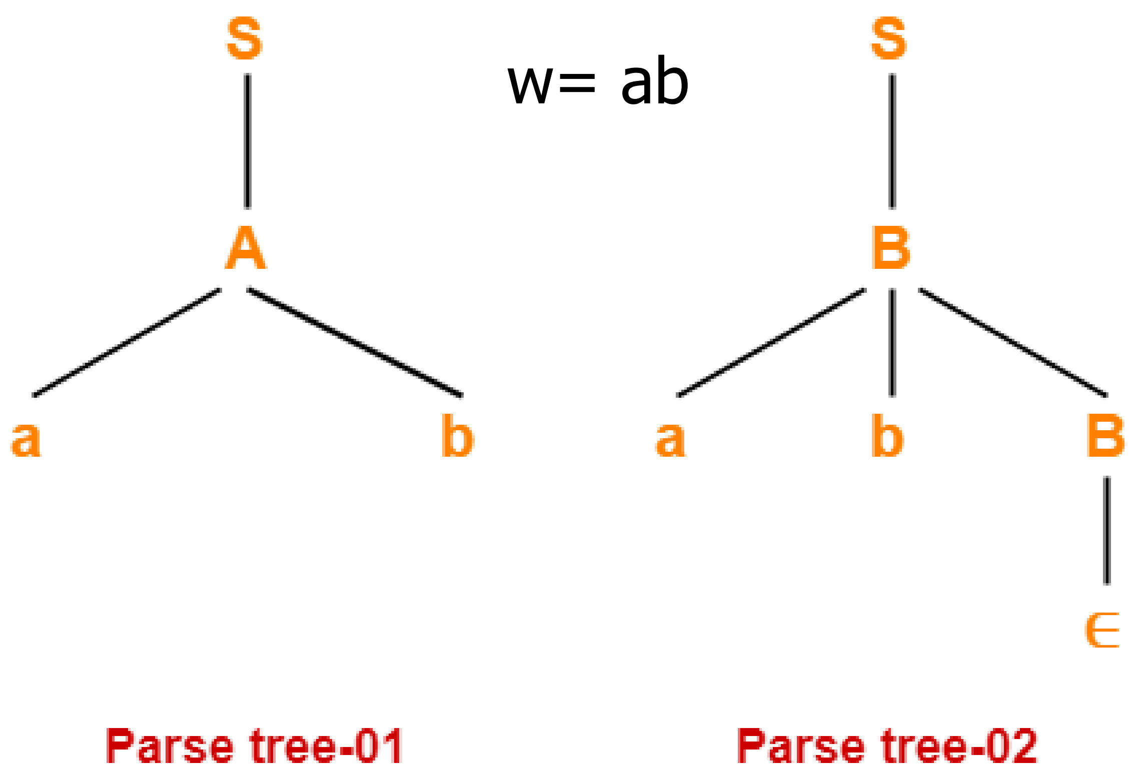
Parse tree-01



Parse tree-02

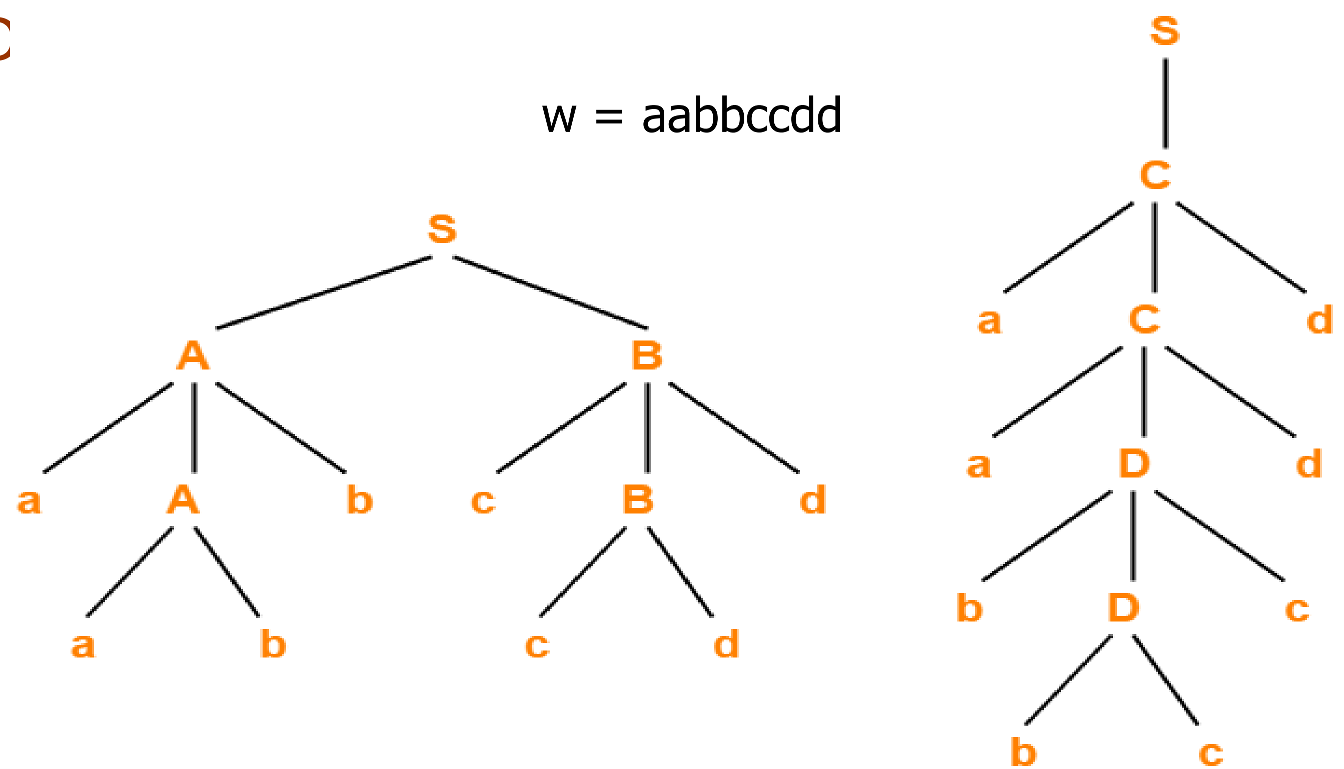
Problems: Checking Ambiguity

2. Check whether the given grammar is ambiguous or not



Problems: Checking Ambiguity

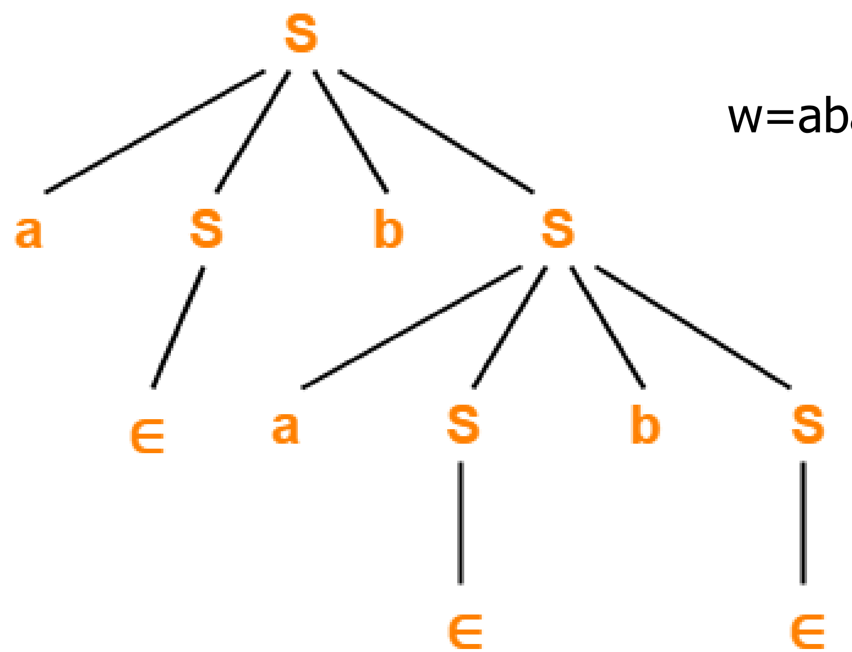
3. Check whether the given grammar is ambiguous or not
   - {S → AB / C, A → aAb / ab, B → cBd / cd, C → aCd / aDd, D Ɛ }



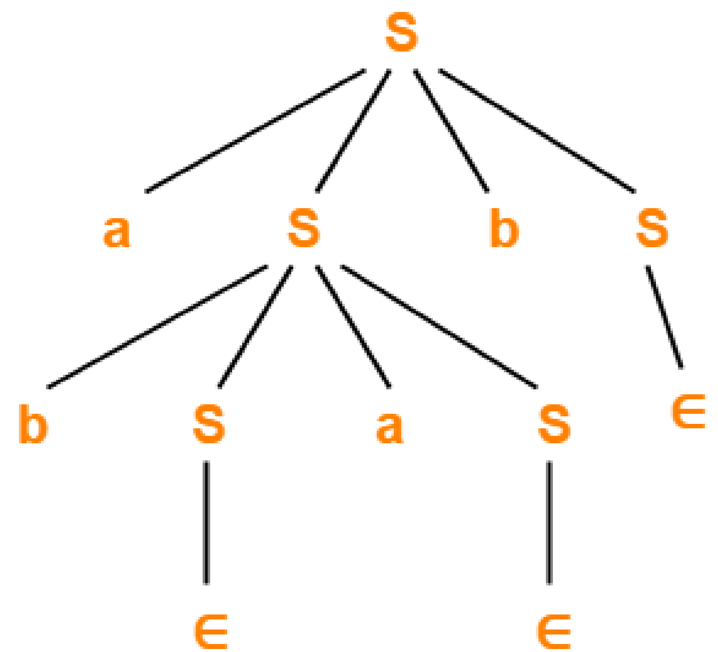
Problems: Checking Ambiguity

4. Check whether the given grammar is ambiguous or not

w=abab



Parse tree-01



Parse tree-02

Summary

- Ambiguous grammar
- Example for ambiguous grammar
- Multiplicity of derivation
- Inherent Ambiguity
- Removing Ambiguity from grammar
- Problems: Checking ambiguity of grammar

References

- John E. Hopcroft, Rajeev Motwani and Jeffrey D. Ullman, *Introduction to Automata Theory,* Languages, and Computation, Pearson, 3rd Edition, 2011.
- Peter Linz, An Introduction to Formal Languages and Automata, Jones and Bartle Learning International, United Kingdom, 6th Edition, 2016.

Next Class:

PDA

THANK YOU.