CS202 – System Software

Dr. Manish Khare

Lecture 9

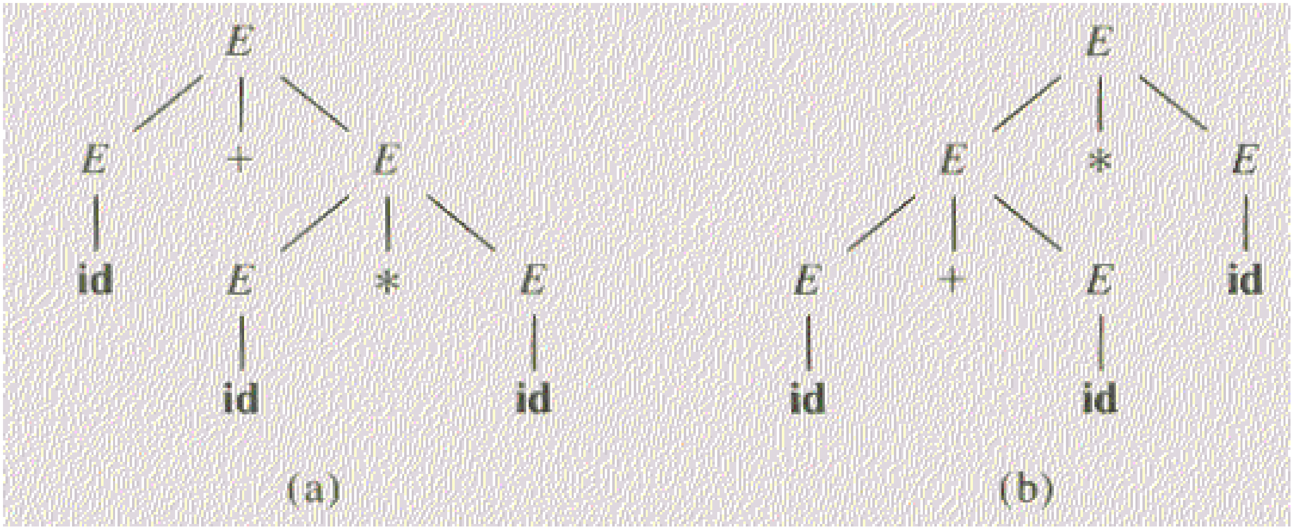
Parse tree

- Parse tree ignores variations in the order in which symbols in sentential forms are replaced.
- The variation in the order in which production are applied can also be eliminated by consider only left most (or right most) derivations.
- Every parse tree has associated with its unique left most and a unique right most derivation.
  1. *Leftmost derivation:* replace the leftmost non-terminal at each step
  2. *Rightmost derivation:* replace the rightmost non-terminal at each step

Parse tree

- For ex. Let us consider the grammar
  - E → E+E | E*E | (E) | id
- The sentence id + id * id has two distinct derivations

E → E+E	**E → E*E**
E → id+E	**E → E+E*E**
E → id+E*E	**E → id+E*E**
E → id+id*E	**E → id+id*E**
E → id+id*id	**E → id+id*id**



The parse tree in fig(a) reflects the commonly assumed precedence of + and *, while the parse tree in fig(b) treat operator * as having higher precedence than +

Ambiguity

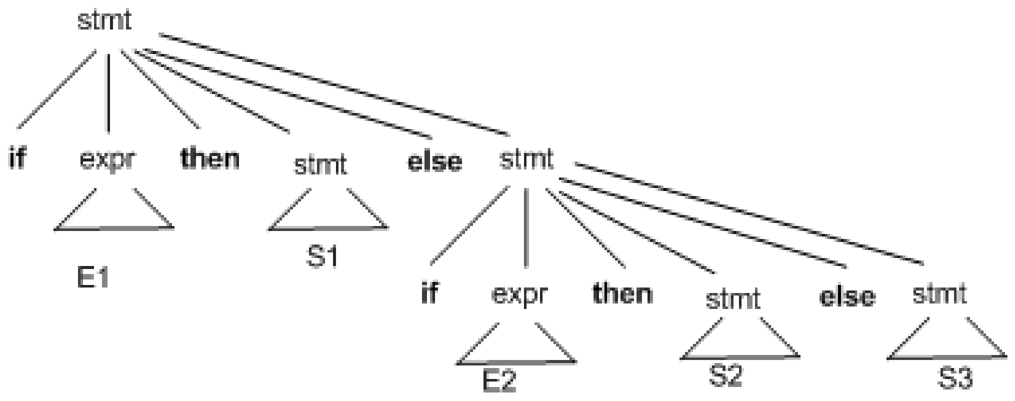
- A grammar that produces more than one parse tree for same sentence is said to be ambiguous.
- Another way, an ambiguous grammar is one that produces more than one left most or more than one right most derivation for the same sentence
- For certain types of parser, it is desirable that grammar to be made unambiguous, for if it is not, we can not uniquely determine which parse tree to select for a sequence.

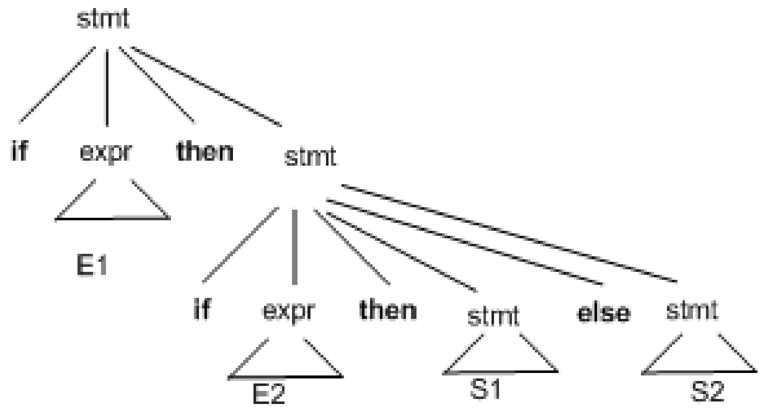
Elimination of Ambiguity

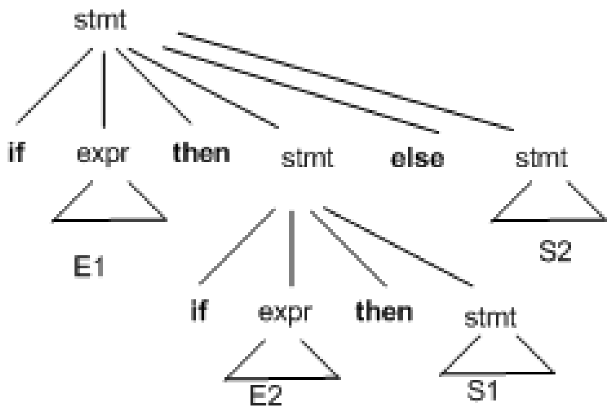
- An ambiguous grammar can be rewritten to eliminate the ambiguity.
- As an example, we shall eliminate the ambiguity from the following dangling-else grammar.

$$\begin{array}{lcl} stmt & \longrightarrow & \text{If } expr \textbf{ then } stmt \\ & | & \text{If } expr \textbf{ then } stmt \textbf{ else } stmt \\ & | & \textbf{other} \end{array}$$

- Here, ‘other’ stands for any other statement. According to this grammar the compound conditional statement.
  - if E1 then S1 else if E2 then S2 else S3
- This grammar is ambiguous, because this have three different parse tree


stmt
if
expr
then
stmt
else
stmt
E1
S1
if
expr
then
stmt
else
stmt
E2
S2
S3


stmt
if
expr
then
stmt
E1
if
expr
then
stmt
else
stmt
E2
S1
S2


stmt
if
expr
then
stmt
else
stmt
E1
S2
if
expr
then
stmt
E2
S1

Exercise

- Find grammar G with following production is ambiguous or not for string ‘abab’?
  - $S \rightarrow a \mid aAb \mid abSb$
  - $A \rightarrow aAAb \mid bS$

Exercise

- Check whether the given grammar is ambiguous or not for string ‘ibtibtibtaea’?
  - $S \rightarrow iCtS \mid iCtSeS \mid a$
  - $C \rightarrow b$

Types of Recursion

- Left Recursion
- Right Recursion
- General Recursion

Left Recursion

- A production of grammar is said to have **left recursion** if the leftmost variable of its RHS is same as variable of its LHS.

- A grammar containing a production having left recursion is called as Left Recursive Grammar.
- Example:
  - $S \rightarrow Sa \,/\, \in$

  (**Left Recursive Grammar**)

- Left recursion is considered to be a problematic situation for Top down parsers.

- Therefore, left recursion has to be eliminated from the grammar.

Right Recursion

- A production of grammar is said to have **right recursion** if the rightmost variable of its RHS is same as variable of its LHS.
- A grammar containing a production having right recursion is called as Right Recursive Grammar.
- Example
  - $S \rightarrow aS \,/\, \in$
- Right recursion does not create any problem for the Top down parsers.
- Therefore, there is no need of eliminating right recursion from the grammar.

General Recursion

- The recursion which is neither left recursion nor right recursion is called as general recursion.

- **Example-**
  - $S \to aSb \,/\, \in$

Elimination of Left Recursion

- Left recursion is eliminated by converting the grammar into a right recursive grammar.
- If we have the left-recursive pair of productions-
  - $\mathbf{A \longrightarrow A\alpha\ /\ \beta}$ (Left Recursive Grammar)
- where β does not begin with an A.

- Then, we can eliminate left recursion by replacing the pair of productions with-
  - $\mathbf{A \longrightarrow \beta A'}$
  - $\mathbf{A' \longrightarrow \alpha A'\ /\ \in}$ (Right Recursive Grammar)

- This right recursive grammar functions same as left recursive grammar.

Example

- Consider the following grammar for athematic expression
  - $E \rightarrow E+T \mid T$
  - $T \rightarrow T*F \mid F$
  - $F \rightarrow (E) \mid id$

- Eliminating the immediate left recursion (productions of the form $A \rightarrow A\alpha$) to the production for E & then for T, we obtain
  - $E \rightarrow TE'$
  - $E' \rightarrow +TE' \mid \varepsilon$
  - $T \rightarrow FT'$
  - $T' \rightarrow *FT' \mid \varepsilon$
  - $F \rightarrow (E) \mid \mathbf{id}$

- Consider the following grammar and eliminate left recursion-
  - $A \rightarrow ABd \,/\, Aa \,/\, a$
  - $B \rightarrow Be \,/\, b$

- The grammar after eliminating left recursion is-
  - $A \rightarrow aA'$
  - $A' \rightarrow BdA' \,/\, aA' \,/\, \in$
  - $B \rightarrow bB'$
  - $B' \rightarrow eB' \,/\, \in$

Elimination of Left recursion

- In general, left recursion can be eliminated by the following techniques, which works for any number of A-production.
- First, group the productions as $A \rightarrow A\alpha_1 \mid A\alpha_2 \mid \ldots \mid A\alpha_m \mid \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$ where no β_i begins with an A. then replace the A-productions by

  - $A \rightarrow \beta_1 A' \mid \beta_2 A' \mid \beta_3 A' \mid \ldots \mid \beta_n A'$
  - $A' \rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \alpha_3 A' \mid \ldots \mid \alpha_m A' \mid \varepsilon$

- The non-terminal A generates the same strings as before but is no longer left recursive. This procedure eliminates all left recursion from A and A' productions, but it does not eliminates left recursion involving derivations of two or more steps.

Example

- Consider the following grammar
  - $S \rightarrow Aa \mid b$
  - $A \rightarrow Ac \mid Sd \mid \varepsilon$
- We have two non-terminal S, A.
- Is the grammar is left recursive?
- The non-terminal S is left recursive because $S \rightarrow Aa \rightarrow Sda$, but it is not immediate left recursive.
- For for removing left recursion, we need to check second grammar. And if we removed left recursion from second grammar, then it will work.
- We substitute this S-production in $A \rightarrow Sd$ to obtain the following A-production
  - $A \rightarrow Ac \mid Aad \mid bd \mid \varepsilon$
- Now we remove left recursion among A-production

Example

- $S \rightarrow Aa \mid b$
- $A' \rightarrow bdA' \mid A'$
- $A' \rightarrow cA' \mid adA' \mid \varepsilon$

Exercise

- Check this grammar
  - $S \rightarrow A \mid B$
  - $A \rightarrow ABc \mid AAdd \mid a \mid aa$
  - $B \rightarrow Bcc \mid b$
- Remove Left Recursion

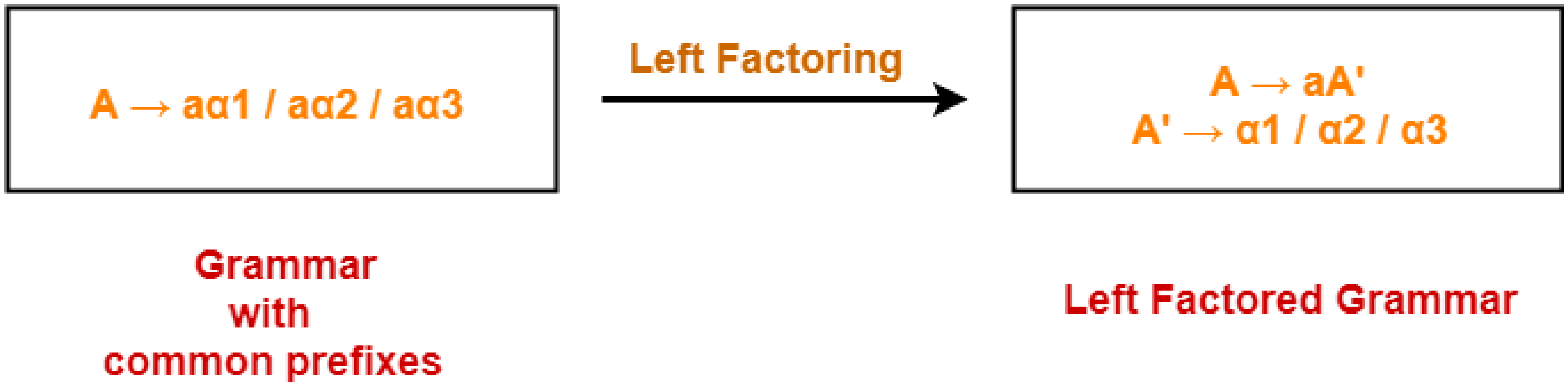
- Ans??
  - $S \rightarrow A \mid B$
  - $A \rightarrow aA' \mid aaA'$
  - $A' \rightarrow BcA' \mid AddA' \mid \varepsilon$
  - $B \rightarrow bB'$
  - $B \rightarrow ccB' \mid \varepsilon$

Exercise

- Check this grammar
  - $S \to Bb \mid a$
  - $B \to Bc \mid Sd \mid e$
- Remove left recursion
- Ans??
  - $S \to Bb \mid a$
  - $B \to adB' \mid eB'$
  - $B' \to cB' \mid bdB' \mid \varepsilon$

Left Factoring

- Left factoring is a grammar transformation that is useful for producing a grammar suitable for predictive, or top-down, parsing.
- In another way, left factoring is a process which isolates the common parts of two productions into a single production
- When the choice between two alternative A-productions in not clear, we may be able to rewrite the productions to defer the decision until enough of the input has been seen that we can make the right choice
- Any production of the form
    - $A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \ldots\ldots\ldots \mid \alpha\beta_n \mid$
  - Can be replaced by
    - $A \rightarrow \alpha A^{/}$
    - $A^{/} \rightarrow \beta_1 \mid \beta_2 \mid \ldots\ldots\ldots \mid \beta_n \mid$
- Left factoring is useful for producing a grammar suitable for predictive parser.


A → aα1 / aα2 / aα3
Left Factoring
A → aA'
A' → α1 / α2 / α3
Grammar with common prefixes
Left Factored Grammar

Example

- Consider the following grammar
  - $A \rightarrow aAB \mid aA \mid a$
  - $B \rightarrow bB \mid b$
- remove the left factoring from it
- $A \rightarrow aAB \mid aA \mid a$ will be replaced by
  - $A \rightarrow aA^{/}$
  - $A^{/} \rightarrow AB \mid A \mid \varepsilon$
- $B \rightarrow bB \mid b$ will be replaced by
  - $B \rightarrow bB^{/}$
  - $B^{/} \rightarrow B \mid \varepsilon$

- Do left factoring in the following grammar-
  - S → iEtS / iEtSeS / a
  - E → b

- The left factored grammar is-
  - S → iEtSS' / a
  - S' → eS / ∈
  - E → b

Relationship Between Left Recursion, Left Factoring & Ambiguity

- All the three concepts are independent and has nothing to do with each other.
- The presence or absence of left recursion does not impact left factoring and ambiguity anyhow.
- The presence or absence of left factoring does not impact left recursion and ambiguity anyhow.
- The presence or absence of ambiguity does not impact left recursion and left factoring anyhow.