Left recursion and Left factoring

Session Outcomes

- At the end of this session, participants will be able to
  - Understand different types of grammar
  - Understand the concepts of left recursion
  - Understand the concepts of left factoring

Outline

- Elimination of left recursion
- Left factoring

Types of grammar

- Ambiguous grammar
- Un ambiguous grammar
- Recursive grammar
- Non deterministic grammar
- Deterministic grammar

Left Recursion

- A grammar is ***left recursive*** if it has a non-terminal A such that there is a derivation.

  $A \overset{+}{\Rightarrow} A\alpha$ for some string α
- Top-down parsing techniques **cannot** handle left-recursive grammars.
- So, we have to convert our left-recursive grammar into an equivalent grammar which is not left-recursive.
- The left-recursion may appear in a single step of the derivation (*immediate left-recursion*), or may appear in more than one step of the derivation.

Immediate Left-Recursion

$A \rightarrow A\,\alpha \mid \beta$ where β does not start with A

$\Downarrow$ eliminate immediate left recursion

$A \rightarrow \beta\, A'$

$A' \rightarrow \alpha\, A' \mid \varepsilon$ an equivalent grammar

In general,

$A \rightarrow A\,\alpha_1 \mid \ldots \mid A\,\alpha_m \mid \beta_1 \mid \ldots \mid \beta_n$ where $\beta_1 \ldots \beta_n$ do not start with A

$\Downarrow$ eliminate immediate left recursion

$A \rightarrow \beta_1\, A' \mid \ldots \mid \beta_n\, A'$

$A' \rightarrow \alpha_1\, A' \mid \ldots \mid \alpha_m\, A' \mid \varepsilon$ an equivalent grammar

Example

$E \rightarrow E+T \ | \ T$

$T \rightarrow T*F \ | \ F$

$F \rightarrow id \ | \ (E)$

$\Downarrow$ eliminate immediate left recursion

$E \rightarrow T\,E'$

$E' \rightarrow +T\,E' \ | \ \varepsilon$

$T \rightarrow F\,T'$

$T' \rightarrow *F\,T' \ | \ \varepsilon$

$F \rightarrow id \ | \ (E)$

Left-Recursion -- Problem

- A grammar cannot be immediately left-recursive, but it still can be left-recursive.
- By just eliminating the immediate left-recursion, we may not get a grammar which is not left-recursive.

$S \rightarrow Aa \mid b$

$A \rightarrow Sc \mid d$ This grammar is not immediately left-recursive, but it is still left-recursive.

$\underline{S} \Rightarrow Aa \Rightarrow \underline{S}ca$ or

$\underline{A} \Rightarrow Sc \Rightarrow \underline{A}ac$ causes to a left-recursion

- So, we have to eliminate all left-recursions from our grammar

Algorithm

- *Input: Grammar G with no cycles or ε-productions*
- Arrange the non terminals in some order $A1, A2, \ldots, An$

for $i = 1, \ldots, n$ **do**
    for $j = 1, \ldots, i-1$ **do**
        replace each
$$Ai \to Aj\ \gamma$$
        with
$$Ai \to \delta 1\ \gamma \mid \delta 2\ \gamma \mid \ldots \mid \delta k\ \gamma$$
        where
$$Aj \to \delta 1 \mid \delta 2 \mid \ldots \mid \delta k$$
    enddo
    eliminate the *immediate left recursion* in *Ai*
enddo

Example1

$S \rightarrow Aa \mid b$
$A \rightarrow Ac \mid Sd \mid f$

- Order of non-terminals: S, A

for S:

- we do not enter the inner loop.
- there is no immediate left recursion in S.

for A:

- Replace $A \rightarrow Sd$ with $A \rightarrow Aad \mid bd$
  So, we will have $A \rightarrow Ac \mid Aad \mid bd \mid f$
- Eliminate the immediate left-recursion in A

$$A \rightarrow bdA' \mid fA'$$
$$A' \rightarrow cA' \mid adA' \mid \varepsilon$$

So, the resulting equivalent grammar which is not left-recursive is:

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

Example2

$S \rightarrow Aa \mid b$
$A \rightarrow Ac \mid Sd \mid f$

- Order of non-terminals: A, S

for A:

- we do not enter the inner loop.
- Eliminate the immediate left-recursion in A

$A \rightarrow SdA' \mid fA'$
$A' \rightarrow cA' \mid \varepsilon$

for S:

- Replace $S \rightarrow Aa$ with $S \rightarrow SdA'a \mid fA'a$
  So, we will have $S \rightarrow SdA'a \mid fA'a \mid b$
- Eliminate the immediate left-recursion in S

$S \rightarrow fA'aS' \mid bS'$
$S' \rightarrow dA'aS' \mid \varepsilon$

So, the resulting equivalent grammar which is not left-recursive is:

$S \rightarrow fA'aS' \mid bS'$
$S' \rightarrow dA'aS' \mid \varepsilon$
$A \rightarrow SdA' \mid fA'$
$A' \rightarrow cA' \mid \varepsilon$

Non deterministic grammar

- Grammar with common prefix between at least two different productions from the same LHS

Eg.

S → aSb | aA |b

A→aB |a

B→ b

Eg.

S→ ab |abA

A→ bab | babc

Disadvantage: During parsing non-deterministic grammar requires lot of back tracking (time consuming)

Deterministic grammar

- Grammar without any common prefix in any of the different productions from same LHS

Note: To make the grammar suitable for predictive or top-down parsing, we need to convert the non deterministic grammar into deterministic grammar using the process called as left factoring.

Left-Factoring

- A predictive parser (a top-down parser without backtracking) insists that the grammar must be *left-factored*.

  grammar ➔ a new equivalent grammar suitable for predictive parsing

stmt $\rightarrow$ `if` expr `then` stmt `else` stmt |
`if` expr `then` stmt

- when we see `if`, we cannot now which production rule to choose to re-write *stmt* in the derivation.

Left-Factoring cont…

- In general,

  $A \rightarrow \alpha\beta_1 \mid \alpha\beta_2$ where α is non-empty and the first symbols of β_1 and β_2 (if they have one)are different.

- when processing α we cannot know whether expand

  A to $\alpha\beta_1$ or A to $\alpha\beta_2$

- But, if we re-write the grammar as follows

  $A \rightarrow \alpha A'$

  $A' \rightarrow \beta_1 \mid \beta_2$ so, we can immediately expand A to $\alpha A'$

Left-Factoring -- Algorithm

- For each non-terminal A with two or more alternatives (production rules) with a common non-empty prefix, let say

$$A \rightarrow \alpha\beta_1 \mid \ldots \mid \alpha\beta_n \mid \gamma_1 \mid \ldots \mid \gamma_m$$

convert it into

$$A \rightarrow \alpha A' \mid \gamma_1 \mid \ldots \mid \gamma_m$$

$$A' \rightarrow \beta_1 \mid \ldots \mid \beta_n$$

Left-Factoring – Example1

$A \rightarrow \underline{a}bB \mid \underline{a}B \mid cdg \mid cdeB \mid cdfB$

$\Downarrow$

$A \rightarrow aA' \mid \underline{cd}g \mid \underline{cd}eB \mid \underline{cd}fB$

$A' \rightarrow bB \mid B$

$\Downarrow$

$A \rightarrow aA' \mid cdA''$

$A' \rightarrow bB \mid B$

$A'' \rightarrow g \mid eB \mid fB$

Left-Factoring – Example2

$$A \rightarrow ad \mid a \mid ab \mid abc \mid b$$

$$\Downarrow$$

$$A \rightarrow aA' \mid b$$

$$A' \rightarrow d \mid \varepsilon \mid b \mid bc$$

$$\Downarrow$$

$$A \rightarrow aA' \mid b$$

$$A' \rightarrow d \mid \varepsilon \mid bA''$$

$$A'' \rightarrow \varepsilon \mid c$$

Summary

- Eliminating left recursion
- Left factoring

Check your understanding?

1. Eliminate left recursion in the following grammars

(a) $A \rightarrow Abd \mid Aa \mid a$

$B \rightarrow Be \mid b$

(b) $S \rightarrow (L) / a$

$L \rightarrow L , S / S$

2. Left factorize the following grammar

(a) $S \rightarrow iEtS \mid iEtSeS \mid a$

$E \rightarrow b$

(b) $S \rightarrow aSSbS \mid aSaSb \mid abb \mid b$