Course: Compiler Construction

Week 5: Syntax Analysis - Left Recursive Grammar

Lecturer: Martha Gichuki

Learning outcomes Week 5: Syntax Analysis - Left Recursive Grammar

At the end of the lecture, you will be able to:

(i) Define left recursion and left factorization

(ii) Describe immediate and general left-recursion methods

(iii) Describe an algorithm to eliminate left-recursion

1. Introduction

1.1. Definition: Left-Recursion

- ✓ A production is left-recursive if the left-most symbol on the right side is similar to the non-terminal on the left side[1]
- ✓ Recursive Grammar has a non-terminal e.g. "**A**" whose derivation contains "**A**" itself as the left-most symbol
- ✓ **Example:** $A \rightarrow A \propto$

There are two types of left-recursions[2]

I. Immediate left-recursion

II. General (Indirect) left-recursion

[1] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 92

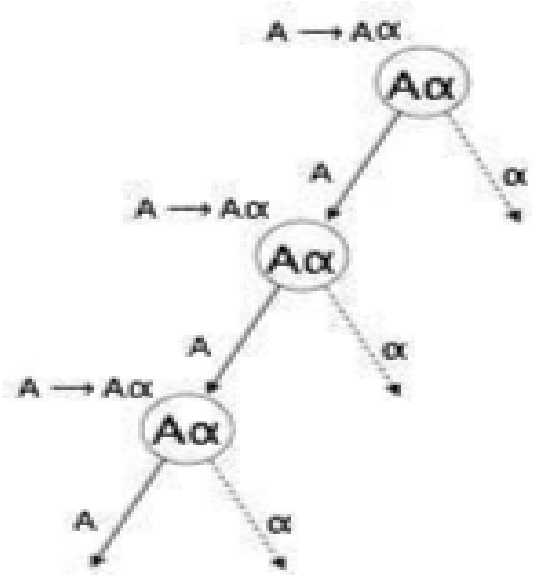
[2] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 237

I). Immediate left-recursion

- ✓ This occurs when a non-terminal A has a production rule of the form: -

  $A \rightarrow A \propto | \beta$.

- ✓ Here, A is any non-terminal symbol and $\propto$ represents a string of non-terminals[3]
- ✓ The top-down parser will first parse the A, which in turn will yield a string consisting of A itself and the parser may go into a loop forever[4].



1.2. Eliminating Left Recursion

- ✓ The immediate left-recursion can be eliminated by ***introducing*** a new non-terminal symbol[5] for instance A'

Example 1: Consider the production $A \rightarrow A\alpha|\beta$

- ✓ **Which can be modified as follows[6]:**

[3] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 237
[4] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 92
[5] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 237
[6] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 237

$A \rightarrow \beta$ A'

$A' \rightarrow \alpha A' | \varepsilon$

Therefore, the rule

$A \rightarrow A\alpha_1 | A\alpha_2 | \ldots | A\alpha_m | B_1 | B_2 | \ldots | B_n$

Can be modified as[7]: -

$A \rightarrow B_1 A' | B_2 A' | \ldots | B_n A'$

$A' \rightarrow \alpha_1 A' | \alpha_2 A' | \ldots | \alpha_m A' | \varepsilon$

Example 1: Eliminate left-recursion from the grammar

P → P + Q|Q

✓ We have the Solution

P → QP'

P' → +QP'|ε

Example 2: Eliminate left-recursion from the grammar

S → S0S1S|01

✓ We have the Solution

S → 01S'

S' → 0S1SS'|ε

Example 3: Eliminate left-recursion from the grammar

A → (B)|b

B → BXA|A

✓ **We have the Solution**

A → (B)|b (No left recursion – expression remains as it is)

B → AB'

B' → XAB'|**ε**

✓ **Example 4: Eliminate left-recursion from the grammar**

$E \rightarrow E + T | T$

[7] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 237

$T \rightarrow T * F|F$

$T \rightarrow (E)|id$

✓ **We have the Solution**

$E \rightarrow TE$'

E' → +TE'|ε

T' → FT'

T' → *FT'|ε

T → $(E)|id$ [8]

II. General (Indirect)left-recursion[9]

✓ This is a left-recursion that occurs due to a number of productions rules.

Example 1: Consider the grammar:

$S \rightarrow Aa$

$A \rightarrow Sb|c$

S is left recursive since $S \rightarrow Aa \rightarrow Sba$

Example 2: Consider the grammar:

$S \rightarrow A\alpha|\beta$

$A \rightarrow S$d

S is left recursive since $S \rightarrow A\alpha \rightarrow Sd\alpha$

Algorithm to eliminate left-recursion[10]:

✓ Generally, left-recursion is eliminated using the following algorithm:

Step 1

Arrange non-terminals in some order for example $A_1, A_2, \ldots A_m$

[8] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 237

[9] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 354

[10] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 238

Step 2

For i=1 to m, do

For j=1 to i-1 do

For each set of production

$A_i \rightarrow A_j\gamma$ *and* $A_j \rightarrow \delta_1|\delta_2|\ldots|\ \delta_k$ *replace* $A_i \rightarrow A_j\gamma$ *by*

$Ai \rightarrow \delta_1\gamma|\ \delta_2\gamma\ |\ldots|\ \delta_k\gamma$

Step 3

Eliminate immediate left recursion from all productions

Example 1: For the grammar

$\boldsymbol{S \rightarrow Aa}$

$\mathbf{A \rightarrow Sb|c}$

Step 1 - Let the order of *non-terminals* be S, A.

Step 2 - For i = 1,

the rule $S \rightarrow Aa$ remains since there is no immediate left-recursion.

For i = 2,

$A \rightarrow Sb|c$ is modified as $A \rightarrow Aab|c$

which has immediate left recursion and once eliminated, we get[16]

$A \rightarrow c$A'

A' $\rightarrow$ abA'| ε[11]

Example 2: The production set:

$\boldsymbol{S \rightarrow A\alpha|\beta}$

A →Sd

- ✓ Apply the above algorithm to get: -

[11] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 238

$S \rightarrow A\alpha|\beta$

$A \rightarrow A\alpha d|\beta$ d is modified as: -

$A \rightarrow Aab|c$[17]

- ✓ Then, remove immediate left recursion using the first technique to get: -

  $\boldsymbol{A \rightarrow \beta dA'}$

  $\boldsymbol{A' \rightarrow adA'|\varepsilon}$

- ✓ Now, none of the productions have either direct or indirect left recursion[12]

2. Left-factorization

- ✓ If two productions for the same non-terminal begin with the same sequence of symbols, then the **top-down parser cannot make a choice** as to which of the production it should take to parse the string in hand[13].

Example: If a top-down parser encounters a production like: -

$\boldsymbol{A \rightarrow \alpha\beta|\alpha\gamma|\ldots}$

- ✓ It cannot determine which production to follow to parse the string, since both productions are starting from the same terminal (or non-terminal).
- ✓ **Left factoring** is a technique used to eliminate this confusion[14].
- ✓ Left factoring transforms the grammar to make it useful **for top-down parsers**.
- ✓ In this technique, we make one production for each common prefixes and the rest of the derivation is added by new productions.
- ✓ Generally, we rewrite the grammar to ensure that: -

(i) the overlapping productions are made into a single production containing the common prefix of the productions

(ii) a new auxiliary non-terminal for the different suffixes is used.

---

[12] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 239

[13] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 239

[14] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 240

Method[15]

If **α≠ε** then **replace all** of the **A productions**

A=> αβ1| αβ2|…| αβn

With: - A=> αA'

A'=> β1|β2|…|βn

- ✓ Where **A'** is a new **non-terminal**
- ✓ Repeat until **no two alternatives for a single non-terminal have a common prefix**

Example: The above productions can be written as: -

A=> αA'

A'=> β|γ|…

- ✓ Now, the parser has only one production per prefix which makes it easier to take decisions[16]

Example1: Consider the grammar:

A=> aAB|aA|a

To remove left factoring, rewrite as:

A=> aC

C=>AB|A| ε

Example2: Consider the grammar:

B=> bB|bAB|bC|b

To remove left factoring, rewrite as:

B=> bC

C=>B|AB|C| ε

[15] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 240

[16] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 240

3. **First and Follow Sets**

3.1. **First Set**

- ✓ Allows us to determine the symbol derived in the **first position by a non-terminal**[26]
- ✓ Example: - for **α → t β**, means that **α derives t (terminal)** in the very first position.
- ✓ Therefore, **t ∈ FIRST(α)**[17]

Algorithm to calculate First Set[18]

(i) If **α is a terminal**, then **FIRST(α) = {α}**

(ii) If **α is a non-terminal**, and **α → ε** is a production, then **FIRST(α) = {ε}**

(iii) If **α is a non-terminal**, and **α → γ1γ2γ3…γn** and any **FIRST(γ)** contains **t** then **t** is in **FIRST(α)**.

3.2. Follow Set

- ✓ Allows us to determine the **terminal symbol** that **immediately follows a non-terminal α** in production rules.
- ✓ The focus is not what the non-terminal can generate but **what would be the next terminal symbol that follows the productions of a non-terminal**[19]**.**

Algorithm to calculate Follow Set[20]

1. If **α is a start symbol**, then **FOLLOW () = {$}**
2. If **α is a non-terminal**, and has a production **α → AB**, then **FIRST(B)** is in **FOLLOW(A)** except **ε**
3. If **α is a non-terminal**, and has a production **α → AB**, where B ε, the **FOLLOW(A)** is in **FOLLOW(α).**

[17] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 245
[18] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 246
[19] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 245
[20] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 247

Example 1: - Find the First & Follow Sets of the grammar

S → Bb|Cd, B→ aB|ε, C → cC|ε

GRAMMAR	FIRST	FOLLOW
S → Bb|Cd	{a,b,c,d}	{$} S does not appear anywhere on the RHS
B→ aB|ε	{a, ε}	{b}
C → cC|ε	{c, ε}	{d}

Example 2: - Find the First & Follow Sets of the grammar

S → aBDh, B→ cC, C→ bC|ε, D→ EF, E→ g|ε, F→ f|ε

GRAMMAR	FIRST	FOLLOW
S → aBDh	{a}	{S}
B→ cC	{c}	{g,f,h}
C→ bC|ε	{b, ε}	{g,f,h}
D→ EF	{g,f, ε}	{h}
E→ g|ε	{g, ε}	{f,h}
F→ f|ε	{f, ε}	{h}

Summary of Syntax Analyzers

- ✓ Syntax analyzers receive their **inputs,** in form of **tokens** from lexical analyzers.
- ✓ Lexical Analyzers are responsible for the **validity of a token** supplied by the syntax analyzer[21].
- ✓ Syntax analyzers have four limitations in terms of the tasks they are able to do…

Limitations of Syntax Analyzers

Syntax analyzers cannot determine the following: -

i. If a token is valid

ii. If a token is declared before being used

iii. If a token is initialized before being used

iv. If an operation performed on a token type is valid or not

- ✓ These limitations are accomplished by **sematic analyzers**[22]

[21] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 185

[22] Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 185

Content Covered in Week 5: Syntax Analysis - Left Recursive Grammar

At the end of the lecture, we were able to:

(i) Define left recursion and left factorization

(ii) Describe immediate and general left-recursion methods

(iii) Describe an algorithm to eliminate left-recursion

Course Text Books

1. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison-Wesley Pub Co, ISBN: 0201100886 (2007))

2. Compiler Construction: Principles and practices; Kenneth C. Louden; Cengage Learning; 1st edition, ISBN-10: 0534939724 (1997)

3. Basics of Compiler Design: Torben Mogensen; DIKU University of Copenhagen Universitetsparken 1 DK-2100 Copenhagen DENMARK (2007)

4. Engineering a Compiler: 2nd edition; Keith D. Cooper and Linda Torczon; Morgan Kaufmann Publishers ISBN: 978-0-12-088478-0 (2003)

5. Compiler Design: Santanu Chattopadhyay; PHI Learning Publishers, ISBN 812032725X, 9788120327252 (2005)