Course: Compiler Construction

Week 5: Syntax Analysis - Left Recursive Grammar

Lecturer: Martha Gichuki

Course Description

- The course begins with a coverage of basic Principles in Compiler Design with an introduction to Formal programming language translation and compiler design concepts; compilers, translators and interpreters.
- A coverage of Language theory, Parsing Context-Free Languages (Top - down and bottom - up parsing), translation specifications and machine- independent code optimization will then follow.

Course Description …

- The main phases of compilation i.e. Lexical, Syntax and Semantic analysis will be taught.
- Code generation and Optimization, Symbol table design, Program compilation, Loading and execution will then be covered.
- Compilation techniques, Optimization, Design of a simple complete compiler; will culminate the course.

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

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:** $A \rightarrow A \propto$

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

There are two types of left-recursion[2]

a) Immediate left-recursion
b) General (Indirect) left-recursion

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:-

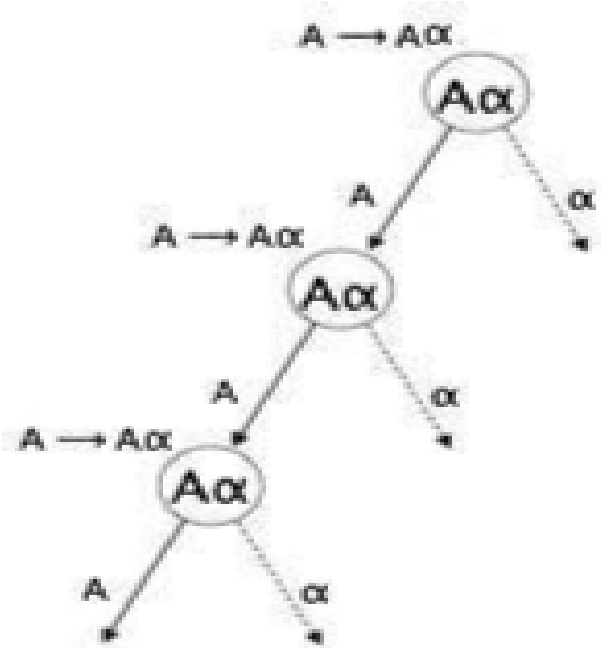
  $$A \rightarrow A \propto | \beta.$$

- ✓ Here, **A** is any **non-terminal** symbol and $\propto$ represents a **string of non-terminals**[3]

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

Immediate left-recursion …

- ✓ 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]



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

Eliminating Left Recursion

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

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

Example 1:

Consider the production

$$A \rightarrow A\alpha \mid \beta$$

Which can be modified as follows[6]:

$$A \rightarrow \beta\ A'$$

$$A' \rightarrow \alpha A' \mid \varepsilon$$

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

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]:-

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

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

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

Example 1: Eliminate left-recursion from the grammar

$P \rightarrow P + Q \mid Q$

✓ **We have the Solution**[8]

$P \rightarrow QP'$

$P' \rightarrow +QP' \mid \varepsilon$

Example 2: Eliminate left-recursion from the grammar

$S \rightarrow S0S1S \mid 01$

✓ **We have the Solution**[8]

$S \rightarrow 01S'$

$S' \rightarrow 0S1SS' \mid \varepsilon$

Example 3: Eliminate left-recursion from the grammar

$A \rightarrow (B) \mid b$

$B \rightarrow BXA \mid A$

✓ **We have the Solution**[8]

$A \rightarrow (B) \mid b$ (No left recursion – expression remains as it is)

$B \rightarrow AB'$

$B' \rightarrow XAB' \mid \varepsilon$

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

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

$$E \longrightarrow E + T \mid T$$

$$T \longrightarrow T * F \mid F$$

$$T \longrightarrow (E) \mid id$$

✓**We have the Solution[9]**

$$E \longrightarrow TE'$$

$$\mathrm{E}' \longrightarrow +\mathrm{TE}' \mid \boldsymbol{\varepsilon}$$

$$\mathrm{T}' \longrightarrow \mathrm{FT}'$$

$$\mathrm{T}' \longrightarrow *\mathrm{FT}' \mid \boldsymbol{\varepsilon}$$

$$\mathrm{T} \longrightarrow (E) \mid id$$

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

II. General (Indirect)left-recursion

- ✓ This is a left-recursion that occurs **due to a number of production rules**[10].

- ✓ **Example 1**: Consider the grammar:

$S \rightarrow Aa$

$A \rightarrow Sb \mid c$

S is **left recursive** since

$S \rightarrow Aa \rightarrow Sba$

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

- ✓ **Example 2**: Consider the grammar[11]:

$$S \rightarrow A\alpha \mid \beta$$

$$A \rightarrow Sd$$

S is **left recursive** since

$$S \rightarrow A\alpha \rightarrow Sd\alpha$$

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

Algorithm to eliminate left-recursion:

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

Step 1[12]

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

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

Algorithm to eliminate left-recursion:

Step 2 [13]

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***

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

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

Algorithm to eliminate left-recursion:

Step 3[14]

Eliminate immediate left recursion from all productions

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

Example 1:

The grammar

$S \rightarrow Aa$

$A \rightarrow Sb \mid c$

- ✓ Step 1 - Let the order of ***non-terminals*** be **S, A**[15]**.**

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

For $i = 1$,

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

For $i = 2$,

$\boldsymbol{A \rightarrow Sb \mid c}$ **is modified as** $\boldsymbol{A \rightarrow Aab \mid c}$

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

$\boldsymbol{A \rightarrow cA'}$

$\boldsymbol{A' \rightarrow abA' \mid \varepsilon}$

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

Example 2: The production set:

$S \rightarrow A\alpha \mid \beta$

A $\rightarrow$Sd

✓Apply the above algorithm to get:-

$S \rightarrow A\alpha \mid \beta$

✓$A \rightarrow$ Aαd $\mid \beta$ d is modified as:-

$A \rightarrow Aab \mid c$[17]

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

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

$$A \rightarrow \beta dA'$$

$$A' \rightarrow adA' \mid \varepsilon$$

- ✓Now, none of the productions have either direct or indirect left recursion[18]

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

Left-factorization

- ✓If two productions for the same nonterminal 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[19].

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

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

$$A \rightarrow \alpha\beta \mid \alpha\gamma \mid \ldots$$

- ✓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[20].

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

- ✓ 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[21].

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

- ✓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[22].

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

Method[23]

If **$\alpha \neq \varepsilon$** then **replace all** of the **A productions**

$A \Rightarrow \alpha\beta 1 \mid \alpha\beta 2 \mid \ldots \mid \alpha\beta n$

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

Method cont..

With

$$A \Rightarrow \alpha A'$$

$$A' \Rightarrow \beta 1 \mid \beta 2 \mid \ldots \mid \beta n$$

- ✓Where A′ is a new non-terminal
- ✓Repeat until **no two alternatives for a single non-terminal have a common prefix**[24]

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

Example: The above productions can be written as:-

$$A \Rightarrow \alpha A'$$
$$A' \Rightarrow \beta \mid \gamma \mid \ldots$$

✓Now, the parser has only one production per prefix which makes it easier to take decisions[25]

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

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

First and Follow Sets

A. First Set

- ✓Allows us to determine the symbol derived in the **first position by a non-terminal**[26]
- ✓Example:- for $\alpha \rightarrow t\ \beta$, means that **$\alpha$ derives t (terminal)** in the very first position.
- ✓Therefore, **$t \in FIRST(\alpha)$**

26. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison-Wesley Pub Co, ISBN: 0201100886 (2007) page 245

Algorithm to calculate First Set[27]

1. If **α is a terminal**, then **FIRST(α) = {α}**
2. If **α is a non-terminal**, and $\alpha \rightarrow \varepsilon$ is a production, then **FIRST(α) = {ε}**
3. If **α is a non-terminal**, and $\alpha \rightarrow \gamma 1 \gamma 2 \gamma 3 \ldots \gamma n$ and any **FIRST(γ)** contains **t** then **t** is in **FIRST(α)**.

27. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison-Wesley Pub Co, ISBN: 0201100886 (2007) page 246

B. 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[28].**

28. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison-Wesley Pub Co, ISBN: 0201100886 (2007) page 245

Algorithm to calculate Follow Set[29]

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

29. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison-Wesley Pub Co, ISBN: 0201100886 (2007) page 247

Find the First & Follow Sets of the grammar

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}

Find the First & Follow Sets of the grammar

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[30].
- ✓Syntax analyzers have four limitations in terms of the tasks they are able to do…

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

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**[31]

31. 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)