Chapter 4
Syntax Analysis

Resolving Difficulties : Left Factoring

A non-terminal with two or more productions, whose right-hand sides start with the same grammar symbols, adds non-determinism to the Parser

Replace productions

$A \rightarrow \alpha\, \beta_1 \,/\, \alpha\, \beta_2 \,/\, \ldots \mid \alpha\, \beta_n \mid \gamma$

with

$A \rightarrow \alpha\, A_R \mid \gamma$

$A_R \rightarrow \beta_1 \,/\, \beta_2 \,/\, \ldots \,/\, \beta_n$

Resolving Difficulties : Left Factoring (Example-1)

Replace productions

$A \rightarrow \alpha\, \beta_1 \,/\, \alpha\, \beta_2 \,/\, \ldots \mid \alpha\, \beta_n \mid \gamma$

with

$A \rightarrow \alpha\, A_R \mid \gamma$

$A_R \rightarrow \beta_1 \,/\, \beta_2 \,/\, \ldots \,/\, \beta_n$

Consider the grammar:

E→ T+ E | T-E | T

T→ F *T | F

F→ a

Left-factored Grammar:

E→ TE_R
E_R→ +E | -E | ^
T→ FT_R
T_R→ *T | ^
F→ a

Resolving Difficulties : Left Factoring (Example-1)

Suppose you want to derive ‘a’ in the previous grammar

Grammar:

E→ TE_R
E_R→ +E | -E | ^
T→ FT_R
T_R→ *T | ^
F→ a

Original Grammar (Non-Deterministic)	Left- Factored Grammar (Deterministic)
E	E
T+ E or T-E or T (Trial and Error)	TE_R
T	FT_RE_R
F*T or F (Trial and Error)	aT_RE_R
F	a E_R
a	a

Resolving Difficulties : Left Factoring (Example-2)

Replace productions

$A \rightarrow \alpha\, \beta_1 \,/\, \alpha\, \beta_2 \,/\, \ldots \mid \alpha\, \beta_n \mid \gamma$

with

$A \rightarrow \alpha\, A_R \mid \gamma$

$A_R \rightarrow \beta_1 \,/\, \beta_2 \,/\, \ldots \,/\, \beta_n$

Consider the grammar:

S→ iEtSeS | iEtS | a

E→ b

Left-factored Grammar:

S→ iEtSS_R | a

S_R→ eS | ^

E→ b

Resolving Difficulties : Left Factoring (Example-2)

Suppose you want to derive **'ibtibtaea'** in the previous grammar

Here, Grammar is Ambiguous

Grammar:

S→ iEtSS_R | a

S_R→ eS | ^

E→ b

Original Grammar (Non-Deterministic)	Left- Factored Grammar (Deterministic)
S	S
iEtSeS or iEtS (Trial and Error)	iEtSS_R
	ibtSS_R
	ibtiEtS$S_R S_R$
	ibtibtS$S_R S_R$
	ibtibta$S_R S_R$
	ibtibtaeSS_R or ibtibtaS_R
	ibtibtaeaS_R or ibtibtaeS
	Ibtibtaea or ibtibtaea

Resolving Difficulties : Left Factoring (Example-3)

Replace productions

$A \rightarrow \alpha\, \beta_1 \,/\, \alpha\, \beta_2 \,/ \ldots |\, \alpha\, \beta_n \,|\, \gamma$

with

$A \rightarrow \alpha\, A_R \,|\, \gamma$

$A_R \rightarrow \beta_1 \,/\, \beta_2 \,/ \ldots /\, \beta_n$

S→ T$

T→ [T] | [ ] T | [T]T | [ ]

S→ T$

T→ [$\mathbf{T_R}$

$\mathbf{T_R}$→ T] |] T | T]T |]

Resolving Difficulties : Left Factoring (Example-3)

S→ T$

T→ [$\mathbf{T_R}$

$\mathbf{T_R}$→ T] |] T | T] T |]

S→ T$

T→ [$\mathbf{T_R}$

$\mathbf{T_R}$→ T] **M** |] **N**

M→ T | ^

N→ T | ^

S→ T$

T→ [$\mathbf{T_R}$

$\mathbf{T_R}$→ T] **M** |] **M**

M→ T | ^

Removing Difficulties Immediate Left-Recursion Elimination (Example – 1)

Rewrite every left-recursive production

$A \to A\,\alpha \mid \beta \mid \gamma \mid A\,\delta$

into a right-recursive production:

$A \to \beta\, A_R \mid \gamma\, A_R$

$A_R \to \alpha\, A_R \mid \delta\, A_R \mid \varepsilon$

Ex. $E \to E + T \mid E - T \mid T$

Here,

A matches with E

α matches with + T

δ matches with - T

β matches with T

$E \to T\, E_R$

$E_R \to + T\, E_R \mid - T E_R \mid \varepsilon$

9

Removing Difficulties Immediate Left-Recursion Elimination (Example – 2)

Rewrite every left-recursive production

$A \to A\,\alpha \mid \beta \mid \gamma \mid A\,\delta$

into a right-recursive production:

$A \to \beta\, A_R \mid \gamma\, A_R$

$A_R \to \alpha\, A_R \mid \delta\, A_R \mid \varepsilon$

$S \to S0S1S \mid 01$

$S \to 01\, S_R$

$S_R \to 0S1S\, S_R \mid \varepsilon$

Removing Difficulties
Immediate Left-Recursion Elimination
(Example – 3)

Rewrite every left-recursive production

$A \rightarrow A\,\alpha \mid \beta \mid \gamma \mid A\,\delta$

into a right-recursive production:

$A \rightarrow \beta\, A_R \mid \gamma\, A_R$

$A_R \rightarrow \alpha\, A_R \mid \delta\, A_R \mid \varepsilon$

S → (L) | a
L → L , S | S

→

S → (L) | a
L → SL$_R$
L$_R$ → ,SL$_R$ |ε

Removing Difficulties
Immediate Left-Recursion Elimination
(Example – 4)

Rewrite every left-recursive production

$A \rightarrow A\,\alpha \mid \beta \mid \gamma \mid A\,\delta$

into a right-recursive production:

$A \rightarrow \beta\, A_R \mid \gamma\, A_R$

$A_R \rightarrow \alpha\, A_R \mid \delta\, A_R \mid \varepsilon$

S → A
A → Ad |Ae | aB | ac
B → bBc | f

→

S → A
A → aBA$_R$ |acA$_R$
A$_R$→ dA$_R$ |eA$_R$ | ε
B → bBc | f

Removing Difficulties Immediate Left-Recursion Elimination (Example – 5)

$A \rightarrow ABd \mid Aa \mid a$
$B \rightarrow Be \mid b$

Here,
A matches with A
α matches with Bd
δ matches with a
β matches with a

$A \rightarrow a\, A_R$
$A_R \rightarrow Bd\, A_R \mid a\, A_R \mid \varepsilon$

Here,
A matches with B
α matches with e
β matches with b

$B \rightarrow b\, B_R$
$B_R \rightarrow eB_R \mid \varepsilon$

Indirect Left-Recursion Elimination

$A \rightarrow Ba \mid Aa \mid c$
$B \rightarrow Bb \mid Ab \mid d$

Step-01:

First let us eliminate left recursion from $A \rightarrow Ba \mid Aa \mid c$

Now, given grammar becomes-
$A \rightarrow BaA_R \mid cA_R$
$A_R \rightarrow aA_R \mid \varepsilon$
$B \rightarrow Bb \mid Ab \mid d$

Indirect Left-Recursion Elimination

$A \rightarrow BaA_R \mid cA_R$
$A_R \rightarrow aA_R \mid \varepsilon$
$B \rightarrow Bb \mid Ab \mid d$

Step-02:

Substituting the productions of A in B → Ab, we get the following grammar-

$A \rightarrow BaA_R \mid cA_R$
$A_R \rightarrow aA_R \mid \varepsilon$
$B \rightarrow Bb \mid BaA_R\ b \mid cA_R b \mid d$

Indirect Left-Recursion Elimination

$A \rightarrow BaA_R \mid cA_R$
$A_R \rightarrow aA_R \mid \varepsilon$
$B \rightarrow Bb \mid BaA_R\ b \mid cA_R b \mid d$

Step-03:

Now, eliminating left recursion from the productions of B, we get the following grammar-

$A \rightarrow BaA_R \mid cA_R$
$A_R \rightarrow aA_R \mid \varepsilon$
$B \rightarrow cA_R bB_R \mid dB_R$
$B_R \rightarrow b\ B_R \mid aA_R\ b\ B_R \mid \varepsilon$

Resolving Difficulties : Indirect Left Recursion

Problem: If left recursion is two-or-more levels deep, this isn't enough

Algorithm:

Input: **Grammar G with ordered Non-Terminals A_1, ..., A_n**

Output: **An equivalent grammar with no left recursion**

1. **Arrange the non-terminals in some order A_1=start NT,A_2,…A_n**
2. **for *i* := 1 to *n* do begin**

    for *j* := 1 to *i* – 1 do begin

    replace each production of the form $A_i \to A_j\gamma$

    by the productions $A_i \to \delta_1\gamma \mid \delta_2\gamma \mid \ldots \mid \delta_k\gamma$

    where $A_j \to \delta_1|\delta_2|\ldots|\delta_k$ are all current A_j productions;

    end

    eliminate the immediate left recursion among A_i productions

   end

17

Using the Algorithm

Apply the algorithm to: $A_1 \to A_2a \mid b \mid \in$

$A_2 \to A_2c \mid A_1d$

i = 1

For A_1 there is no left recursion

i = 2

for j=1 to 1 do

Take productions: $A_2 \to A_1\gamma$ and replace with

$A_2 \to \delta_1\gamma \mid \delta_2\gamma \mid \ldots \mid \delta_k\gamma \mid$

where $A_1 \to \delta_1 \mid \delta_2 \mid \ldots \mid \delta_k$ **are A_1 productions**

in our case $A_2 \to A_1d$ becomes $A_2 \to A_2ad \mid bd \mid d$

What's left: $A_1 \to A_2a \mid b \mid \in$

$A_2 \to A_2c \mid A_2ad \mid bd \mid d$

Are we done ?

18

Using the Algorithm (2)

No ! We must still remove A_2 left recursion !

$A_1 \rightarrow A_2 a \mid b \mid \in$

$A_2 \rightarrow A_2\, c \mid A_2\, ad \mid bd \mid d$

$A_1 \rightarrow A_2 a \mid b \mid \in$

$A_2 \rightarrow bdA_R \mid dA_R$

$A_R \rightarrow c\, A_R \mid adA_R \mid \in$

19

Example Left Recursion Elimination

$A \rightarrow B\ C \mid \mathbf{a}$

$B \rightarrow C\ A \mid A\ \mathbf{b}$

$C \rightarrow A\ B \mid C\ C \mid \mathbf{a}$

$i = 1$: nothing to do

$i = 2, j = 1$: $B \rightarrow C\ A \mid \underline{A}\ \mathbf{b}$

$\Rightarrow \quad B \rightarrow C\ A \mid \underline{B\ C}\ \mathbf{b} \mid \underline{\mathbf{a}}\ \mathbf{b}$

$\Rightarrow_{\text{(imm)}} \quad B \rightarrow C\ A\ B_R \mid \mathbf{a}\ \mathbf{b}\ B_R$

$\quad B_R \rightarrow C\ \mathbf{b}\ B_R \mid \varepsilon$

$i = 3, j = 1$: $C \rightarrow \underline{A}\ B \mid C\ C \mid \mathbf{a}$

$\Rightarrow \quad C \rightarrow \underline{B\ C}\ B \mid \underline{\mathbf{a}}\ B \mid C\ C \mid \mathbf{a}$

$i = 3, j = 2$: $C \rightarrow \underline{B}\ C\ B \mid \mathbf{a}\ B \mid C\ C \mid \mathbf{a}$

$\Rightarrow \quad C \rightarrow \underline{C\ A\ B_R}\ C\ B \mid \underline{\mathbf{a}\ \mathbf{b}\ B_R}\ C\ B \mid \mathbf{a}\ B \mid C\ C \mid \mathbf{a}$

$\Rightarrow_{\text{(imm)}} \quad C \rightarrow \mathbf{a}\ \mathbf{b}\ B_R\ C\ B\ C_R \mid \mathbf{a}\ B\ C_R \mid \mathbf{a}\ C_R$

$\quad C_R \rightarrow A\ B_R\ C\ B\ C_R \mid C\ C_R \mid \varepsilon$

20

Top-Down Parsing

1. **Can be viewed as an attempt to find a leftmost derivation for an input string.**
2. **Why ?**
   1. **By always replacing the leftmost non-terminal symbol via a production rule, we are guaranteed of developing a parse tree in a left-to-right fashion that is consistent with scanning the input.**
   2. **$A \Rightarrow aBc \Rightarrow adDc \Rightarrow adec$ (scan a, scan d, scan e, scan c - accept!)**
3. **Recursive-descent parsing concepts – may involve backtracking**
4. **Predictive parsing**
   1. **Recursive / Brute force technique**
   2. **non-recursive / table driven**
5. **Error recovery**
6. **Implementation**

Recursive Descent Parsing Concepts

- **General category of Parsing Top-Down**
- **Choose production rule based on input symbol**
- **May require backtracking to correct a wrong choice.**
- **Example:** $S \rightarrow c\,A\,d$
  $A \rightarrow ab \mid a$

  input: cad