THEORY OF COMPUTATION AND COMPILERS

Unit - II

CONTEXT FREE GRAMMARS AND PARSING

- Introduction
- Context-Free Grammars - Derivation, Parse trees, Ambiguity
- **Types of Parsers** 
- LL(K) grammars and LL(1) parsing
- Bottom-up Parsing - handle pruning
- LR Grammar Parsing
- LALR parsing
- Parsing ambiguous grammars
- Error Recovery in Parsing
- YACC programming specification

Dr. R. Madana Mohana
Professor, Artificial Intelligence & Data Science | I/c-Head, Artificial Intelligence & Machine Learning
CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY
Hyderabad - 500 075, Telangana, INDIA
www.cbit.ac.in

Unit-II: Syntax Analysis (or) Parser
Top-Down Parsing:
Predictive Parser (Part-2)

Outline:

- FIRST
- FOLLOW
- Example Problems

FIRST and FOLLOW

- The **predictive parser** can be easily constructed once we know **FIRST** and **FOLLOW** sets.
- These sets of symbols help us to construct the **predictive parsing table** very easily.

Computing FIRST Symbols

Definition: FIRST

- **FIRST(α)** is defined as set of terminals that appear in the beginning of derivation derived from **α**.
- Formally, **FIRST(α)** is defined as shown below:

FIRST(α)=	**ϵ**	if **α = ϵ**
	ϵ	if **α $\Rightarrow_*$ ϵ**
	a	if **α $\Rightarrow_*$ aβ**

Computing FIRST Symbols

Example-1: FIRST

Compute FIRST sets for each non-terminals in the following grammar.

1. E → TE′
2. E′ → +TE′ | ϵ
3. T → FT′
4. T′ → *FT′| ϵ
5. F → (E) | id

Computing FIRST Symbols

Example-1: Solution

The FIRST sets for the given grammar can be computed by obtaining various derivations as shown below:

1. E => T E′ => FT′E′ => (E)T′E′

2. E => T E′ => FT′E′ => id T′E′

FIRST (E) = FIRST (T) = FIRST (F)={(, id}

Computing FIRST Symbols

Example-1: Solution

Consider the derivations not used in previous derivation:

3. E′ => +TE′

4. E′ => ϵ

FIRST (E′) = {ϵ, +}

5. T′ => *FT′

6. T′ => ϵ

FIRST (T′) = {ϵ, *}

Computing FIRST Symbols

Example-1: Solution

Now, the final FIRST sets are shown below:

	E	E′	T	T′	F
FIRST	(, id	ϵ, +	(, id	ϵ, *	(, id

Computing FIRST Symbols

Use of FIRST sets:

The FIRST sets can be used during predictive parsing while constructing the predictive parsing table as shown below:

- Consider the **A**-Production **A → α | β** and assume **FIRST**(**α**) and **FIRST**(**β**) are disjoint i.e., **FIRST(α) ∩ FIRST(β) = { }** which is an empty set.
- If the input symbol obtained from **lexical analyzer** is '**a**' and if '**a**' is in **FIRST**(**α**) then use the production **A → α** during parsing.

Computing FIRST Symbols

Use of FIRST sets:

- If the input symbol obtained from **lexical analyzer** is '**b**' and if '**b**' is in **FIRST**(**β**) then use the production **A → β** during parsing.

Thus, using **FIRST** sets we can choose what production to use between the two productions **A → α | β** when input symbol is **a** or **b**.

Algorithm or Rules to compute FIRST(X):

Algorithm FIRST(X) :

To compute **FIRST(X)** for all grammar symbols **X**, apply the following rules until no more terminals or **ϵ** can be added to any **FIRST** set.

Rule 1:

If **X → aα** is a production, where '**a**' is a terminal then **FIRST(X) = {a}**.

Rule 2:

If **X → ϵ** is a production, then add **ϵ** to **FIRST(X)** i.e., **FIRST(X) = {ϵ}**.

Algorithm or Rules to compute FIRST(X):

Algorithm FIRST(X) :

Rule 3: If **X** is a non-terminal and **$X \rightarrow Y_1Y_2..Y_n$** is a production, and if **$Y_1Y_2..Y_{i-1} =>_* \epsilon$**, then **FIRST(X) = non-ϵ** symbols in **FIRST(Y_i)**.

Rule 4:

If **$X \rightarrow Y_1Y_2..Y_n$** is a production, and if **$Y_1Y_2..Y_{i-1} =>_* \epsilon$**, then **FIRST(X) = {$\epsilon$}**.

Rule 5:

If **X** is a terminal or **ϵ** then **FIRST(X) = {X}**.

Algorithm or Rules to compute FIRST(X):

Examples: FIRST(X)

1. **Rule 1** is applied if the first symbol on the right hand side of the production is a terminal. If so, then add only the first symbol.

- Ex 1: if **A → aBC**, then **FIRST(A) = {a}**
- Ex 2: if **E → +TE′**, then **FIRST(E) = {+}**
- Ex 3: if **A → abc**, then **FIRST(A) = {a}**

Algorithm or Rules to compute FIRST(X):

Examples: FIRST(X)

2. Rule 2 is applied only for **ϵ**-productions.

- Ex 1: if **A → ϵ**, then **FIRST(A) = {ϵ}**
- Ex 2: if **E′ → ϵ**, then **FIRST(E′) = {ϵ}**

Algorithm or Rules to compute FIRST(X):

Examples: FIRST(X)

3. Rule 3 is applied for all productions not considered in first two steps.

Ex: Consider the productions:

S → ABCd

A → ϵ | +B

B → ϵ | *B

C → ϵ | %B

Algorithm or Rules to compute FIRST(X):

Examples: FIRST(X)

3. Rule 3 is applied for all productions not considered in first two steps.

Ex: FIRST(A), **FIRST(B)** and **FIRST(C)** are computed using rules 1 & 2 as shown below:

	S	A	B	C
FIRST		ϵ, +	ϵ, *	ϵ, %

Algorithm or Rules to compute FIRST(X):

Examples: FIRST(X)

3. Rule 3 is applied for all productions not considered in first two steps.

Ex: To compute **FIRST(S)** consider the production **S → ABCd** and apply rule 3 as shown below:

a) S → ABCd add non-**ϵ** symbols of **FIRST(A)** to **FIRST(S)**,

i.e., **FIRST(S) = FIRST(A)-ϵ = {+}**

Algorithm or Rules to compute FIRST(X):

Examples: FIRST(X)

3. Rule 3 is applied for all productions not considered in first two steps.

Ex: To compute **FIRST(S)** consider the production **S → ABCd** and apply rule 3 as shown below:

b) S → ABCd since **A => ϵ**, add non-**ϵ** symbols of **FIRST(B)** to **FIRST(S)**,

i.e., **FIRST(S) = FIRST(B)-ϵ = {*}**

Algorithm or Rules to compute FIRST(X):

Examples: FIRST(X)

3. Rule 3 is applied for all productions not considered in first two steps.

Ex: To compute **FIRST(S)** consider the production **S → ABCd** and apply rule 3 as shown below:

c) S → ABCd since **AB => ϵ**, add non-**ϵ** symbols of **FIRST(C)** to **FIRST(S)**,

i.e., **FIRST(S) = FIRST(C)-ϵ = {%}**

Algorithm or Rules to compute FIRST(X):

Examples: FIRST(X)

3. Rule 3 is applied for all productions not considered in first two steps.

Ex: To compute **FIRST(S)** consider the production **S → ABCd** and apply rule 3 as shown below:

d) S → ABCd since **ABC => ϵ**, add non-**ϵ** symbols of **FIRST(d)** to **FIRST(S)**,

i.e., **FIRST(S) = FIRST(d)-ϵ = {d}**

Algorithm or Rules to compute FIRST(X):

Examples: FIRST(X)

3. Rule 3 is applied for all productions not considered in first two steps.

Ex: To compute **FIRST(S)** consider the production **S → ABCd** and apply rule 3 as shown below:

So, all the **FIRST** values are shown below:

	S	A	B	C
FIRST	%, *, +, d	ϵ, +	ϵ, *	ϵ, %

Algorithm or Rules to compute FIRST(X):

Examples: FIRST(X)

4. Rule 4 is applied for all productions whose right hand side gives **ϵ**.

Ex: Consider the productions:

S → ABC

A → ϵ | +B

B → ϵ | *B

C → ϵ | %B

Algorithm or Rules to compute FIRST(X):

Examples: FIRST(X)

4. Rule 4 is applied for all productions whose right hand side gives **ϵ**.

Ex: FIRST(A), **FIRST(B)** and **FIRST(C)** are computed using rules 1 & 2 as shown below:

	S	A	B	C
FIRST		ϵ, +	ϵ, *	ϵ, %

Algorithm or Rules to compute FIRST(X):

Examples: FIRST(X)

4. Rule 4 is applied for all productions whose right hand side gives **ϵ**.

Ex: To compute **FIRST(S)**, consider the production **S → ABC** and apply rule 3 as shown below:

a) S → ABC add non-**ϵ** symbols of **FIRST(A)** to **FIRST(S)**,

i.e., **FIRST(S) = FIRST(A)-ϵ = {+}**

Algorithm or Rules to compute FIRST(X):

Examples: FIRST(X)

4. Rule 4 is applied for all productions whose right hand side gives **ϵ**.

Ex: To compute **FIRST(S)**, consider the production **S → ABC** and apply rule 3 as shown below:

b) S → ABC since **A => ϵ**, add non-**ϵ** symbols of **FIRST(B)** to **FIRST(S)**,

i.e., **FIRST(S) = FIRST(B)-ϵ = {*}**

Algorithm or Rules to compute FIRST(X):

Examples: FIRST(X)

4. Rule 4 is applied for all productions whose right hand side gives **ϵ**.

Ex: To compute **FIRST(S)**, consider the production **S → ABC** and apply rule 3 as shown below:

c) S → ABC since **AB => ϵ**, add non-**ϵ** symbols of **FIRST(C)** to **FIRST(S)**,

i.e., **FIRST(S) = FIRST(C)-ϵ = {%}**

Algorithm or Rules to compute FIRST(X):

Examples: FIRST(X)

4. Rule 4 is applied for all productions whose right hand side gives **ϵ**.

Ex: To compute **FIRST(S)**, consider the production **S → ABC** and apply rule 3 as shown below:

d) S → ABC since **ABC => ϵ**, add **ϵ** to **FIRST(S).[Rule 4]**

i.e., **FIRST(S) = {ϵ}**

Algorithm or Rules to compute FIRST(X):

Examples: FIRST(X)

4. Rule 4 is applied for all productions whose right hand side gives **ϵ**.

Ex: So, All the above **FIRST** values are shown below:

	S	A	B	C
FIRST	%, *, +, ϵ	ϵ, +	ϵ, *	ϵ, %

Algorithm or Rules to compute FIRST(X):

Examples: FIRST(X)

5.Rule 5 is applied only for terminals.

Ex 1: **+** is terminal. So, **FIRST(+) = {+}**

Ex 2: **a** is terminal. So, **FIRST(a) = {a}**

Ex 3: **id** is terminal. So, **FIRST(id) = {id}**

Algorithm or Rules to compute FIRST(X):

Note: FIRST(X)

FIRST($X_1X_2X_3...X_n$) can be computed as follows:

1. **FIRST($X_1X_2X_3...X_n$) = FIRST(X_1) - {ϵ}**
2. If **FIRST(X_1) = ϵ**, then **FIRST($X_1X_2X_3...X_n$) = FIRST(X_2) - {ϵ}**
3. If **FIRST(X_1) = ϵ** and **FIRST(X_2) = ϵ** then **FIRST($X_1X_2X_3...X_n$) = FIRST(X_3) - {ϵ}**

.....

4. If **FIRST(X_1) = FIRST(X_2) = ...= FIRST(X_n) = ϵ** then **FIRST($X_1X_2X_3...X_n$) = {ϵ}**

Algorithm or Rules to compute FIRST(X):

Practice Problem: FIRST(X)

Compute **FIRST** values for the following grammar:

E → TE′

E′ → +TE′|ϵ

T → FT′

T′ → *FT′|ϵ

F → (E)|id

	E	E′	T	T′	F
FIRST	(, id	ϵ, +	(, id	ϵ, *	(, id

Computing FOLLOW Symbols

Definition: FOLLOW

The **FOLLOW(A)** for a non-terminal **A** is defined as the set of terminals '**a**' that will appear immediately to the right of **A** in some sentential form. That is, the set of terminals '**a**' such that there exists a derivation of the form:

S =>* αAaβ for some **α** and **β**.

If **A** is appeared as the last symbol in some sentential form, then place **$** in **FOLLOW(A)** where the symbol **$** is treated as "**end marker**" symbol.

Algorithm or Rules to compute FOLLOW(A):

Algorithm FOLLOW(A) :

Rule 1:

FOLLOW(S) $\leftarrow$ **$**, if **S** is the **Start Symbol**. i.e.,
FOLLOW(S) = {$}

Rule 2:

If **A** $\rightarrow$ **αBβ** is a production and **β** $\neq$ **ϵ** then
FOLLOW(B) $\leftarrow$ non - **ϵ** symbols in **FIRST(β)** i.e.,
FOLLOW(B) = FIRST(β) - {ϵ}.

Algorithm or Rules to compute FOLLOW(A):

Algorithm FOLLOW(A):

Rule 3:

If **A → αBβ** or **A → αB** is a production and **β =>*ϵ** (i.e. **FIRST(β)** contains **ϵ**) then **FOLLOW(B) ← FOLLOW(A)** i.e., **FOLLOW(B) = FOLLOW(A)**.

Computing FOLLOW Symbols

Example: FOLLOW(A)

Compute **FIRST** and **FOLLOW** sets for the following grammar:

E → TE′

E′ → +TE′|ϵ

T → FT′

T′ → *FT′|ϵ

F → (E)|id

Computing FOLLOW Symbols

Example: *Solution* FOLLOW(A)

FIRST values are shown below:

$E \rightarrow TE'$

$E' \rightarrow +TE' \mid \epsilon$

$T \rightarrow FT'$

$T' \rightarrow *FT' \mid \epsilon$

$F \rightarrow (E) \mid id$

	E	E′	T	T′	F
FIRST	(, id	+, ϵ	(, id	*, ϵ	(, id

Computing FOLLOW Symbols

Example: *Solution* FOLLOW(A)

Computing **FOLLOW** sets:

Rule 1: $ is placed in **FOLLOW(E)** since **E** is the start symbol.

i.e., **FOLLOW(E) = {$}**

$$E \rightarrow TE'$$

$$E' \rightarrow +TE' \mid \epsilon$$

$$T \rightarrow FT'$$

$$T' \rightarrow *FT' \mid \epsilon$$

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

Computing FOLLOW Symbols

Example: ***Solution*** **FOLLOW(A)**

Computing **FOLLOW** sets:

Rule 2 & 3: apply rule 2 & 3 for every production of the form **A → αBβ** where **B** is a non-terminal.

$E \rightarrow TE'$

$E' \rightarrow +TE' \mid \epsilon$

$T \rightarrow FT'$

$T' \rightarrow *FT' \mid \epsilon$

$F \rightarrow (E) \mid id$

Computing FOLLOW Symbols

Example: *Solution* FOLLOW(A)

Computing **FOLLOW** sets:

$A \rightarrow \alpha B \beta$ Rule 2 ($\beta \neq \epsilon$): FOLLOW (B) ← (=) FIRST(β) - {ϵ}	$A \rightarrow \alpha B \beta$ Rule 3 ($\beta =>^* \epsilon$): FOLLOW (B) ← (=) FOLLOW(A)
$E \rightarrow TE'$ $A \rightarrow \alpha B \beta$, here B is T & β is E' FOLLOW(T) = FIRST(E') - {ϵ} = {+}	$E \rightarrow TE'$ $A \rightarrow \alpha B \beta$, here B is T & β is E' and $E' =>^* \epsilon$ FOLLOW(T) = FOLLOW(E)
$E \rightarrow TE'$ $A \rightarrow \alpha B \beta$, here B is E' & β is ϵ Rule 2 not applicable.	$E \rightarrow TE'$ $A \rightarrow \alpha B \beta$, here B is E' & $\beta =>^* \epsilon$ FOLLOW(E') = FOLLOW(E)

Computing FOLLOW Symbols

Example: *Solution* FOLLOW(A)

Computing **FOLLOW** sets:

$A \rightarrow \alpha B \beta$ Rule 2 ($\beta \neq \epsilon$): FOLLOW (B) ← (=) FIRST(β) - {ϵ}	$A \rightarrow \alpha B \beta$ Rule 3 ($\beta =>^* \epsilon$): FOLLOW (B) ← (=) FOLLOW(A)
$E' \rightarrow +TE'$ $A \rightarrow \alpha B \beta$, here B is T & β is E' FOLLOW(T) = FIRST(E') - {ϵ} = {+}	$E' \rightarrow +TE'$ $A \rightarrow \alpha B \beta$, here B is T & β is E' and $E' =>^* \epsilon$ FOLLOW(T) = FOLLOW(E')
$E' \rightarrow +TE'$ $A \rightarrow \alpha B \beta$, here B is E' & β is ϵ Rule 2 not applicable.	$E' \rightarrow +TE'$ $A \rightarrow \alpha B \beta$, here B is E' & $\beta =>^* \epsilon$ FOLLOW(E') = FOLLOW(E')

Computing FOLLOW Symbols

Example: *Solution* FOLLOW(A)

Computing **FOLLOW** sets:

$A \rightarrow \alpha B \beta$ Rule 2 ($\beta \neq \epsilon$): FOLLOW (B) ← (=) FIRST(β) - {ϵ}	$A \rightarrow \alpha B \beta$ Rule 3 ($\beta =>^* \epsilon$): FOLLOW (B) ← (=) FOLLOW(A)
$T \rightarrow FT'$ $A \rightarrow \alpha B \beta$, here B is F & β is T' FOLLOW(F) = FIRST(T') - {ϵ} = {*}	$T \rightarrow FT'$ $A \rightarrow \alpha B \beta$, here B is F & β is T' and $T' =>^* \epsilon$ FOLLOW(F) = FOLLOW(T)
$T \rightarrow FT'$ $A \rightarrow \alpha B \beta$, here B is T' & β is ϵ Rule 2 not applicable.	$T \rightarrow FT'$ $A \rightarrow \alpha B \beta$, here B is T' & $\beta =>^* \epsilon$ FOLLOW(T') = FOLLOW(T)

Computing FOLLOW Symbols

Example: *Solution* FOLLOW(A)

Computing **FOLLOW** sets:

$A \rightarrow \alpha B \beta$ Rule 2 ($\beta \neq \epsilon$): FOLLOW (B) ← (=) FIRST(β) - {ϵ}	$A \rightarrow \alpha B \beta$ Rule 3 ($\beta =>^* \epsilon$): FOLLOW (B) ← (=) FOLLOW(A)
$T' \rightarrow *FT'$ $A \rightarrow \alpha B \beta$, here B is F & β is T' FOLLOW(F) = FIRST(T') - {ϵ} = {*}	$T' \rightarrow *FT'$ $A \rightarrow \alpha B \beta$, here B is F & β is T' and $T' =>^* \epsilon$ FOLLOW(F) = FOLLOW(T')
$T' \rightarrow *FT'$ $A \rightarrow \alpha B \beta$, here B is T' & β is ϵ Rule 2 not applicable.	$T' \rightarrow *FT'$ $A \rightarrow \alpha B \beta$, here B is T' & $\beta =>^* \epsilon$ FOLLOW(T') = FOLLOW(T')

Computing FOLLOW Symbols

Example: *Solution* FOLLOW(A)

Computing **FOLLOW** sets:

$A \rightarrow \alpha B \beta$ Rule 2 ($\beta \neq \epsilon$): FOLLOW (B) ← (=) FIRST(β)-{ϵ}	$A \rightarrow \alpha B \beta$ Rule 3 ($\beta =>^{*} \epsilon$): FOLLOW (B) ← (=) FOLLOW(A)
F → (E) A → αB β, here B is E & β is ')' FOLLOW(E) = FIRST())-{ϵ} = {)}	F → (E) A → αB β, here B is E & β is ')' and ≠ ϵ Rule 3 not applicable.

Computing FOLLOW Symbols

Example: *Solution* FOLLOW(A)

FOLLOW values are shown below:

$$E \rightarrow TE'$$
$$E' \rightarrow +TE' \mid \epsilon$$
$$T \rightarrow FT'$$
$$T' \rightarrow *FT' \mid \epsilon$$
$$F \rightarrow (E) \mid id$$

	E	E′	T	T′	F
FOLLOW	$,)	$,)	+, $,)	+, $,)	+, *, $,)

Computing FOLLOW Symbols

Practice Problem: FOLLOW(A)

Compute **FIRST** and **FOLLOW** sets for the following grammar:

S → iCtS | iCtSeS | a

C → b

Summary...

Top-Down Parsing : Predictive Parser (Part-2)

- FIRST
- FOLLOW
- Example Problems

Reading: Aho2, Section 4.4.2

Next Lecture: Predictive parser cont.