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-3)

Outline:

- Predictive Parsing Table Construction
- LL (1) Grammar
- Error Recovery in Predictive Parsing

Construction of Predictive Parsing Table

By using **FIRST** and **FOLLOW** sets, we can easily construct the **predictive parsing table** and the productions are entered into the table **M[A, a]** where

- **M** is a 2-dimensional array representing the **predictive parsing table**.
- **A** is a non-terminal or variable which represent the row values.
- **'a'** is a terminal or **$** (which is endmarker) and represent the column values.

Construction of Predictive Parsing Table

Algorithm:

`Construction of a predictive parsing table.`

Input: Grammar **`G`**.

Output: Predictive Parsing table **`M`**.

Construction of Predictive Parsing Table

Procedure:

For each production **A → α** of the grammar **G** apply the following rules:

Rule 1:

For each terminal '**a**' in **FIRST(α)** , add **A → α** to **M[A, a]**.

Rule 2:

If **FIRST(α)** contains **ϵ**, for each terminal '**b**' in **FOLLOW(A)**, add **A → α** to **M[A, b]**.

If **ϵ** is in **FIRST(α)** and **$** in **FOLLOW(A)**, add **A → α** to **M[A, $]** as well.

Construction of Predictive Parsing Table

Example:

Construct the predictive parsing table for the following grammar

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

Construction of Predictive Parsing Table

Example: *Solution*

The **FIRST** and **FOLLOW** sets of each non-terminal of the given grammar are shown below:

	E	E′	T	T′	F
FIRST	(, id	+, ϵ	(, id	*, ϵ	(, id
FOLLOW	), $	), $	+,), $	+,), $	+,*,), $

Construction of Predictive Parsing Table

Example: *Solution*

For every production of the form **A → α**, we compute **FIRST(α)** and entries of the predictive parsing table can be done as shown below:

Productions A → α	a = FIRST(α)	M[A, a] = A → α	Rule
E → TE′	(, id	M[E, (] = E → TE′ M[E, id] = E → TE′	1
E′ → +TE′	+	M[E′, +] = E′ → +TE′	1
E′→ ϵ	ϵ	FOLLOW(E′) = {), $} M[E′,)] = E′→ ϵ M[E′, $] = E′→ ϵ	2

Construction of Predictive Parsing Table

Example: *Solution*

Productions A → α	a = FIRST(α)	M[A, a] = A → α	Rule
T → FT′	(, id	M[T, (] = T → FT′ M[T, id] = T → FT′	1
T′ → *FT′	*	M[T′, *] = T′ → *FT′	1
T′→ ϵ	ϵ	FOLLOW(T′) = {+,), $} M[T′, +] = T′→ ϵ M[T′,)] = T′→ ϵ M[T′, $] = T′→ ϵ	2

Construction of Predictive Parsing Table

Example: *Solution*

Productions A → α	a = FIRST(α)	M[A, a] = A → α	Rule
F → (E)	(	M[F, (] = F → (E)	1
F → id	id	M[F, id] = F → id	1

Construction of Predictive Parsing Table

Example: *Solution*

The **Predictive Parsing Table** is show below:

	id	+	*	(	)	$
E	E → TE′			E → TE′		
E′		E′→ +TE′			E′→ϵ	E′→ϵ
T	T → FT′			T → FT′		
T′		T′→ϵ	T′ → *FT′		T′→ϵ	T′→ϵ
F	F → id			F → (E)		

Construction of Predictive Parsing Table

Practice Problem:

Construct the predictive parsing table for the following grammar

1. S → iCtS | iCtSeS | a
2. C→ a

LL(1) Grammars

Definition:

The **grammar** from which a **predictive parser**, that is **recursive-descent parser without backtracking** is constructed is called **LL(1) grammar**.

where

- The *first* **L** stands for left-to-right scan of the input.
- The *second* **L** stands for leftmost derivation. So, the predictive parsers always mimic (derivatives or imitate) the leftmost derivation.
- The *digit* **1** indicates one input symbol of lookahead at each step to make parsing action decisions.

LL(1) Grammars

- In predictive parsing table if there are no multiple entries, then the given grammar is called **LL(1)** grammar. If multiple entries are present in the parsing table, then the grammar is not **LL(1)**. The predictive parser accepts only the language generated from **LL(1)** grammar.
- In **LL(1)** parsing technique or predictive parsing if two or, more alternative productions are there, the predictive parser also called **LL(1)** parser chooses the correct production by guessing using one lookahead token.

LL(1) Grammars

The following grammars are not LL(1) grammars:

- **Ambiguous grammar** is **not LL(1)**
- **Left recursive grammar** is **not LL(1)**
- The grammar which is **not left factored** (that is, if two or more alternative productions have common prefix), the grammar is **not LL(1)**
- The grammar that results in **multiple entries in the parsing table** is **not LL(1)**

Construction of Predictive Parsing Table

Example Problem:

Construct the predictive parsing table for the following grammar:

1. S → iCtS | iCtSeS | a

2. C→ b

Is the grammar **LL(1)**?

Construction of Predictive Parsing Table

Example Problem: *Solution*

Given grammar:

1. S → iCtS | iCtSeS | a

2. C→ b

The grammar is **not left factored**, so eliminate using **left factoring**.

The resultant grammar after applying **left factoring** is given below:

1. S → iCtSS′| a

2. S′ → ϵ| eS

3. C→ b

Construction of Predictive Parsing Table

Example Problem: *Solution*

1. S → iCtSS′| a
2. S′ → ϵ| eS
3. C→ b

The **FIRST** and **FOLLOW** sets are given below:

	S	S′	C
FIRST	i, a	e, ϵ	b
FOLLOW	e, $	e, $	t

Construction of Predictive Parsing Table

Example Problem: *Solution*

1. S → iCtSS′| a
2. S′ → ϵ| eS
3. C→ b

The **Predictive Parsing Table** is show below:

	i	a	b	e	t	$
S	S → iCtSS′	S → a				
S′		E′→ +TE′		S′ → eS S′ → ϵ		S′ → ϵ
C			C → b			

Construction of Predictive Parsing Table

Example Problem: *Solution*

The **Predictive Parsing Table** is show below:

	i	a	b	e	t	$
S	S → iCtSS′	S → a				
S′		E′→ +TE′		S′ → eS S′ → ϵ		S′ → ϵ
C			C → b			

The above **Predictive Parsing Table M[S′, e]** contains two entries. So the given grammar is **not LL(1)** grammar.

Construction of Predictive Parsing Table

Practice Problem-1:

Test whether the grammar is LL(1) or not and construct a predictive parsing table for it? [*JNTUH: March/April-2021*]

1. **S → AaAb | BaBa|a|b**
2. **A → ϵ**
3. **B → ϵ**

Construction of Predictive Parsing Table

Practice Problem-2:

Verify whether the following grammar is LL(1) or not?
[*JNTUH: May-2019*]

1. **E → E + T | T**
2. **T → T * F | F**
3. **F → (E) | a | b**

Error Recovery in Predictive Parsing

An error is detected during predictive parsing when the following two situations occur:

- The terminal on top of the stack does not match with the next input symbol.
- When non-terminal **`A`** is on top of the stack, '**`a`**' is the next input symbol and **`M[A, a]`** has blank entry (blank denote an error).

Error Recovery in Predictive Parsing

The error recovery is done using *panic mode* and *phrase-level recovery* as shown below:

`Panic Mode Recovery:`

In this approach, error recovery is done by skipping symbols from the input until a token matches with synchronizing tokens. The synchronizing tokens are selected such that the parser should quickly recover from the errors that are likely to occur in practice. Some of the recovery techniques are shown below:

Error Recovery in Predictive Parsing

Panic Mode Recovery:

Some of the recovery techniques are shown below:

1. For a non-terminal **A**, consider the symbols in **FOLLOW (A)**. These symbols can be considered as synchronizing tokens and are added into parsing table replacing only blank entries. Now, whenever there is a mismatch, keep skipping the tokens till we get one of the synchronizing character and remove **A** from the stack. It is likely that parsing can continue.

Error Recovery in Predictive Parsing

Panic Mode Recovery:

2. For a non-terminal **A**, consider the symbols in **FIRST (A)**. These symbols can be considered as synchronizing characters and add to the parsing table replacing only blank entries. Now, whenever there is a mismatch, keep skipping the tokens till we get one of the synchronizing character and remove **A** from the stack. It is also likely that parsing can continue.

Error Recovery in Predictive Parsing

Panic Mode Recovery:

3. If a terminal on top of the stack cannot be matched, pop the terminal from the stack and issue "Error Message" and insert the corresponding terminal and continue parsing.

Error Recovery in Predictive Parsing

Phrase Level Recovery:

1. This recovery method is implemented by filling the blank entries in the predictive parsing table with pointers to error routines.
2. These routines may change, insert, replace or delete symbols from the input and issue appropriate error messages.
3. They may also pop from the stack.

Error Recovery in Predictive Parsing

Example:

Consider the following grammar:

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

And the predictive parsing table is shown below:

Error Recovery in Predictive Parsing

Example:

The Predictive Parsing Table is shown below:

	id	+	*	(	)	$
E	E → TE′			E → TE′		
E′		E′→ +TE′			E′→ϵ	E′→ϵ
T	T → FT′			T → FT′		
T′		T′→ϵ	T′ → *FT′		T′→ϵ	T′→ϵ
F	F → id			F → (E)		

Add the synchronizing tokens for the above parsing table and show the sequence of moves made by the parser for the string "**)id*+id**"

Error Recovery in Predictive Parsing

Example: *Solution*

- The synchronizing characters are the characters present in FIRST or FOLLOW sets of each non-terminal.
- In this example, let us add synchronizing characters by considering FOLLOW of each non-terminal replacing each blank entry in the parsing table.

Error Recovery in Predictive Parsing

Example: *Solution*

- The FOLLOW sets of each non-terminal are shown below:

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

Now, **FOLLOW (E) = {$,)}**. So, **M[E,$] = M[E,)] = synch** for blank entries.

Similarly, **FOLLOW (F) = {+, *, $,)}**. So, **M[F,+] = M[F, *] = M[F, $] = M[F,)] = synch**.

And **FOLLOW (T) = {+, $,)}**. So, **M[T,+] = M[T, $] = M[T,)] = synch**.

Error Recovery in Predictive Parsing

Example: *Solution*

By adding synchronizing characters to the parsing table as shown below:

	id	+	*	(	)	$
E	E → TE′			E → TE′	synch	synch
E′		E′→ +TE′			E′→ϵ	E′→ϵ
T	T → FT′	synch		T → FT′	synch	synch
T′		T′→ϵ	T′ → *FT′		T′→ϵ	T′→ϵ
F	F → id	synch	synch	F → (E)	synch	synch

Error Recovery in Predictive Parsing

Example: *Solution*

Now, the sequence of moves made by the parser for the string "`)id*+id`" is shown below:

Stack (X)	Input (a)	Output (M[A, a])	Action
$E	)id*+id$	M[E,)] = Error, skip	Remove) from the input.
$E	id*+id$	E → TE′	Pop E and Push TE′ in reverse.

Error Recovery in Predictive Parsing

Example: *Solution*

Now, the sequence of moves made by the parser for the string **")id*+id"** is shown below:

Stack (X)	Input (a)	Output (M[A, a])	Action
$E′T	id*+id$	T → FT′	Pop T and Push FT′ in reverse.
$E′T′F	id*+id$	F → id	Pop F and Push id.
$E′T′id	id*+id$	Match id	Pop id and increment input pointer.

Error Recovery in Predictive Parsing

Example: *Solution*

Now, the sequence of moves made by the parser for the string **")id*+id"** is shown below:

Stack (X)	Input (a)	Output (M[A, a])	Action
$E′T′	*+id$	T → *FT′	Pop T′ and Push *FT′ in reverse.
$E′T′F*	*+id$	Match *	Pop * and increment input pointer.
$E′T′F	+id$	Error, skip	Pop + from the input.

Error Recovery in Predictive Parsing

Example: *Solution*

Now, the sequence of moves made by the parser for the string **")id*+id"** is shown below:

Stack (X)	Input (a)	Output (M[A, a])	Action
$E'T'F	id$	F → id	Pop F and Push id.
$E'T'id	id$	Match id	Pop id and increment input pointer.
$E'T'	$	T' → ϵ	Remove T' from the stack.

Error Recovery in Predictive Parsing

Example: *Solution*

Now, the sequence of moves made by the parser for the string "**)id*+id**" is shown below:

Stack (X)	Input (a)	Output (M[A, a])	Action
$E′	$	E′ → ϵ	Remove E′ from the stack.
$	$	ACCEPT	

Note: Observe that parsing is successful and the parser has also recognized two errors. By looking at these errors if the programmer corrects the program, parsing action is successful without any errors.

Summary...

Top-Down Parsing : Predictive Parser (Part-3)

- Predictive Parsing Table Construction
- LL (1) Grammar
- Error Recovery in Predictive Parsing

Reading: Aho2, Section 4.4.3 & 4.4.5

Next Lecture: Bottom-Up Parsing