Predictive Parser

Dr. Meera Thapar Khanna

CSE Department

Pandit Deendayal Energy University

LL (1) Parsing or Predictive Parsing

LL (1) stands for, left to right scan of input, uses a Left most derivation, and the parser takes 1 symbol as the look ahead symbol from the input in taking parsing action decision.

A non recursive predictive parser can be built by maintaining a stack explicitly, rather than implicitly via recursive calls.

Parsing Table

- It is the process of placing all the productions of the grammar in the parse table based on the FIRST and FOLLOW values of the productions.

Rules to construct the Parsing Table (M):

For Each production A-> α of the grammar, do the bellow steps:

1. For each terminal symbol ‘a’ in FIRST (α), add the production A-> α to M [A, a].
2. If ε is in FIRST(α) add production A->α to M [A, b], where b is all terminals in FOLLOW (A).
3. If ε is in FIRST(α) and $ is in FOLLOW(A) then add production A->α to M [A, $].
4. Mark other entries in the parsing table as error.

In the parsing table, rows will contain the Non-Terminals and the column will contain the Terminal Symbols. All the Null Productions of the Grammars will go under the Follow elements and the remaining productions will lie under the elements of the First set.

Example

Grammar

1. E -> TE’
2. E’ -> +TE’ | ε
3. T -> FT’
4. T’ -> *FT’ | ε
5. F -> (E) | id

First and Follow

NON TERMINAL	FIRST	FOLLOW
E	{ (, id }	{ $,) }
E’	{ +, ε }	{ $,) }
T	{ (, id}	{ +, $,) }
T’	{*, ε}	{ +, $,) }
F	{ (, id}	{ *, +, $,) }

Parsing Table Entries

For Production E -> TE’:

First(TE’) = First(T) = {(, id}

So this production is added to M[E, (] and M[E, id].

For Production E’ -> +TE’

First(+TE’) = {+}

So this production is added to M[E’, +].

According to 3rd rule E’ -> ε is added to M[E’,)] and M[E’, $]

Parsing Table M

Non-Terminal	Input Symbol					
	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)		

If there are no multiple entries in the table for single a terminal then grammar is accepted by LL(1) Parser.

LL(1) Parsing Algorithm

The parser acts on basis on the basis of two symbols

- A, the symbol on the top of the stack
- a, the current input symbol

There are three conditions for A and ‘a’, that are used fro the parsing program.

- If A=a=$ then parsing is Successful.
- If A=a≠$ then parser pops off the stack and advances the current input pointer to the next.
- If A is a Non terminal the parser consults the entry M [A, a] in the parsing table. If M[A, a] is a Production A-> X1X2..Xn, then the program replaces the A on the top of the Stack by X1X2..Xn in such a way that X1 comes on the top.

Example:
id+id*id

Stack	Input	Action	Comment
$E	id+id*id $	E -> TE’	E on top of the stack is replaced by TE’
$E’T	id+id*id $	T -> FT’	T on top of the stack is replaced by FT’
$E’T’F	id+id*id $	F -> id	F on top of the stack is replaced by id
$E’T’id	id+id*id $	pop and remove id	Condition 2 is satisfied
$E’T’	+id*id $	T’ -> ε	T’ on top of the stack is replaced by ε
$E’	+id*id $	E’ -> +TE’	E’ on top of the stack is replaced by +TE’
$E’T+	+id*id $	pop and remove +	Condition 2 is satisfied
$E’T	id*id $	T -> FT’	T on top of the stack is replaced by FT’
$E’T’F	id*id $	F -> id	F on top of the stack is replaced by id
$E’T’id	id*id$	pop and remove id	Condition 2 is satisfied
$E’T’	*id $	T’ -> FT’	T’ on top of the stack is replaced by *FT’
$E’T’F*	*id $	pop and remove *	Condition 2 is satisfied
$E’T’F	id $	F -> id	F on top of the stack is replaced by id
$E’T’id	id $	pop and remove id	Condition 2 is satisfied
$E’T’	$	T’ -> ε	T’ on top of the stack is replaced by ε
$E’	$	E’ -> ε	E’ on top of the stack is replaced by ε
$	$	Parsing is successful	Condition 1 is satisfied

Parse Tree