Asansol Engineering College

Department of Information technology

Topic: **LL1 Parser Algorithm**

Name : Priyanka Kumari

University Roll No: 10800221133

Subject : Compiler Design

Contents

Introduction:

LL(1) Parsing:

Here the 1st **L** represents that the scanning of the Input will be done from Left to Right manner and the second **L** shows that in this parsing technique we are going to use Left most Derivation Tree. And finally, the **1** represents the number of look-ahead, which means how many symbols are you going to see when you want to make a decision.

Essential conditions to check first are as follows:

1. The grammar is free from left recursion.
2. The grammar should not be ambiguous.
3. The grammar has to be left factored in so that the grammar is deterministic grammar.

Algorithm:

Step 1**:** First check all the essential conditions mentioned above and go to step 2.

Step 2**:** Calculate First() and Follow() for all non-terminals.

1**.<u>First()</u>:** If there is a variable, and from that variable, if we try to drive all the strings then the beginning Terminal Symbol is called the First.

2.**<u>Follow()</u>:** What is the Terminal Symbol which follows a variable in the process of derivation.

Step 3**:** For each production A –> α. (A tends to alpha)

1.Find First(α) and for each terminal in First(α), make entry A –> α in the table.

2.If First(α) contains ε (epsilon) as terminal than, find the Follow(A) and for each terminal in Follow(A), make entry A –> α in the table.

3.If the First(α) contains ε and Follow(A) contains $ as terminal, then make entry A –> α in the table for the $.

A figure of LL1 Parser:

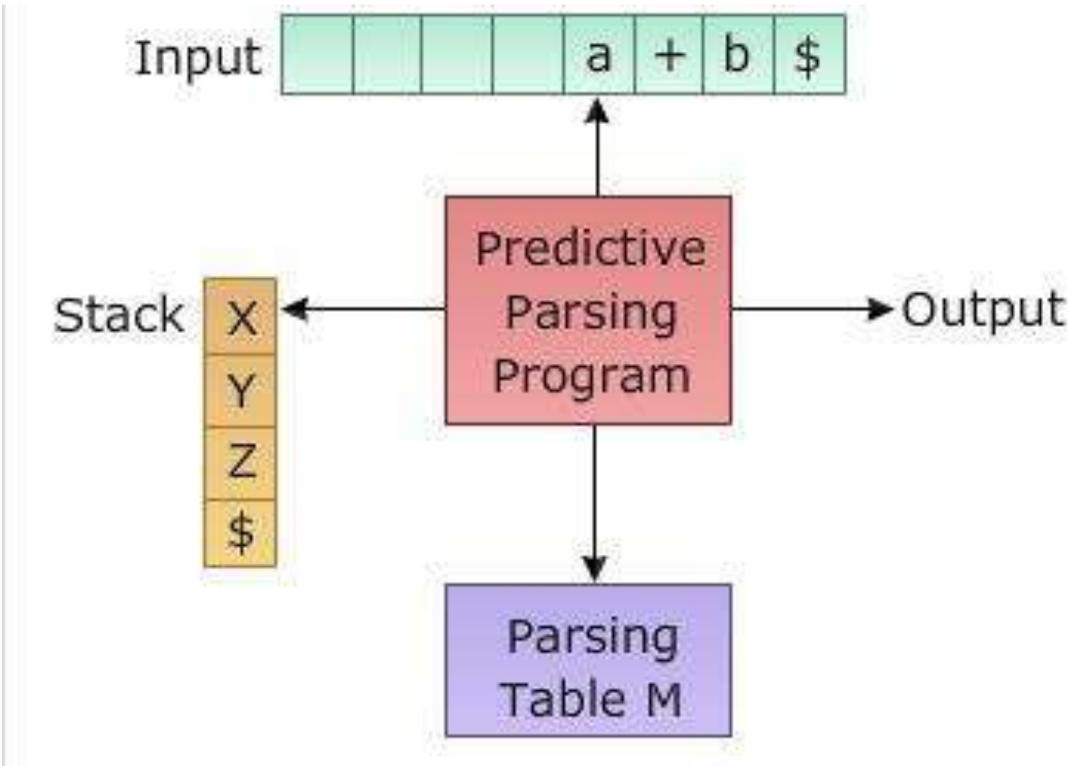

Table:

LL(1) Parsing Table Example

Left-factored grammar

E → T X
X → + E | ε
T → (E) | int Y
Y → * T | ε

End of input symbol

The LL(1) parsing table

	int	*	+	(	)	$
E	T X			T X		
X			+ E		ε	ε
T	int Y			(E)		
Y		* T	ε		ε	ε

Implementation:

Consider the Grammar:

E --> TE'

E' --> +TE' | ε

T --> FT'

T' --> *FT' | ε

F --> id | (E)

*ε denotes epsilon

Step1 – The grammar satisfies all properties in step 1
Step 2 – calculating first() and follow()

Find their First and Follow sets:

	First	Follow
E –> TE’	{ id, (}	{ $,) }
E’ –> +TE’/ε	{ +, ε }	{ $,) }
T –> FT’	{ id, (}	{ +, $,) }
T’ –> *FT’/ε	{ *, ε }	{ +, $,) }
F –> id/(E)	{ id, (}	{ *, +, $,) }

Step 3 – making parser table

Now, the LL(1) Parsing Table is:

	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)		

This grammar is LL1.

So, the parse tree can be derived from the stack implementation of the given parsing table.

Conclusion:

From this report , we came to know about the about introduction , algorithm ,example of LL1 Parser ,how to implement the algorithm .

Most important it's applications and applications .

References:

Book reference:

1.. *Rosenkrantz, D. J.; Stearns, R. E. (1970). "Properties of Deterministic Top Down Grammars". Information and Control.* ***17*** *(3): 226–256. doi:10.1016/s0019-9958(70)90446-8.*

Site reference:

1.*https://en.wikipedia.org/wiki/LL_parser*

2.*https://www.geeksforgeeks.org/construction-of-ll1-parsing-table/*