

**UNIT 5: TOP-DOWN PARSING TECHNIQUES**

We want to develop a translator for a language of arithmetic expressions to code C, Pascal or Java. The characteristics of the language are:

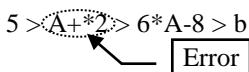
- Operands are variables and integer numeric constants.
- The arithmetic operators are: +, -, \*, /
- All operations are integer.
- There is no precedence between the different arithmetic operators. Expressions are evaluated from left to right.
- The operator > loads the result of the arithmetic operation to the left of the operator in the variable to the right of it. To perform this "assignment" operation on the right only an identifier can appear as an expression, otherwise the result of evaluating the expression is lost.
- The iteration statement indicates the expressions that it affects with the operators [], the number of iterations indicating the value of the arithmetic expression enclosed in parentheses that will appear after the symbol ].
- When the variables are first used, they are initialized with a value of 0.

Example:

3>A>[2\*A-1+b>b](A)

The operations that are performed are: value 3 is assigned to variable A, the operation 2 \* A-1 + b is performed and the result, 5, is assigned to variable b, this last operation is repeated 5 times, therefore, after running the entire line, variable b will take the value 15.

An erroneous expression could be:

5 > A+\*2 > 6\*A-8 > b  


It is required

1. Define the grammar G for the translator.
2. Construct the LL(1) analysis table for the translator.

**Solution:**

1. The tokens of the proposed grammar in this solution are:  $\{[, ], (, ), >, op, id, num\}$ . The token "op" represents any operation (+, -, \*, /), can be treated as a grammatical symbol because they have the same precedence. The token "id" represents variables and "num" represents numeric constants (it could also return the lexical analyzer a single token for variables and constants). The production rules of the grammar are:

$$\begin{aligned}
 S &\rightarrow E > S \mid E \\
 E &\rightarrow [S] (E) \mid P \text{ op } E \mid P \\
 P &\rightarrow id \mid num
 \end{aligned}$$

The axiom, S, constructs the line of expressions joined with the > operator and the iterations. The non-terminal "E" constructs the arithmetic expressions, set of operands joined by an operator

After factoring the grammar:

$$\begin{aligned}
 S &\rightarrow E S' \\
 S' &\rightarrow > S \mid \lambda \\
 E &\rightarrow [S] (E) \mid P E' \\
 E' &\rightarrow op E \mid \lambda \\
 P &\rightarrow id \mid num
 \end{aligned}$$

2.

$\Sigma_N$	First			Follow			
<b>E</b>	Id	num	[	)	>	]	\$
<b>E'</b>	Op	$\lambda$		)	>	]	\$
<b>P</b>	Id	num		op	)	>	]
<b>S</b>	Id	num	[	]	\$		
<b>S'</b>	>	$\lambda$		]	\$		

Table LL(1)		$\Sigma_T$								
		\$	>	(	)	[	]	id	num	op
$\Sigma_N$	<b>E</b>	E → [S] (E)						E → P E'	E → P E'	
	<b>E'</b>	E' → $\lambda$	E' → $\lambda$	E' → $\lambda$		E' → $\lambda$				E' → op E
	<b>P</b>						P → id	P → num		
	<b>S</b>					S → E S'		S → E S'	S → E S'	
	<b>S'</b>	S' → $\lambda$	S' → > S				S' → $\lambda$			

