Syntax Analysis

Sudakshina Dutta

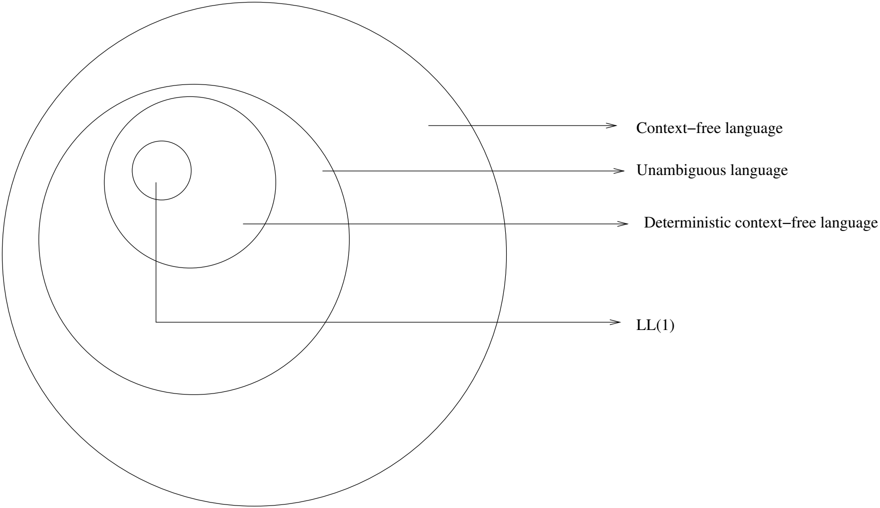
IIT Goa

18^{th} February, 2022

LL(1) grammar

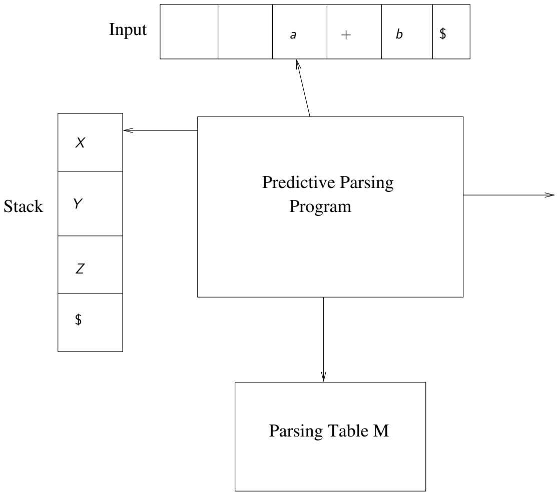
- A grammar G is LL(1) if and only if whenever $A \to \alpha|\beta$ are two productions from G and the following conditions hold:
  - For no terminal *a* do both α and β derives strings beginning with *a*
  - At most one of α and β can derive empty string
  - If β in one or more derivations lead to ϵ, then α does not derive any string beginning with a terminal in FOLLOW(A). Similarly for α

LL(1) grammar



Context−free language
Unambiguous language
Deterministic context−free language
LL(1)

Predictive parsing framework



Algorithm to construct Predictive Parsing Table

- **Input** : Grammar G
- **Output** : Parsing table M

Method

For each production $A \rightarrow \alpha$ of the grammar, do the following:

- For each terminal a in FIRST(α), add $A \rightarrow \alpha$ to $M[A, a]$
- If ϵ is in FIRST(α), then for each terminal b in FOLLOW(A), add $A \rightarrow \alpha$ to $M[A, b]$. If ϵ is in FIRST(α) and \$ is in FOLLOW(A), add $A \rightarrow \alpha$ to $M[A, \$]$ as well.

If no cell of this table has repeated entries, Then we can say that grammar is LL(1). This can be shown equivalent to previously stated conditions of LL(1) grammar.

Algorithm for predictive parsing

- Initially the parser is in a configuration with $w\$$ in the input buffer and the start symbol S of G on the top of the stack, above $\$$

Algorithm for predictive parsing

- let a be the first symbol of w
- let X be the top stack symbol
- while($X \neq \$$){
  - if ($X = a$) pop the stack and let a be the next symbol of w;
  - else if (X is a terminal) error();
  - else if ($M[X, a]$ is an error entry) error();
  - else if($M[X, a] = X \rightarrow Y_1 Y_2 \cdots Y_k$){
    output the production $X \rightarrow Y_1 Y_2 \cdots Y_k$;
    Pop the stack;
    Push Y_k, Y_{k-1}, $\cdots$, Y_1 onto the stack, with Y_1 on top;
    }
  - let X be the top stack symbol;

  }

Algorithm for predictive parsing

Where is the Input G of the below table?

Non–terminal	Input symbol			
	id	$+$	$*$	$\$$
E	$E \to TE'$			
E'		$E' \to +TE'$		$E' \to \epsilon$
T	$T \to FT'$			
T'		$T' \to \epsilon$	$T' \to *FT'$	$T' \to \epsilon$
F	$F \to id$			

Algorithm for predictive parsing

- The algorithm starts with $E\$$ in the stack
- If the input string is $+id$, an error is shown
  - $M[E, +] = error$

Note that all the empty entries in the table are treated as Errors.

- On input $id + id * id$, the non-recursive predictive parser makes the following sequence of moves

Table (T, id) result comes in next row.

MATCHED	STACK	INPUT	ACTION
	$E\$$	$id + id * id\$$	
	$TE'\$$	$id + id * id\$$	output $E \to TE'$
	$FT'E'\$$	$id + id * id\$$	output $T \to FT'$
	$id\ T'E'\$$	$id + id * id\$$	output $F \to id$
id	$T'E'\$$	$+id * id\$$	match id
id	$E'\$$	$+id * id\$$	output $T' \to \epsilon$
id	$+TE'\$$	$+id * id\$$	output $E' \to +TE'$
id+	$TE'\$$	$id * id\$$	match $+$
id+	$FT'E'\$$	$id * id\$$	output $T \to FT'$
id+	$idT'E'\$$	$id * id\$$	output $F \to id$
id+id	$T'E'\$$	$*id\$$	match id
id+id	$*FT'E'\$$	$*id\$$	output $T' \to *FT'$
id+id*	$FT'E'\$$	$id\$$	match $*$
id+id*	$id\ T'E'\$$	$id\$$	output $F \to id$
id+id*id	$T'E'\$$	$\$$	match id
id+id*id	$E'\$$	$\$$	output $T' \to \epsilon$
id+id*id	$\$$	$\$$	output $E' \to \epsilon$

- **Example**

  Consider the following grammar

$$S \rightarrow AaBb \; A \rightarrow c|\epsilon \; B \rightarrow d|\epsilon$$

- $FIRST(S) = \{c, a\}$, $FIRST(A) = \{c, \epsilon\}$, $FIRST(B) = \{d, \epsilon\}$
- $FOLLOW(S) = \{\$\}$, $FOLLOW(A) = \{a\}$, $FOLLOW(B) = \{b\}$
- Consider the input string to be "*cab*"

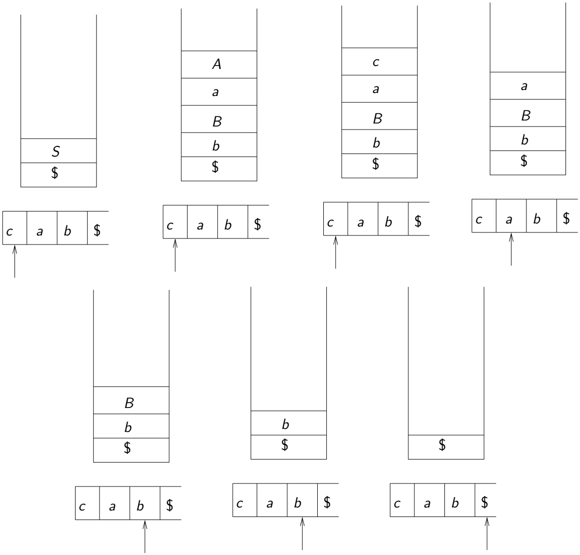
Note that before starting this Algo, we do (1st two steps are optional.)
- Left factoring
- Left Recursion elimination
- Construct below Parsing Table to see if it is LL(1)
- Then apply the actual algorithm.

Done!

NON–TERMINAL	INPUT SYMBOL				
	a	b	c	d	\$
S	$S \rightarrow AaBb$		$S \rightarrow AaBb$		
A	$A \rightarrow \epsilon$		$A \rightarrow c$		
B		$B \rightarrow \epsilon$		$B \rightarrow d$	

It is LL(1) grammar.

Done!



Note that when we have "Eps" to be pushed on to stack, We can just ignore it. No need to push it. In the Last but one figure, This can be observed.