SYNTAX ANALYSIS

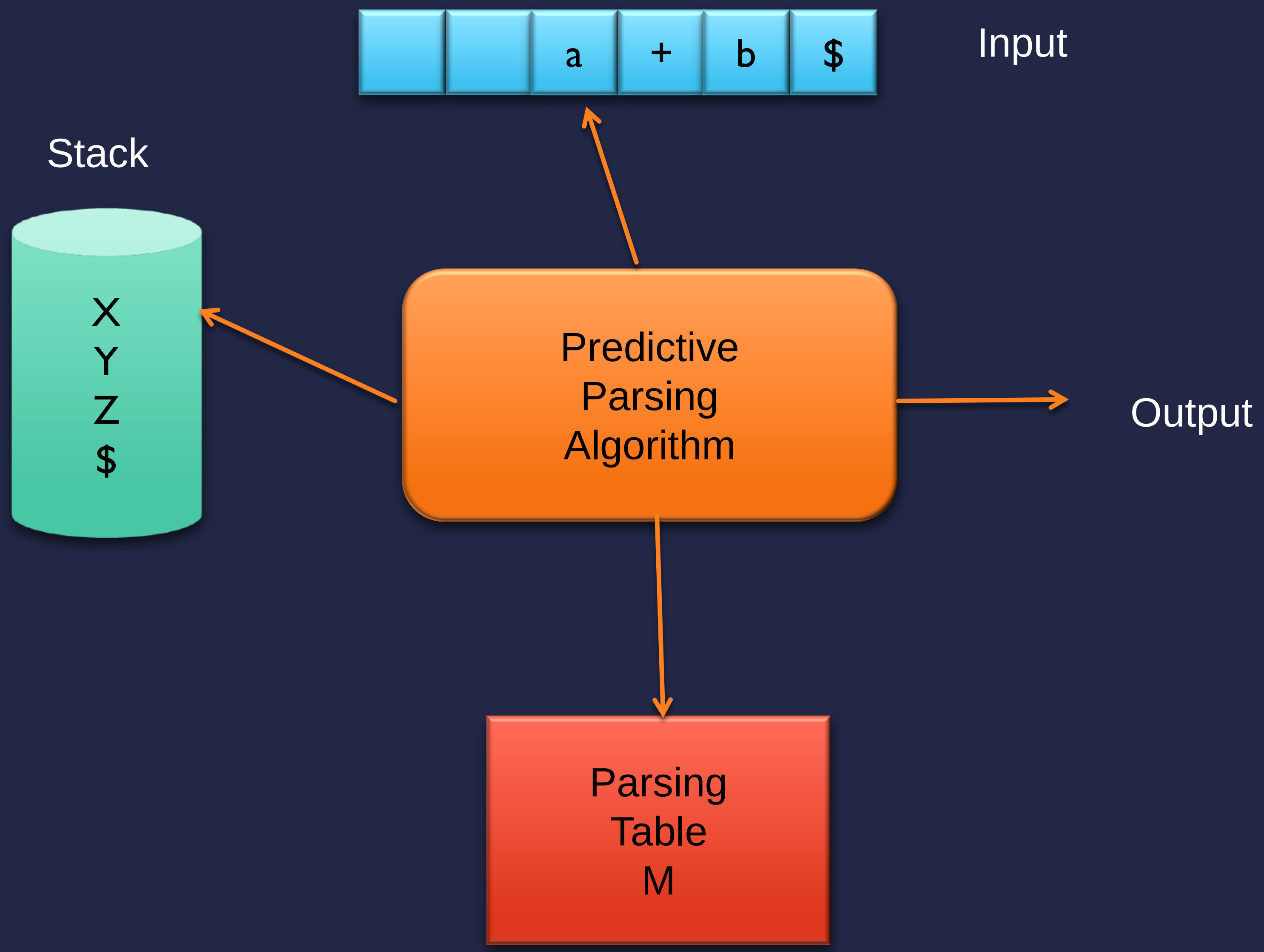
CONTENT

- LL (1) Parser
- Model of Non-Recursive Predictive Parser
- Construction of Predictive Parser Table
- Parsing a string

LL (1) GRAMMAR

- Used to construct Predictive Parser
- Predictive Parser – Recursive Descent Parser with no need of Backtracking
- First 'L' - scanning input from Left to Right
- Second 'L' - Leftmost Derivation
- "1" - One input symbol of Look ahead at each step to make parsing action decisions
- Left Recursive and Ambiguous grammar is NOT LL(1)

Model Of Non-Recursive Predictive Parser



Example:

E → TE'

E' → +TE' | ε

T → FT'

T' → *FT' | ε

F → (E) | **id**

	id	**+**	*****	**(**	**)**	**$**
E						
E'						
T						
T'						
F						

Construction Of Predictive Parsing Table

Input: Grammar G
Output: Parsing Table M

Table M has Non-Terminals as row and Terminals as columns

For Each Production $A \rightarrow \alpha$ of grammar do step 1 and 2

Step 1: For each terminal 'a' in FIRST (α)

Add $A \rightarrow \alpha$ to M [A , a]

Step 2:

Case 1:

If ε is in FIRST (α) then

for each terminal b in FOLLOW (A)

Add $A \rightarrow \alpha$ to M [A , b]

Case 2:

If ε is in FIRST (α) and \$ is in FOLLOW (A) then

for each terminal b in FOLLOW (A)

Add $A \rightarrow \alpha$ to M [A , b]

Step 3: Make each undefined entry of M be an error

Example:

E → TE'

E' → +TE' | ε

T → FT'

T' → *FT' | ε

F → (E) | **id**

	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)		

Consider Production F → id

FIRST (id) = { id }

Add F → id to M [F, id]

Construction Of Predictive Parsing Table

- For every LL grammar each parsing table entry uniquely identifies a production or signals an error
- For some grammars however M may have some entries that are multiply defined
- Such grammars are not LL(1) Grammar

Predictive Parsing Algorithm

```
Let a be the first symbol of w
Let X be the top of the Stack symbol
while ( X != $)
{
    if ( X == a)
        pop the stack and let 'a' be the next symbol of w
    else if ( X is a terminal )                         // X != a and X is terminal
        Error ( )
    else if ( M [ X , a ] is an error entry )
        Error ( )
    else if ( M [ X , a ] = Y1 Y2 … Yk )
        Output the production X → Y1 Y2 … Yk
        Pop the stack
        Push Yk Yk-1 … Y1 onto the stack with Y1 on top
   Let X be the top stack symbol
}
```

Predictive Parsing Algorithm

	id	+	*	(	)	$
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \varepsilon$	$E' \rightarrow \varepsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \varepsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \varepsilon$	$T' \rightarrow \varepsilon$
F	$F \rightarrow id$			$F \rightarrow (E)$		

Matched	Stack	Input	Action
	E $	**id + id * id $**	
	TE' $	**id + id * id $**	**Output E** $\rightarrow$ TE'
	FT'E' $	id + id * id $	Output T$\rightarrow$ FT'
	idT'E' $	id + id * id $	Output F $\rightarrow$ id
id	T'E' $	+ id * id $	match id
id	E' $	+ id * id $	Output T' $\rightarrow \varepsilon$
id	+TE' $	+ id * id $	Output E' $\rightarrow$ +TE'

Predictive Parsing Algorithm

	id	+	*	(	)	$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \varepsilon$	$E' \to \varepsilon$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \varepsilon$	$T' \to *FT'$		$T' \to \varepsilon$	$T' \to \varepsilon$
F	$F \to id$			$F \to (E)$		

Matched	Stack	Input	Action
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 *	idT'E' $	id $	Output $F \to id$

Predictive Parsing Algorithm

	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)		

Matched	Stack	Input	Action
id + id * id	T'E' $	$	match id
id + id * id	E' $	$	Output T' → ε
id + id * id	$	$	Output E' → ε