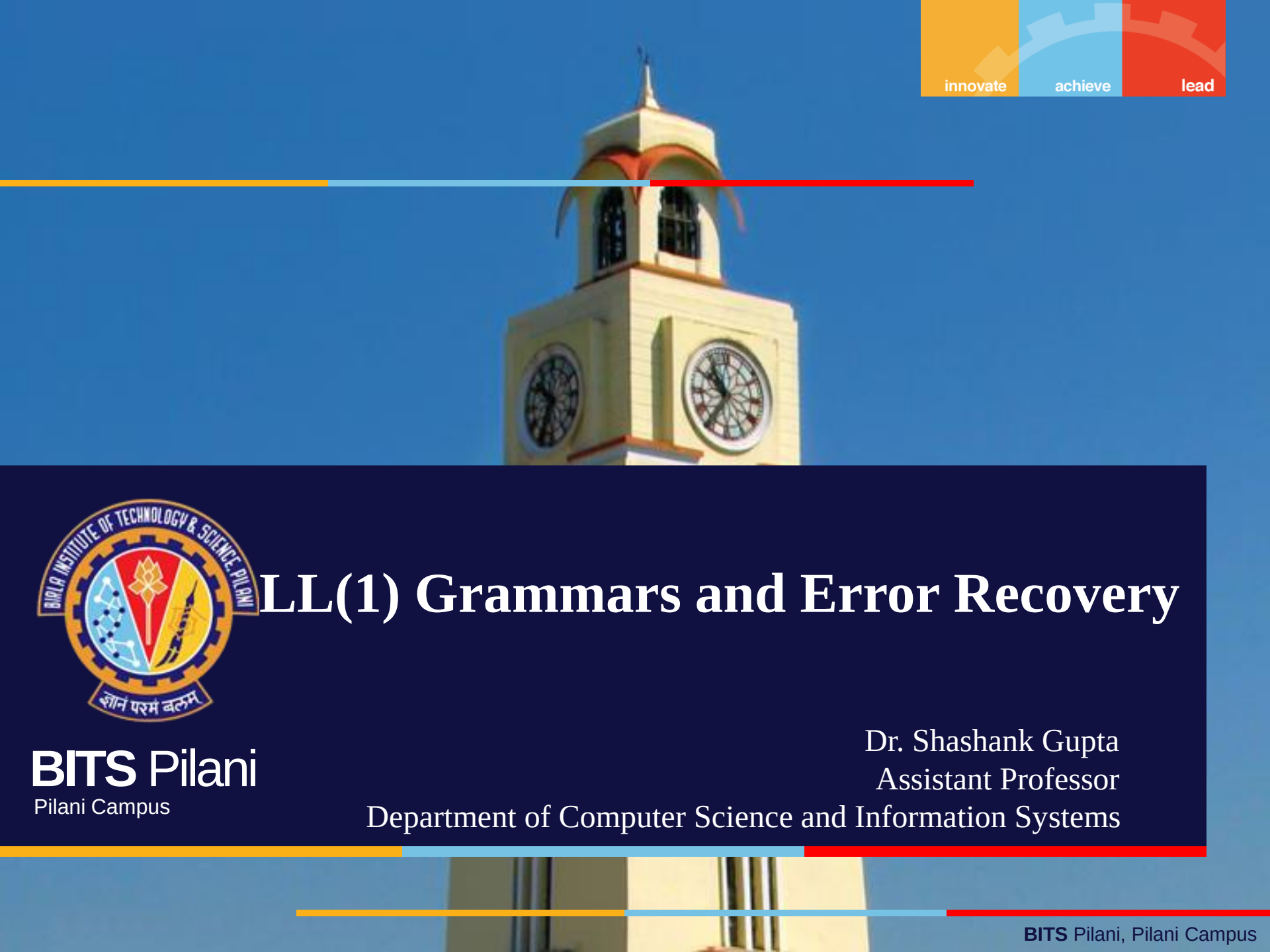

innovate achieve lead

LL(1) Grammars and Error Recovery

Dr. Shashank Gupta
Assistant Professor
Department of Computer Science and Information Systems

BITS Pilani
Pilani Campus

Designing of Parse Table

for each production A → α do

– for each terminal ‘a’ in first(α)

M[A, a] = A → α

– if ε is in First(α)

M[A, b] = A → α

for each terminal b in **follow(A)**

– if ε is in First(α) and $ is in **follow(A)**

M[A, $] = A → α

Algorithm of Predictive Parsing

- Initially, predictive parser assumes 'X' symbol on top of stack, and 'a' the current input symbol.
- Consider '$' is a special token that is at the bottom of the stack and also terminates the input string.
- if X = a = $ then stop
- if X = a ≠ $ then pop(X) and increment the look ahead pointer.
- if X is a non terminal

  then if M[X,a] = {X → PQR}

  then begin pop(X); push(R,Q,P)

  end

  else error

More Examples on LL(1) Parser

Construct a Predictive parsing table for the following Grammar

$$E \rightarrow T\,E^{'}$$

$$E^{'} \rightarrow +T\,E^{'} \mid \in$$

$$T \rightarrow F\,T^{'}$$

$$T^{'} \rightarrow *F\,T^{'} \mid \in$$

$$F \rightarrow id \mid (E)$$

Predictive Parsing

$E \rightarrow T\,E'$

$E' \rightarrow +T\,E' \mid \in$

$T \rightarrow F\,T'$

$T' \rightarrow *F\,T' \mid \in$

$F \rightarrow id \mid (E)$

Non-Terminal	First	Follow
E	{id, (}	{$,) }
E'	{+, ∈ }	{$,) }
T	{id, (}	{+,), $}
T'	{*, ∈ }	{+,), $}
F	{id, (}	{*, +,), $}

Terminals **Non-Terminals**	**id**	**+**	*****	**(**	**)**	**$**
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'						
T						
T'						
F						

Predictive Parsing

$E \rightarrow T\,E'$

$E' \rightarrow +T\,E' \mid \in$

$T \rightarrow F\,T'$

$T' \rightarrow *F\,T' \mid \in$

$F \rightarrow id \mid (E)$

Non-Terminal	First	Follow
E	{id, (}	{$,) }
E'	{+, ∈ }	{$,) }
T	{id, (}	{+,), $}
T'	{*, ∈ }	{+,), $}
F	{id, (}	{*, +,), $}

Terminals **Non-Terminals**	**id**	**+**	*****	**(**	**)**	**$**
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \in$	$E' \rightarrow \in$
T						
T'						
F						

Predictive Parsing

$E \rightarrow T\,E'$

$E' \rightarrow +T\,E' \mid \in$

$T \rightarrow F\,T'$

$T' \rightarrow *F\,T' \mid \in$

$F \rightarrow id \mid (E)$

Non-Terminal	First	Follow
E	{id, (}	{$,) }
E'	{+, ∈ }	{$,) }
T	{id, (}	{+,), $}
T'	{*, ∈ }	{+,), $}
F	{id, (}	{*, +,), $}

Terminals **Non-Terminals**	**id**	**+**	*****	**(**	**)**	**$**
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \in$	$E' \rightarrow \in$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'						
F						

Predictive Parsing

$E \rightarrow T\,E'$

$E' \rightarrow +T\,E' \mid \in$

$T \rightarrow F\,T'$

$T' \rightarrow *F\,T' \mid \in$

$F \rightarrow id \mid (E)$

Non-Terminal	First	Follow
E	{id, (}	{$,) }
E'	{+, ∈ }	{$,) }
T	{id, (}	{+,), $}
T'	{*, ∈ }	{+,), $}
F	{id, (}	{*, +,), $}

Terminals **Non-Terminals**	**id**	**+**	*****	**(**	**)**	**$**
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \in$	$E' \rightarrow \in$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \in$	$T' \rightarrow *FT'$		$T' \rightarrow \in$	$T' \rightarrow \in$
F						

Predictive Parsing

$E \rightarrow T\,E'$

$E' \rightarrow +T\,E' \mid \in$

$T \rightarrow F\,T'$

$T' \rightarrow *F\,T' \mid \in$

$F \rightarrow id \mid (E)$

Non-Terminal	First	Follow
E	{id, (}	{$,) }
E'	{+, ∈ }	{$,) }
T	{id, (}	{+,), $}
T'	{*, ∈ }	{+,), $}
F	{id, (}	{*, +,), $}

Terminals **Non-Terminals**	**id**	**+**	*****	**(**	**)**	**$**
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \in$	$E' \rightarrow \in$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \in$	$T' \rightarrow *FT'$		$T' \rightarrow \in$	$T' \rightarrow \in$
F	$F \rightarrow id$			$F \rightarrow (E)$		

Predictive Parsing

$E \rightarrow T\,E'$

$E' \rightarrow +T\,E' \mid \in$

$T \rightarrow F\,T'$

$T' \rightarrow *F\,T' \mid \in$

$F \rightarrow id \mid (E)$

Non-Terminal	First	Follow
E	{id, (}	{$,) }
E'	{+, ∈ }	{$,) }
T	{id, (}	{+,), $}
T'	{*, ∈ }	{+,), $}
F	{id, (}	{*, +,), $}

Terminals **Non-Terminals**	**id**	**+**	*****	**(**	**)**	**$**
E	$E \rightarrow TE'$	**Error**	**Error**	$E \rightarrow TE'$	**Error**	**Error**
E'	**Error**	$E' \rightarrow +TE'$	**Error**	**Error**	$E' \rightarrow \in$	$E' \rightarrow \in$
T	$T \rightarrow FT'$	**Error**	**Error**	$T \rightarrow FT'$	**Error**	**Error**
T'	**Error**	$T' \rightarrow \in$	$T' \rightarrow *FT'$	**Error**	$T' \rightarrow \in$	$T' \rightarrow \in$
F	$F \rightarrow id$	**Error**	**Error**	$F \rightarrow (E)$	**Error**	**Error**

LL(1) Parsing

Input to be Parsed: **id+id$**

Stack	Input	Action
$	id+id$	Push E
$E	id+id$	Pop E, Push E`T
$ E`T	id+id$	Pop T, Push T`F
$ E`T`F	id+id$	Pop F, Push id
$ E`T`id	id+id$	Pop id, ILP++
$ E`T`	+id$	Pop T`
$ E`	+id$	Pop E`, Push E`T+
$ E`T+	+id$	Pop +, ILP++
$ E`T	id$	Pop T, Push T`F
$ E`T`F	id$	Pop F, Push id
$ E`T`id	id$	Pop id, ILP++
$ E`T`	$	Pop T`
$ E`	$	Pop E`
$	$	**STOP!!!! Parsing DONE**

Terminals **Non-Terminals**	**id**	**+**	*****	**(**	**)**	**$**
E	$E \rightarrow TE'$	**Error**	**Error**	$E \rightarrow TE'$	**Error**	**Error**
E'	**Error**	$E' \rightarrow +TE'$	**Error**	**Error**	$E' \rightarrow \in$	$E' \rightarrow \in$
T	$T \rightarrow FT'$	**Error**	**Error**	$T \rightarrow FT'$	**Error**	**Error**
T'	**Error**	$T' \rightarrow \in$	$T' \rightarrow *FT'$	**Error**	$T' \rightarrow \in$	$T' \rightarrow \in$
F	$F \rightarrow id$	**Error**	**Error**	$F \rightarrow (E)$	**Error**	**Error**

LL(1) GRAMMAR

- NON-LEFT RECURSIVE
- LEFT-FACTORED
- UNAMBIGUOUS

Conditions for a Grammar to be LL(1)

Table can be constructed if for every non-terminal, every lookahead symbol can be handled by at most one production.

- If there are multiple productions, grammar must not be LL(1).

A grammar whose parse table has no multiple defined entries is known as LL(1) Grammar.

Conditions for Grammar to be in LL (1)

$$A \rightarrow \alpha_1 | \alpha_2 | \alpha_3 | ----- | \alpha_n$$

$$First(\alpha_1) \cap (First(\alpha_2) \cap (First(\alpha_3) ------ \cap First(\alpha_n) = \phi$$

OR

$$A \rightarrow \alpha | \in$$

$$First(\alpha) \cap (Follow(A) = \phi$$

On the other hand, if the parse table is containing multiple entries of same production in the same cell, then Grammar is not LL(1).

Examples

Consider the following grammar and find out whether it is LL(1) or not.

$$S \rightarrow aABb$$

$$A \rightarrow c \mid \in$$

$$B \rightarrow d \mid \in$$

For $A \rightarrow c \mid \in$, $First(c) \cap Follow(A) = \phi \Rightarrow c \cap \{d, b\} = \phi$

For $B \rightarrow d \mid \in$, $First(d) \cap Follow(B) = \phi \Rightarrow d \cap b = \phi$

Hence, the Grammar is LL(1).

More Examples

Construct Parse Table for the following Grammar and find out whether it is LL(1) or not.

$$S \rightarrow a\,S\,b\,S$$
$$| \; b\,S\,a\,S$$
$$| \; \in$$

Terminals **Non Terminals**	**a**	**b**	**$**
S	$S \rightarrow a\,Sb\,S$ $S \rightarrow \in$	$S \rightarrow bSaS$ $S \rightarrow \in$	$S \rightarrow \in$

There are multiple defined entries in the parse table. Hence, the Grammar is not LL (1).

Examples

Consider the following two grammars and find out whether it is LL(1) or not.

$$S \rightarrow i\,E\,t\,S\,S^{'} \mid a$$

$$S^{'} \rightarrow e\,S \mid \in$$

$$E \rightarrow b$$

$$S \rightarrow aSA \mid \in$$

$$A \rightarrow c \mid \in$$

ERROR RECOVERY

Error Recovery

Stop immediately at the first error.

Compiler must recover from errors and identify as many errors as possible.

- Panic mode is the most frequently used error recovery technique.

Panic Mode

When an error is encountered anywhere in the statement, the rest of the statement is ignored by not processing the input from erroneous input to delimiters.

This mode prevents the parser from developing infinite loops and is considered as the easiest way for recovery of the errors.

Panic Mode

Consider the following code

```
a = p - q;
s = p r;
y = z / 2;
```

- **Second statement has syntactical error.**
- **Panic mode error recovery will skip all the tokens till ; and try to parse the next expression.**
- **May fail also if no further ; is detected.**

Panic Mode

Error detection happens when an entry in parse table is found written error i.e. `M[A, a] = error`

- Keep discarding the tokens until you hit any token, which comes in the First set of any non-terminal of grammar that is placed on the top of stack and resume parsing from that token.

Place the symbols in follow (A) in syn set in parse table. Skip the tokens until an element in follow (A) is seen. Pop (A) and continue parsing from that token.

- Skip the tokens in an i/p until a token in syn set appears.

Error Recovery in Predictive Parsing

$E \rightarrow T\,E'$

$E' \rightarrow +T\,E' \mid \in$

$T \rightarrow F\,T'$

$T' \rightarrow *F\,T' \mid \in$

$F \rightarrow id \mid (E)$

Non-Terminal	First	Follow
E	{id, (}	{$,) }
E'	{+, ∈ }	{$,) }
T	{id, (}	{+,), $}
T'	{*, ∈ }	{+,), $}
F	{id, (}	{*, +,), $}

Terminals **Non-Terminals**	**id**	**+**	*****	**(**	**)**	**$**
E	$E \rightarrow TE'$	**Error**	**Error**	$E \rightarrow TE'$	**Error**	**Error**
E'	**Error**	$E' \rightarrow +TE'$	**Error**	**Error**	$E' \rightarrow \in$	$E' \rightarrow \in$
T	$T \rightarrow FT'$	**Error**	**Error**	$T \rightarrow FT'$	**Error**	**Error**
T'	**Error**	$T' \rightarrow \in$	$T' \rightarrow *FT'$	**Error**	$T' \rightarrow \in$	$T' \rightarrow \in$
F	$F \rightarrow id$	**Error**	**Error**	$F \rightarrow (E)$	**Error**	**Error**

Error Recovery

$E \rightarrow T\,E'$

$E' \rightarrow +T\,E' \mid \in$

$T \rightarrow F\,T'$

$T' \rightarrow *F\,T' \mid \in$

$F \rightarrow id \mid (E)$

Non-Terminal	First	Follow
E	{id, (}	{$,) }
E'	{+, ∈}	{$,) }
T	{id, (}	{+,), $}
T'	{*, ∈}	{+,), $}
F	{id, (}	{*, +,), $}

Terminals / Non-Terminals	id	+	*	(	)	$
E	$E \rightarrow TE'$	Error	Error	$E \rightarrow TE'$	syn	syn
E'	Error	$E' \rightarrow +TE'$	Error	Error	$E' \rightarrow \in$	$E' \rightarrow \in$
T	$T \rightarrow FT'$	syn	Error	$T \rightarrow FT'$	syn	syn
T'	Error	$T' \rightarrow \in$	$T' \rightarrow *FT'$	Error	$T' \rightarrow \in$	$T' \rightarrow \in$
F	$F \rightarrow id$	syn	syn	$F \rightarrow (E)$	syn	syn

Error Recovery through Panic Mode

As soon as, you hit any error state, do the following

Keep discarding the token until you hit any token, that comes in the follow set of any non-terminal of grammar that is placed on the top of stack. Pop that non-terminal from the stack and continue parsing from that token.

Keep discarding the tokens until you hit any token, which comes in the First set of any non-terminal of grammar that is placed on the top of stack. Resume parsing from this token.

Example

Parse the input id+*id and show error recovery

Terminals / Non-Terminals	id	+	*	(	)	$
E	$E \rightarrow TE'$	Error	Error	$E \rightarrow TE'$	syn	syn
E'	Error	$E' \rightarrow +TE'$	Error	Error	$E' \rightarrow \in$	$E' \rightarrow \in$
T	$T \rightarrow FT'$	syn	Error	$T \rightarrow FT'$	syn	syn
T'	Error	$T' \rightarrow \in$	$T' \rightarrow *FT'$	Error	$T' \rightarrow \in$	$T' \rightarrow \in$
F	$F \rightarrow id$	syn	syn	$F \rightarrow (E)$	syn	syn

Stack	Input	Action
$E	id + * id $	Expand by E->TE’
$E’T	id + * id $	Expand by T->FT’
$E’T’F	id + * id $	Expand by F->id
$E’T’id	id + * id $	Pop id and ip++
$E’T’	+ * id $	Expand by T’->ε
$E’	+ * id $	Expand by E’->+TE’
$E’T+	+ * id $	Pop + and ip++

Example

Terminals / Non-Terminals	id	+	*	(	)	$
E	$E \rightarrow TE'$	Error	Error	$E \rightarrow TE'$	syn	syn
E'	Error	$E' \rightarrow +TE'$	Error	Error	$E' \rightarrow \in$	$E' \rightarrow \in$
T	$T \rightarrow FT'$	syn	Error	$T \rightarrow FT'$	syn	syn
T'	Error	$T' \rightarrow \in$	$T' \rightarrow *FT'$	Error	$T' \rightarrow \in$	$T' \rightarrow \in$
F	$F \rightarrow id$	syn	syn	$F \rightarrow (E)$	syn	syn

Stack	Input	Action
$E'T+	+ * id $	Pop + and ip++
$E'T	* id $	Discard i/p symbol * and ip++
$E'T	id $	Expand by T->FT'
$E'T'F	id $	Expand by F->id
$E'T'id	id $	Pop id and ip++
$E'T'	$	Expand by T'->ε
$E'	$	Expnad by E'->ε
$	$	**Accept**