Left recursion & Left factoring

Left recursion

- A grammar is said to be left recursive if it has a non terminal A such that there is a derivation $A \rightarrow A\alpha$ for some string α.

Algorithm to eliminate left recursion

1. Arrange the non terminals in some order $A_1, \ldots, An$
2. For $i := 1$ *to* n **do begin**
   for $j := 1$ *to* $i - 1$ **do begin**
   replace each production of the form $A_i \rightarrow Ai\gamma$
   by the productions $A_i \rightarrow \delta_1\gamma \mid \delta_2\gamma \mid \ldots\ldots \mid \delta_k\gamma$,
   where $A_j \rightarrow \delta_1 \mid \delta_2 \mid \ldots\ldots \mid \delta_k$ are all the current A_j productions;
   end
   eliminate the immediate left recursion among the A_i - productions

end

Left recursion elimination

$$A \rightarrow A\alpha \mid \beta \longrightarrow \begin{array}{l} A \rightarrow \quad A' \\ A' \rightarrow \quad A' \mid \epsilon \end{array}$$

Examples: Left recursion elimination

E→E+T | T

E→TE'

E'→+TE' | ε

T→T*F | F

T→FT'

T'→*FT' | ε

X→X%Y | Z

X→ZX'

X'→%YX' | ε

Exercise: Left recursion

1. A→Abd | Aa | a

   B→Be | b
2. A→AB | AC | a | b
3. S→A | B

   A→ABC | Acd | a | aa

   B→Bee | b
4. Exp→Exp+term | Exp-term | term

Left factoring

Left factoring is a grammar transformation that is useful for producing a grammar suitable for predictive parsing.

Algorithm to left factor a grammar

Input: Grammar G

Output: An equivalent left factored grammar.

Method:

For each non terminal A find the longest prefix α common to two or more of its alternatives. If $\alpha \neq \in$, i.e., there is a non trivial common prefix, replace all the A productions $A \rightarrow \alpha\beta_1 | \alpha\beta_2 | \ldots\ldots\ldots\ldots | \alpha\beta_n | \gamma$ where γ represents all alternatives that do not begin with α by

$$A \rightarrow \alpha A' | \gamma$$
$$A' \rightarrow \beta_1 | \beta_2 | \ldots\ldots\ldots\ldots | \beta_n$$

Here A' is new non terminal. Repeatedly apply this transformation until no two alternatives for a non-terminal have a common prefix.

Left factoring elimination

$$A \rightarrow \alpha\,\beta \mid \alpha\,\delta \longrightarrow \begin{array}{l} A \rightarrow \quad A' \\ A' \rightarrow \quad \mid \end{array}$$

Example: Left factoring elimination

S→aAB | aCD

S→aS'

S'→AB | CD

A→ xByA | xByAzA | a

A→ xByAA' | a

A'→ Є | zA

A→ aAB | aA |a

A→aA'

A'→AB | A | ϵ

A'→AA'' | ϵ

A''→B | ϵ

Exercise

1. S→iEtS | iEtSeS | a
2. A→ ad | a | ab | abc | x

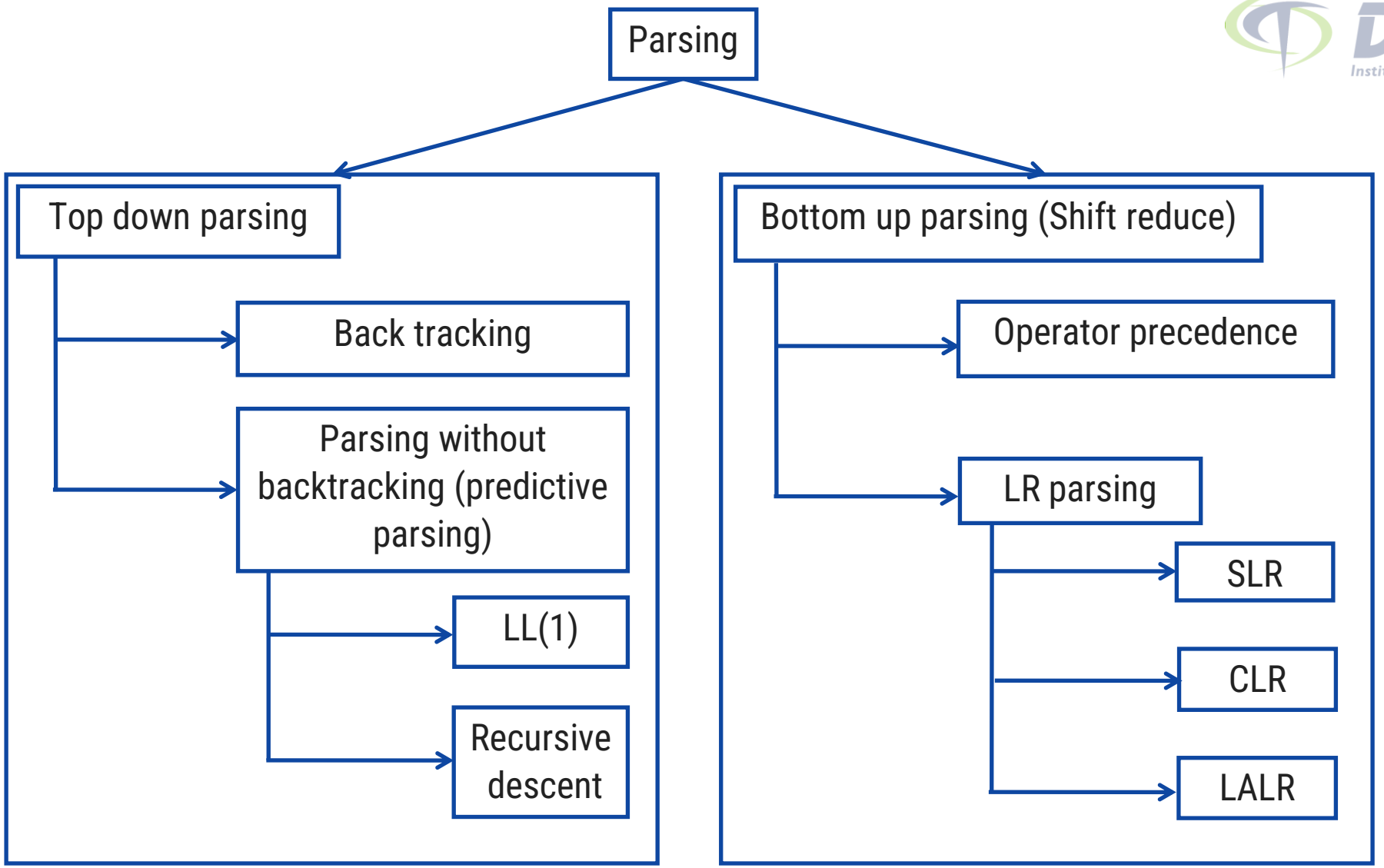
Parsing

Parsing

- Parsing is a technique that takes input string and produces output either a parse tree if string is valid sentence of grammar, or an error message indicating that string is not a valid.
- Types of parsing are:

1. **Top down parsing**: In top down parsing parser build parse tree from top to bottom.
2. **Bottom up parsing**: Bottom up parser starts from leaves and work up to the root.

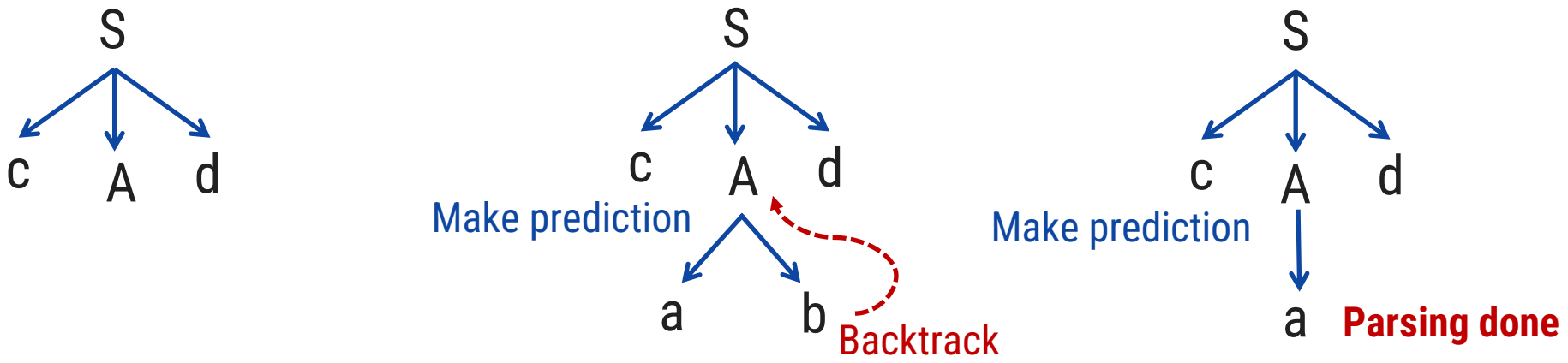
Classification of parsing methods



Backtracking

- In backtracking, expansion of nonterminal symbol we choose one alternative and if any mismatch occurs then we try another alternative.
- Grammar: S→ cAd Input string: cad

  A→ ab | a



Exercise

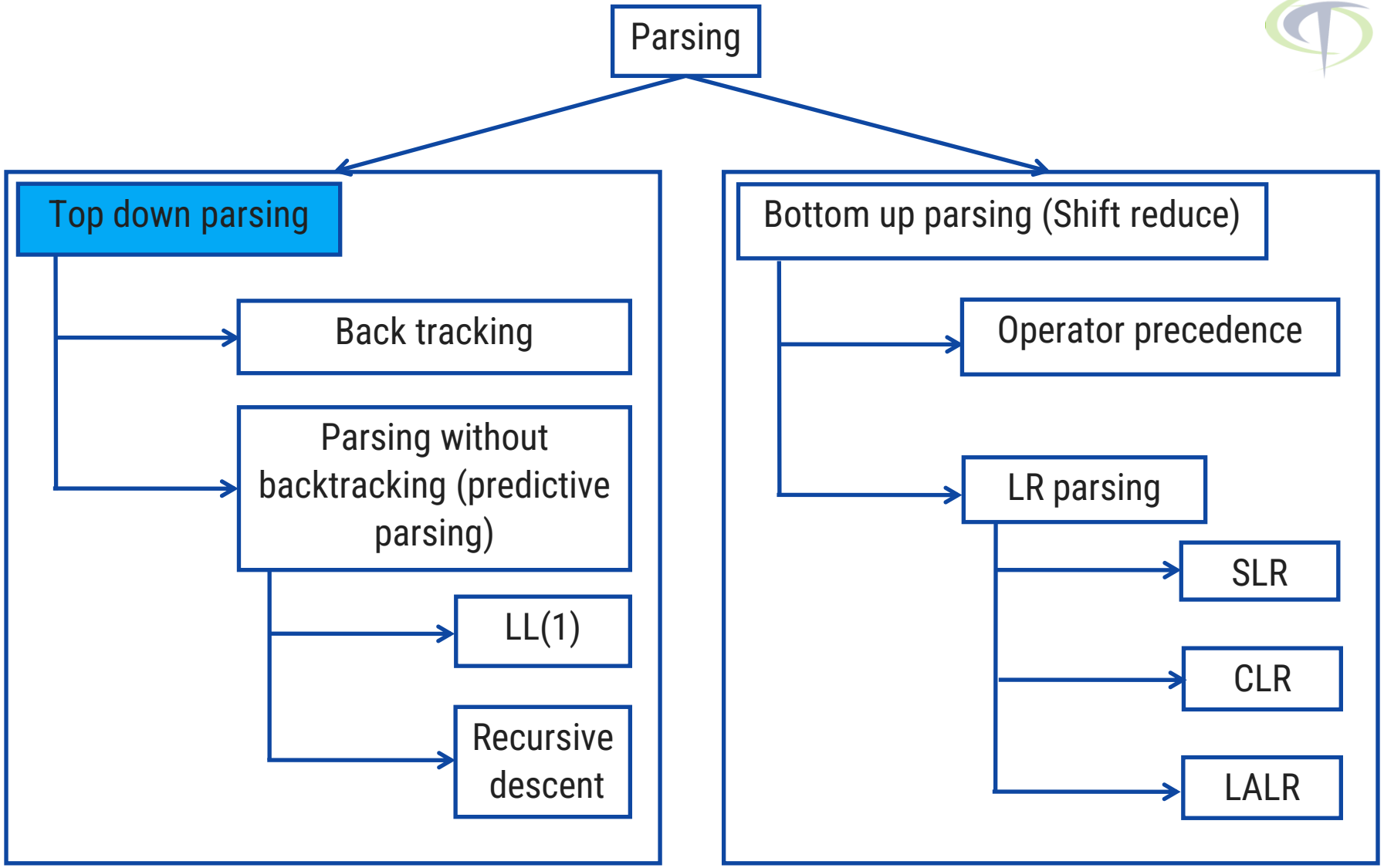
1. E→ 5+T | 3-T

   T→ V | V*V | V+V

   V→ a | b

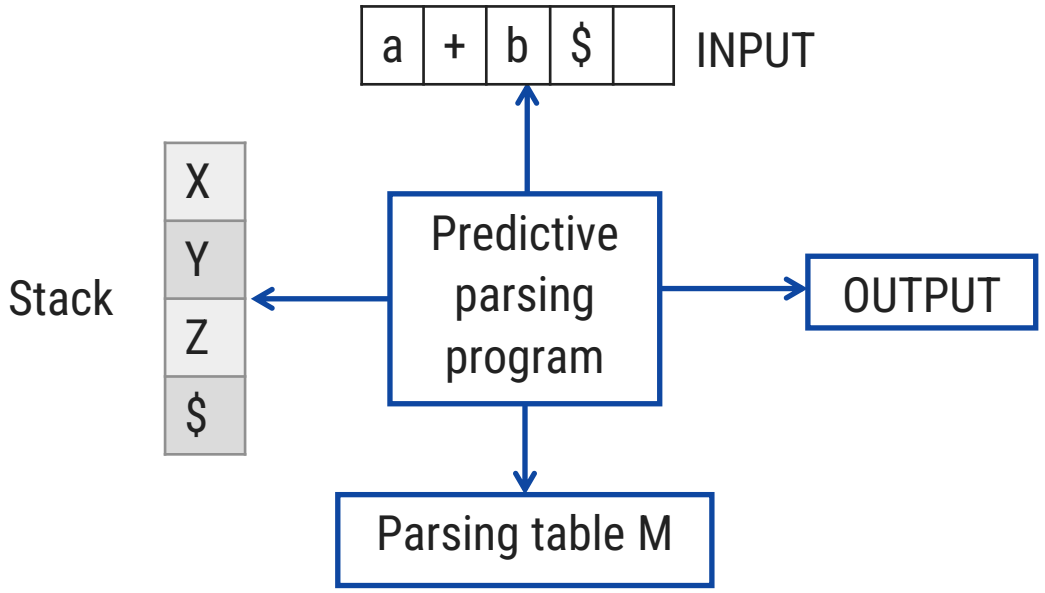
   String: 3-a+b

Parsing Methods



LL(1) parser (predictive parser)

- LL(1) is non recursive top down parser.
  1. First **L** indicates input is scanned from left to right.
  2. The second **L** means it uses leftmost derivation for input string
  3. **1** means it uses only input symbol to predict the parsing process.



LL(1) parsing (predictive parsing)

Steps to construct LL(1) parser

1. Remove left recursion / Perform left factoring (if any).
2. Compute FIRST and FOLLOW of non terminals.
3. Construct predictive parsing table.
4. Parse the input string using parsing table.

Rules to compute first of non terminal

1. If $A \rightarrow \alpha$ and α is terminal, add α to $FIRST(A)$.
2. If $A \rightarrow \in$, add $\in$ to $FIRST(A)$.
3. If X is nonterminal and $X \rightarrow Y_1\ Y_2 \ldots . Y_k$ is a production, then place a in $FIRST(X)$ if for some i, a is in $FIRST(Yi)$, and ϵ is in all of $FIRST(Y_1), \ldots \ldots \ldots, FIRST(Y_{i-1})$; that is $Y_1 \ldots Y_{i-1} \Rightarrow \epsilon$. If ϵ is in $FIRST(Y_j)$ for all $j = 1,2, \ldots . ., k$ then add ϵ to $FIRST(X)$.

   Everything in $FIRST(Y_1)$ is surely in $FIRST(X)$ If Y_1 does not derive ϵ, then we do nothing more to $FIRST(X)$, but if $Y_1 \Rightarrow \epsilon$, then we add $FIRST(Y_2)$ and so on.

Rules to compute first of non terminal

Simplification of Rule 3

If $A \rightarrow Y_1 Y_2 \ldots\ldots Y_K$,

- If Y_1 does not derives $\in$ $then, FIRST(A) = FIRST(Y_1)$
- If Y_1 derives $\in$ $then,$

  $FIRST(A) = FIRST(Y_1) - \epsilon \text{ U } FIRST(Y_2)$
- If Y_1 & Y_2 derives $\in$ $then,$

  $FIRST(A) = FIRST(Y_1) - \epsilon \text{ U } FIRST(Y_2) - \epsilon \; U \; FIRST(Y_3)$
- If Y_1 , Y_2 & Y_3 derives $\in$ $then,$

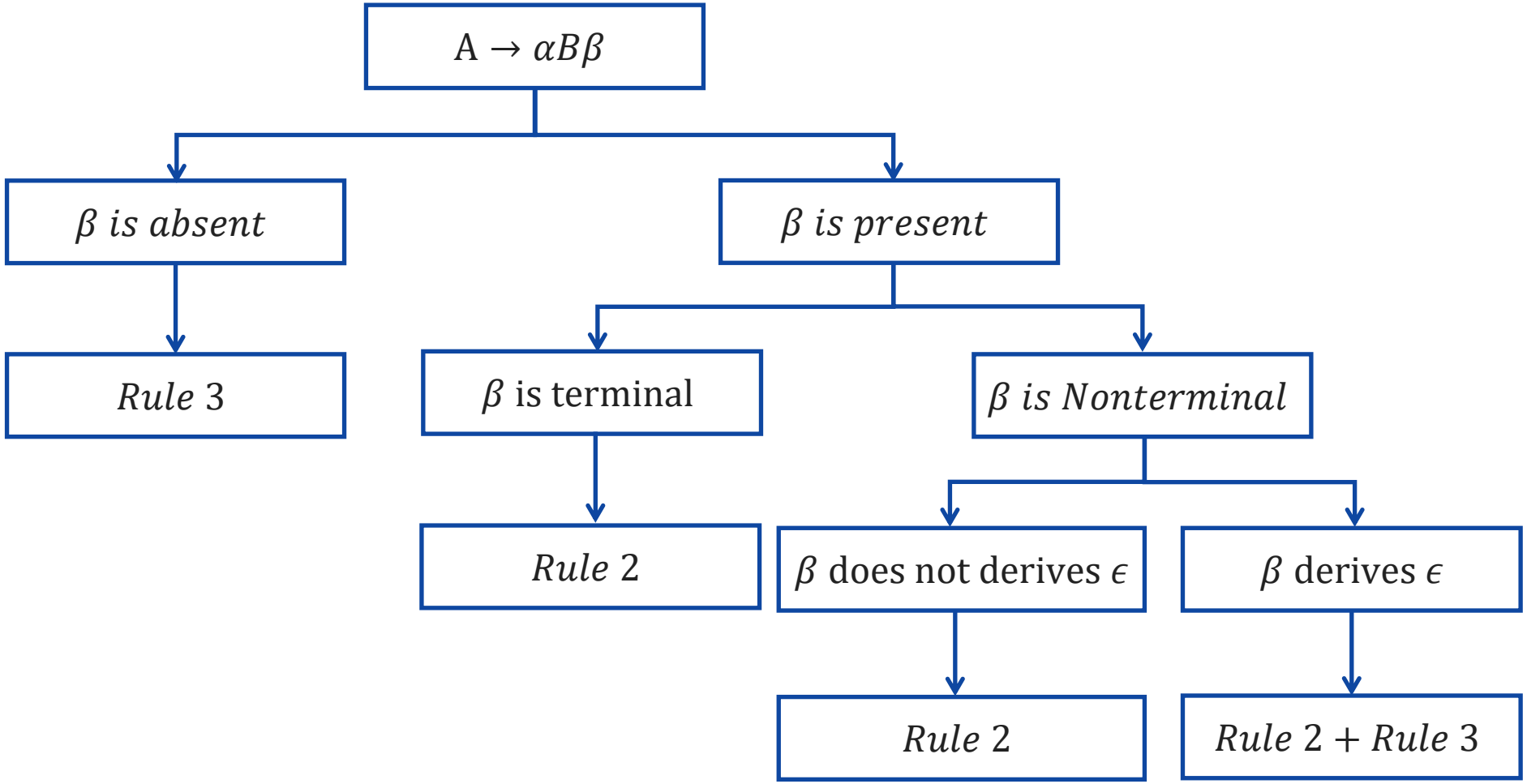
  $FIRST(A) = FIRST(Y_1) - \epsilon \; U \; FIRST(Y_2) - \epsilon \; U \; FIRST(Y_3) - \; \epsilon \; U \; FIRST(Y_4)$
- If Y_1 , Y_2 , Y_3 …..Y_K all derives $\in$ $then,$

  $FIRST(A) = FIRST(Y_1) - \epsilon \; U \; FIRST(Y_2) - \epsilon \; U \; FIRST(Y_3) - \epsilon \; U \; FIRST(Y_4) -$
  $\epsilon \; U \ldots\ldots\ldots\ldots FIRST(Y_k)$ (note: if all non terminals derives $\in$ then add $\in$ to FIRST(A))

Rules to compute FOLLOW of non terminal

1. Place \$ $in\ follow(S)$. (S is start symbol)
2. If $A \rightarrow \alpha B \beta$, then everything in $FIRST(\beta)$ except for ϵ is placed in $FOLLOW(B)$
3. If there is a production $A \rightarrow \alpha B$ or a production $A \rightarrow \alpha B \beta$ where $FIRST(\beta)$ contains ϵ then everything in $FOLLOW(A) = FOLLOW(B)$

How to apply rules to find FOLLOW of non terminal?



Rules to construct predictive parsing table

1. For each production $A \rightarrow \alpha$ of the grammar, do steps 2 and 3.
2. For each terminal a in $first(\alpha)$, Add $A \rightarrow \alpha$ to $M[A, a]$.
3. If ϵ is in $first(\alpha)$, Add $A \rightarrow \alpha$ to $M[A, b]$ for each terminal b in $FOLLOW(B)$. If ϵ is in $first(\alpha)$, and \$ is in $FOLLOW(A)$, add $A \rightarrow \alpha$ to $M[A, \$]$.
4. Make each undefined entry of M be error.

Example-1: LL(1) parsing

S→aBa
B→bB | ϵ

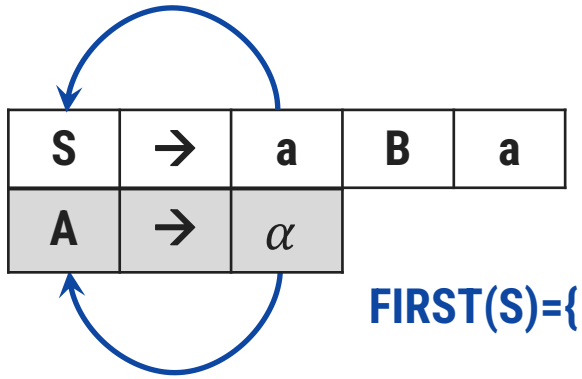
NT	First
S	
B	

Step 1: Not required

Step 2: Compute FIRST

First(S)

S→aBa

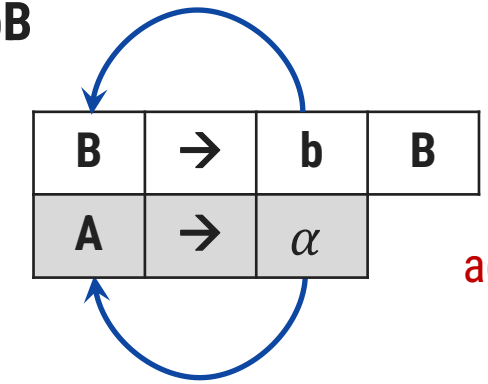


Rule 1
add α to $FIRST(A)$

FIRST(S)={ a }

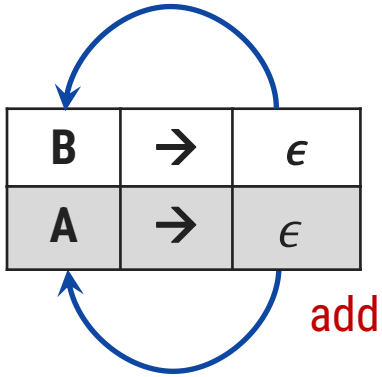
First(B)

B→bB



Rule 1
add α to $FIRST(A)$

B→ϵ



Rule 2
add ϵ to $FIRST(A)$

FIRST(B)={ b , ϵ }

Example-1: LL(1) parsing

S→aBa
B→bB | ϵ

NT	First	Follow
S	{a}	
B	{b,ϵ}	

Step 2: Compute FOLLOW

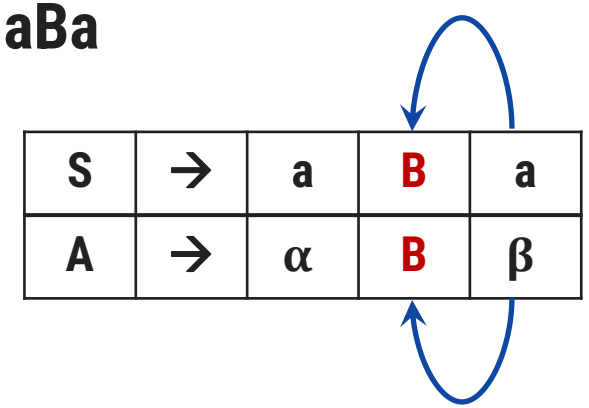
Follow(S)

Rule 1: Place $ in FOLLOW(S)

Follow(S)={ $ }

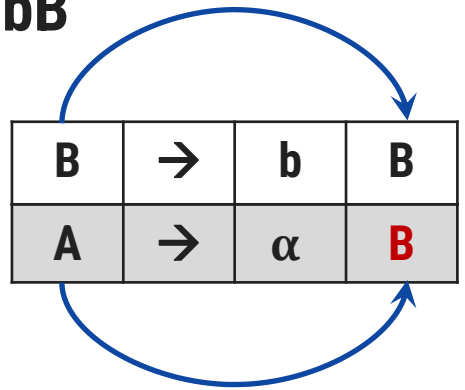
Follow(B)

S→aBa



Rule 2
First(β) − ϵ

B→bB



Rule 3
Follow(A)=follow(B)

Follow(B)={ a }

Example-1: LL(1) parsing

S→aBa
B→bB | ϵ

NT	First	Follow
S	{a}	{$}
B	{b,ϵ}	{a}

Step 3: Prepare predictive parsing table

NT	Input Symbol		
	a	b	$
S			
B			

S→aBa

a=FIRST(aBa)={ a }

M[S,a]=S→aBa

Rule: 2
A→ α
a = first(α)
M[A,a] = A→ α

Example-1: LL(1) parsing

S→aBa
B→bB | ϵ

NT	First	Follow
S	{a}	{$}
B	{b,ϵ}	{a}

Step 3: Prepare predictive parsing table

NT	Input Symbol		
	a	b	$
S	S→aBa		
B			

B→bB

a=FIRST(bB)={ b }

M[B,b]=B→bB

Rule: 2
A→ α
a = first(α)
M[A,a] = A→ α

Example-1: LL(1) parsing

S→aBa
B→bB | ϵ

NT	First	Follow
S	{a}	{$}
B	{b,ϵ}	{a}

Step 3: Prepare predictive parsing table

NT	Input Symbol		
	a	b	$
S	S→aBa		
B		B→bB	

B→ϵ

b=FOLLOW(B)={ a }

M[B,a]=B→ϵ

Rule: 3
A→ α
b = follow(A)
M[A,b] = A→ α

Example-2: LL(1) parsing

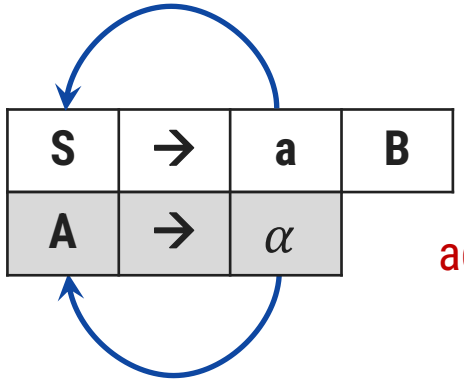
S→aB | ϵ
B→bC | ϵ
C→cS | ϵ

Step 1: Not required

Step 2: Compute FIRST

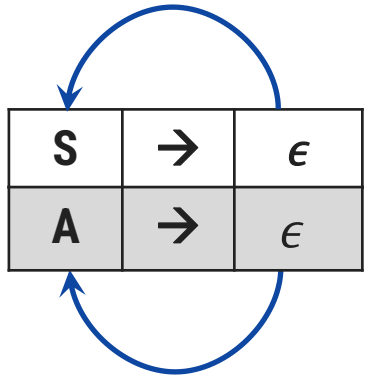
First(S)

S→aB



Rule 1
add α to $FIRST(A)$

S→ϵ



Rule 2
add ϵ to $FIRST(A)$

NT	First
S	
B	
C	

FIRST(S)={ a , ϵ }

Example-2: LL(1) parsing

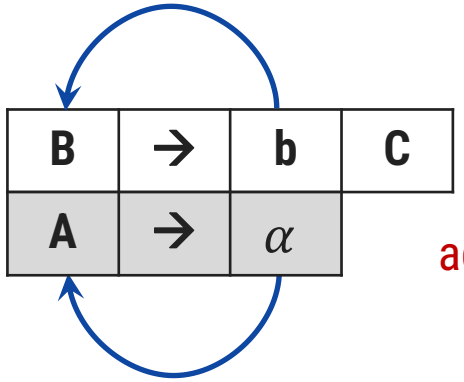
S→aB | ϵ
B→bC | ϵ
C→cS | ϵ

Step 1: Not required

Step 2: Compute FIRST

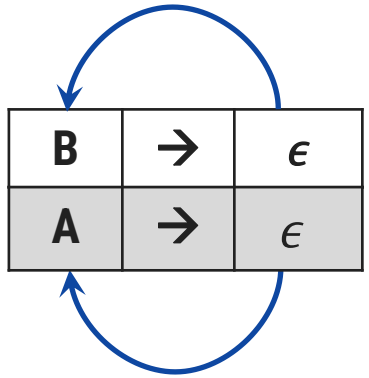
First(B)

B→bC



Rule 1
add α to $FIRST(A)$

B→ϵ



Rule 2
add ϵ to $FIRST(A)$

NT	First
S	{ a, ϵ }
B	
C	

FIRST(B)={ b , ϵ }

Example-2: LL(1) parsing

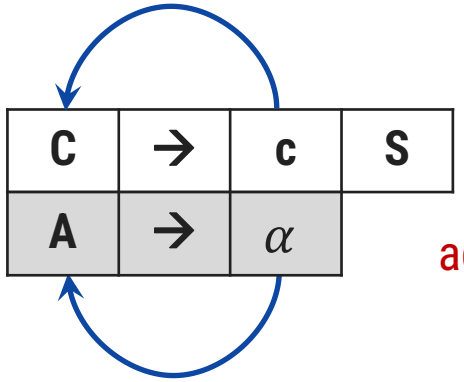
S→aB | ϵ
B→bC | ϵ
C→cS | ϵ

Step 1: Not required

Step 2: Compute FIRST

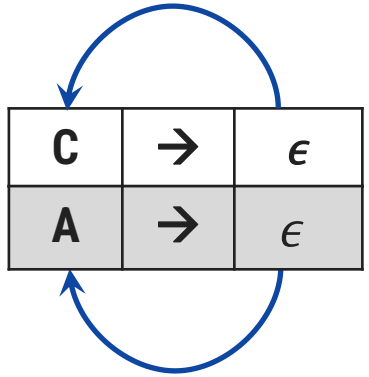
First(C)

C→cS



Rule 1
add α to $FIRST(A)$

C→ϵ



Rule 2
add ϵ to $FIRST(A)$

NT	First
S	{ a, ϵ }
B	{b,ϵ}
C	

FIRST(B)={ c , ϵ }

Example-2: LL(1) parsing

Step 2: Compute FOLLOW

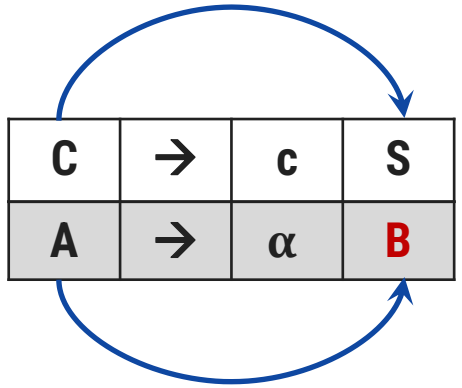
Follow(S)

Rule 1: Place $ in FOLLOW(S)

Follow(S)={ $ }

S→aB | ϵ
B→bC | ϵ
C→cS | ϵ

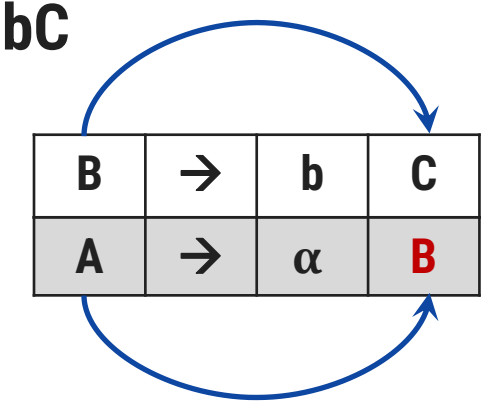
C→cS



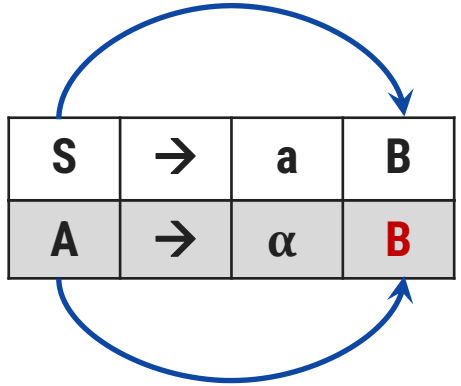
Rule 3
Follow(A)=follow(B)
Follow(S)=Follow(C) ={$}

NT	First	Follow
S	{a,ϵ}	
B	{b,ϵ}	
C	{c,ϵ}	

B→bC



Rule 3
Follow(A)=follow(B)
Follow(C)=Follow(B) ={$}



Rule 3
Follow(A)=follow(B)
Follow(B)=Follow(S)={$}

Example-2: LL(1) parsing

S→aB | ϵ
B→bC | ϵ
C→cS | ϵ

NT	First	Follow
S	{a,ϵ}	{$}
B	{b,ϵ}	{$}
C	{c,ϵ}	{$}

Step 3: Prepare predictive parsing table

NT	Input Symbol			
	a	b	c	$
S				
B				
C				

S→aB

a=FIRST(aB)={ a }

M[S,a]=S→aB

Rule: 2
A→ α
a = first(α)
M[A,a] = A→ α

Example-2: LL(1) parsing

S→aB | ϵ
B→bC | ϵ
C→cS | ϵ

NT	First	Follow
S	{a}	{$}
B	{b,ϵ}	{$}
C	{c,ϵ}	{$}

Step 3: Prepare predictive parsing table

NT	Input Symbol			
	a	b	c	$
S	S→aB			
B				
C				

S→ϵ

b=FOLLOW(S)={ $ }

M[S,$]=S→$\epsilon$

Rule: 3
A→ α
b = follow(A)
M[A,b] = A→ α

Example-2: LL(1) parsing

S→aB | ϵ
B→bC | ϵ
C→cS | ϵ

NT	First	Follow
S	{a}	{$}
B	{b,ϵ}	{$}
C	{c,ϵ}	{$}

Step 3: Prepare predictive parsing table

NT	Input Symbol			
	a	b	c	$
S	S→aB			S→ϵ
B				
C				

B→bC

a=FIRST(bC)={ b }

M[B,b]=B→bC

Rule: 2
A→ α
a = first(α)
M[A,a] = A→ α

Example-2: LL(1) parsing

S→aB | ϵ
B→bC | ϵ
C→cS | ϵ

NT	First	Follow
S	{a}	{$}
B	{b,ϵ}	{$}
C	{c,ϵ}	{$}

Step 3: Prepare predictive parsing table

NT	Input Symbol			
	a	b	c	$
S	S→aB			S→ϵ
B		B→bC		
C				

B→ϵ

b=FOLLOW(B)={ $ }

M[B,$]=B→ϵ

Rule: 3
A→ α
b = follow(A)
M[A,b] = A→ α

Example-2: LL(1) parsing

S→aB | ϵ
B→bC | ϵ
C→cS | ϵ

NT	First	Follow
S	{a}	{$}
B	{b,ϵ}	{$}
C	{c,ϵ}	{$}

Step 3: Prepare predictive parsing table

NT	Input Symbol			
	a	b	c	$
S	S→aB			S→ϵ
B		B→bC		B→ϵ
C				

C→cS

a=FIRST(cS)={ c }

M[C,c]=C→cS

Rule: 2
A→ α
a = first(α)
M[A,a] = A→ α

Example-2: LL(1) parsing

S→aB | ϵ
B→bC | ϵ
C→cS | ϵ

NT	First	Follow
S	{a}	{$}
B	{b,ϵ}	{$}
C	{c,ϵ}	{$}

Step 3: Prepare predictive parsing table

NT	Input Symbol			
	a	b	c	$
S	S→aB			S→ϵ
B		B→bB		B→ϵ
C			C→cS	

C→ϵ

b=FOLLOW(C)={ $ }

M[C,$]=C→ϵ

Rule: 3
A→ α
b = follow(A)
M[A,b] = A→ α

Example-3: LL(1) parsing

E→E+T | T

T→T*F | F

F→(E) | id

Step 1: Remove left recursion

E→TE'

E'→+TE' | ϵ

T→FT'

T'→*FT' | ϵ

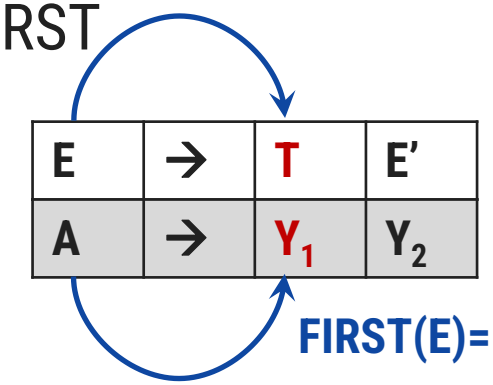
F→(E) | id

Example-3: LL(1) parsing

Step 2: Compute FIRST

First(E)

E→TE'

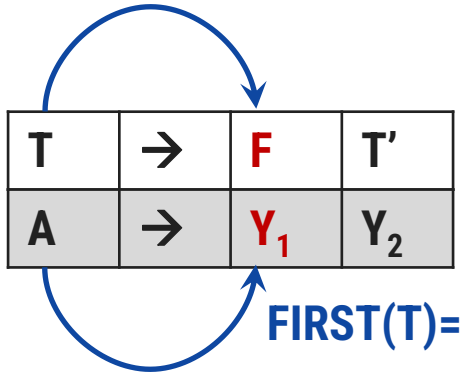


Rule 3
First(A)=First(Y1)

FIRST(E)=FIRST(T)= {(, id }

First(T)

T→FT'

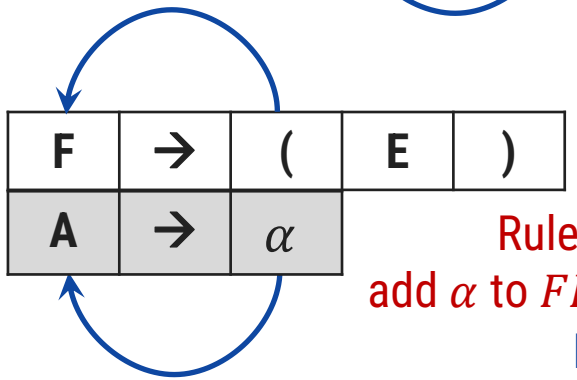


Rule 3
First(A)=First(Y1)

FIRST(T)=FIRST(F)= {(, id }

First(F)

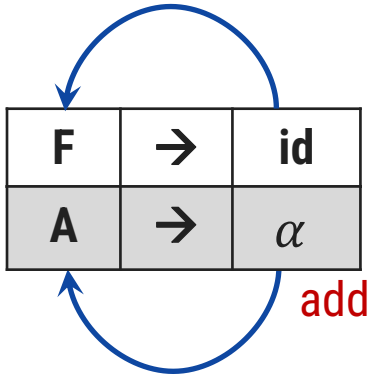
F→(E)



Rule 1
add α to $FIRST(A)$

FIRST(F)={ (, id }

F→id



Rule 1
add α to $FIRST(A)$

E→TE'
E'→+TE' | ϵ
T→FT'
T'→*FT' | ϵ
F→(E) | id

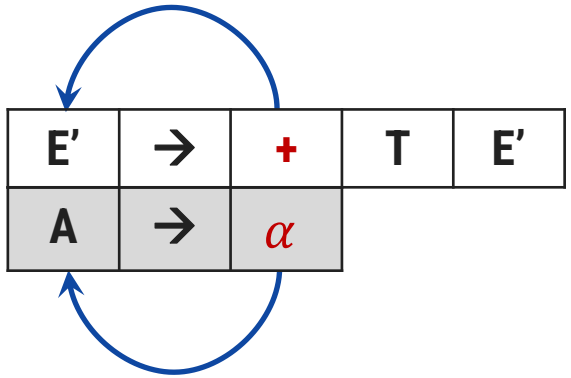
NT	First
E	
E'	
T	
T'	
F	

Example-3: LL(1) parsing

Step 2: Compute FIRST

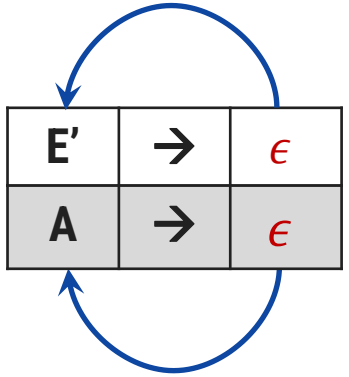
First(E')

E'→+TE'



Rule 1
add α to $FIRST(A)$

E'→ϵ



Rule 2
add ϵ to $FIRST(A)$

FIRST(E')={ + , ϵ }

E→TE'
E'→+TE' | ϵ
T→FT'
T'→*FT' | ϵ
F→(E) | id

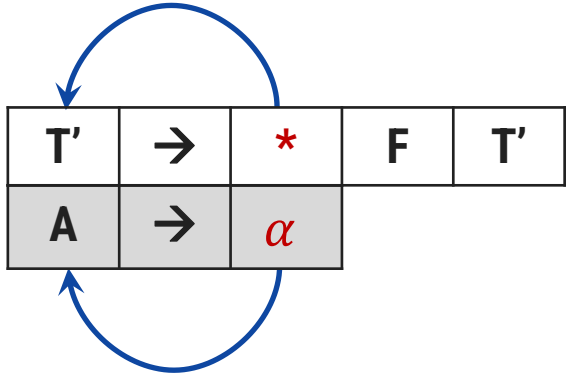
NT	First
E	{ (,id }
E'	
T	{ (,id }
T'	
F	{ (,id }

Example-3: LL(1) parsing

Step 2: Compute FIRST

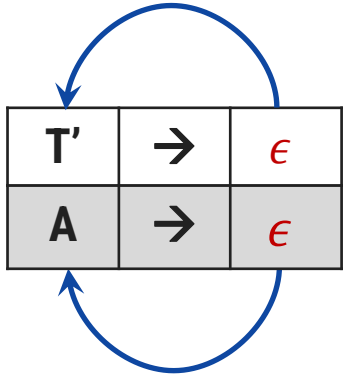
First(T')

T'→*FT'



Rule 1
add α to $FIRST(A)$

T'→ϵ



Rule 2
add ϵ to $FIRST(A)$

FIRST(T')={ *, ϵ }

E→TE'
E'→+TE' | ϵ
T→FT'
T'→*FT' | ϵ
F→(E) | id

NT	First
E	{ (,id }
E'	{ +, ϵ }
T	{ (,id }
T'	
F	{ (,id }

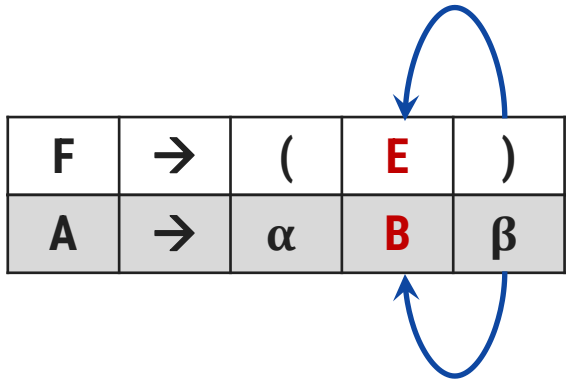
Example-3: LL(1) parsing

Step 2: Compute FOLLOW

FOLLOW(E)

Rule 1: Place $ in FOLLOW(E)

F→(E)



Rule 2

FOLLOW(E)={ $,) }

E→TE'
E'→+TE' | ϵ
T→FT'
T'→*FT' | ϵ
F→(E) | id

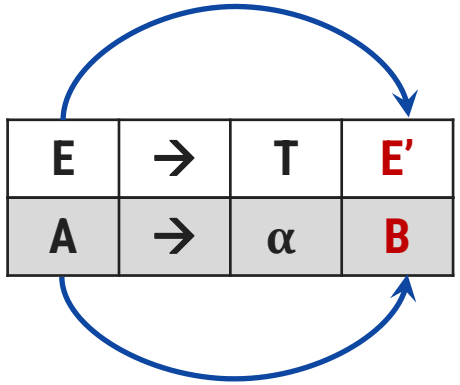
NT	First	Follow
E	{ (,id }	
E'	{ +, ϵ }	
T	{ (,id }	
T'	{ *, ϵ }	
F	{ (,id }	

Example-3: LL(1) parsing

Step 2: Compute FOLLOW

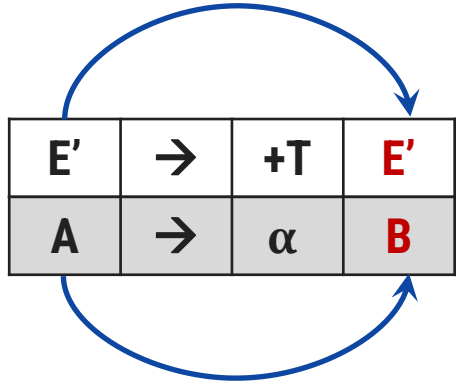
FOLLOW(E')

E→TE'



Rule 3

E'→+TE'



Rule 3

FOLLOW(E')={ $,) }

E→TE'
E'→+TE' | ϵ
T→FT'
T'→*FT' | ϵ
F→(E) | id

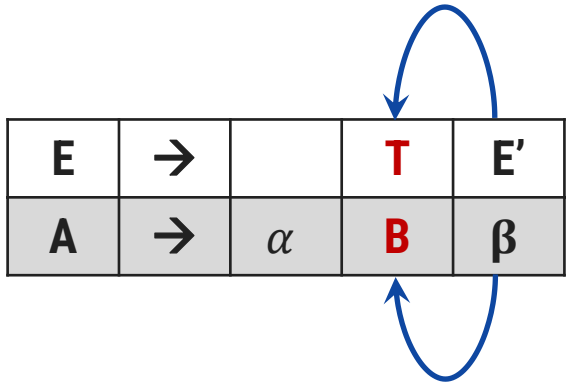
NT	First	Follow
E	{ (,id }	{ $,) }
E'	{ +, ϵ }	
T	{ (,id }	
T'	{ *, ϵ }	
F	{ (,id }	

Example-3: LL(1) parsing

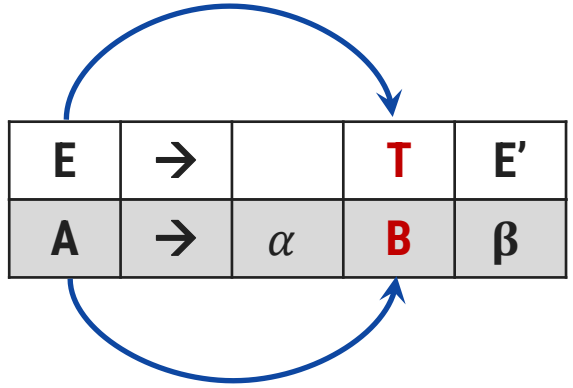
Step 2: Compute FOLLOW

FOLLOW(T)

E→TE'



Rule 2



Rule 3

E→TE'
E'→+TE' | ϵ
T→FT'
T'→*FT' | ϵ
F→(E) | id

NT	First	Follow
E	{ (,id }	{ $,) }
E'	{ +, ϵ }	{ $,) }
T	{ (,id }	
T'	{ *, ϵ }	
F	{ (,id }	

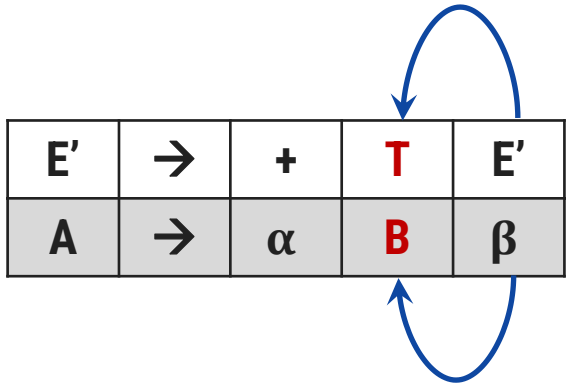
FOLLOW(T)={ +, $,)

Example-3: LL(1) parsing

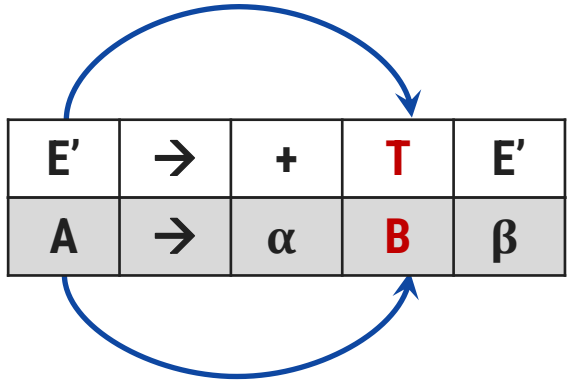
Step 2: Compute FOLLOW

FOLLOW(T)

E'→+TE'



Rule 2



Rule 3

FOLLOW(T)={ +, $,) }

E→TE'
E'→+TE' | ϵ
T→FT'
T'→*FT' | ϵ
F→(E) | id

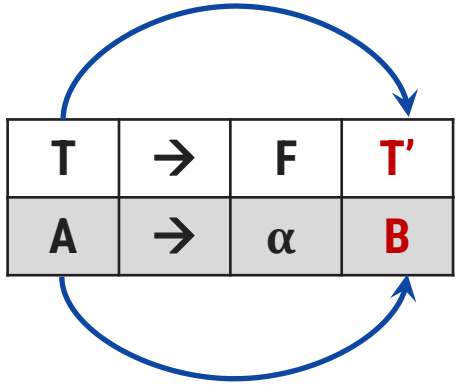
NT	First	Follow
E	{ (,id }	{ $,) }
E'	{ +, ϵ }	{ $,) }
T	{ (,id }	
T'	{ *, ϵ }	
F	{ (,id }	

Example-3: LL(1) parsing

Step 2: Compute FOLLOW

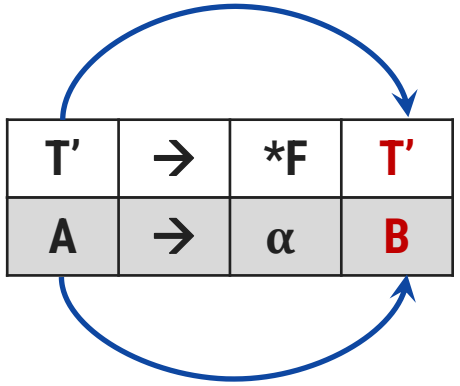
FOLLOW(T')

T→FT'



Rule 3

T'→*FT'



Rule 3

FOLLOW(T')={+ $,) }

E→TE'
E'→+TE' | ϵ
T→FT'
T'→*FT' | ϵ
F→(E) | id

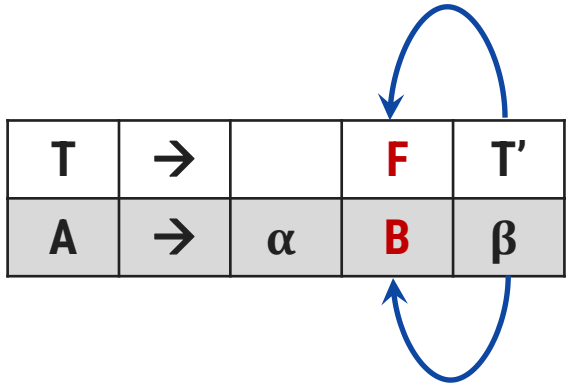
NT	First	Follow
E	{ (,id }	{ $,) }
E'	{ +, ϵ }	{ $,) }
T	{ (,id }	{ +,$,) }
T'	{ *, ϵ }	
F	{ (,id }	

Example-3: LL(1) parsing

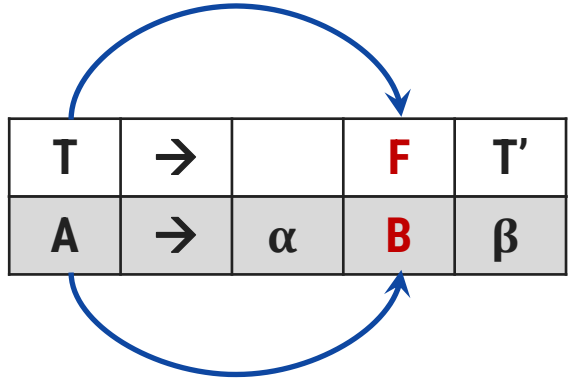
Step 2: Compute FOLLOW

FOLLOW(F)

T→FT'



Rule 2



Rule 3

E→TE'
E'→+TE' | ϵ
T→FT'
T'→*FT' | ϵ
F→(E) | id

NT	First	Follow
E	{ (,id }	{ $,) }
E'	{ +, ϵ }	{ $,) }
T	{ (,id }	{ +,$,) }
T'	{ *, ϵ }	{ +,$,) }
F	{ (,id }	

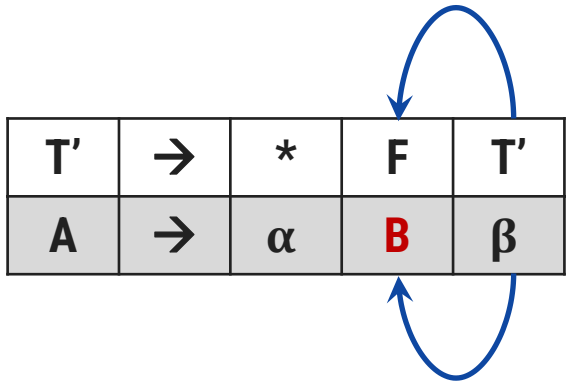
FOLLOW(F)={ *, + ,$,)

Example-3: LL(1) parsing

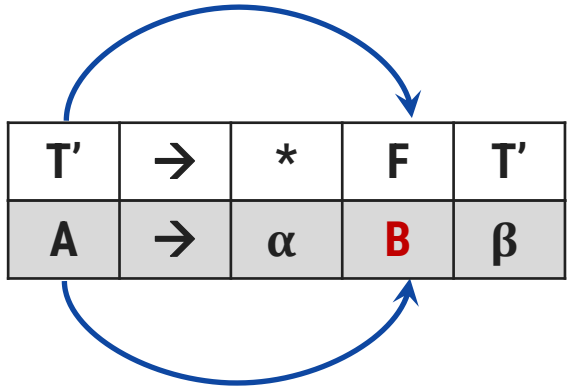
Step 2: Compute FOLLOW

FOLLOW(F)

T'→*FT'



Rule 2



Rule 3

E→TE'
E'→+TE' | ϵ
T→FT'
T'→*FT' | ϵ
F→(E) | id

NT	First	Follow
E	{ (,id }	{ $,) }
E'	{ +, ϵ }	{ $,) }
T	{ (,id }	{ +,$,) }
T'	{ *, ϵ }	{ +,$,) }
F	{ (,id }	

FOLLOW(F)={ *,+, $,) }

Example-3: LL(1) parsing

Step 3: Construct predictive parsing table

E→TE'
E'→+TE' | ϵ
T→FT'
T'→*FT' | ϵ
F→(E) | id

NT	Input Symbol					
	id	+	*	(	)	$
E						
E'						
T						
T'						
F						

NT	First	Follow
E	{ (,id }	{ $,) }
E'	{ +, ϵ }	{ $,) }
T	{ (,id }	{ +,$,) }
T'	{ *, ϵ }	{ +,$,) }
F	{ (,id }	{*,+,$,)}

E→TE'

a=FIRST(TE')={ (,id }

M[E,(]=E→TE'

M[E,id]=E→TE'

Rule: 2
A→ α
a = first(α)
M[A,a] = A→ α

Example-3: LL(1) parsing

Step 3: Construct predictive parsing table

E→TE'
E'→+TE' | ϵ
T→FT'
T'→*FT' | ϵ
F→(E) | id

NT	Input Symbol					
	id	+	*	(	)	$
E	E→TE'			E→TE'		
E'						
T						
T'						
F						

NT	First	Follow
E	{ (,id }	{ $,) }
E'	{ +, ϵ }	{ $,) }
T	{ (,id }	{ +,$,) }
T'	{ *, ϵ }	{ +,$,) }
F	{ (,id }	{*,+,$,)}

E'→+TE'

a=FIRST(+TE')={ + }

M[E',+]=E'→+TE'

Rule: 2
A→ α
a = first(α)
M[A,a] = A→ α

Example-3: LL(1) parsing

Step 3: Construct predictive parsing table

E→TE'
E'→+TE' | ϵ
T→FT'
T'→*FT' | ϵ
F→(E) | id

NT	Input Symbol					
	id	+	*	(	)	$
E	E→TE'			E→TE'		
E'		E'→+TE'				
T						
T'						
F						

NT	First	Follow
E	{ (,id }	{ $,) }
E'	{ +, ϵ }	{ $,) }
T	{ (,id }	{ +,$,) }
T'	{ *, ϵ }	{ +,$,) }
F	{ (,id }	{*,+,$,)}

E'→ϵ

b=FOLLOW(E')={ $,) }

M[E',$]=E'→ϵ

M[E',)]=E'→ϵ

Rule: 3
A→ α
b = follow(A)
M[A,b] = A→ α

Example-3: LL(1) parsing

Step 3: Construct predictive parsing table

E→TE'
E'→+TE' | ϵ
T→FT'
T'→*FT' | ϵ
F→(E) | id

NT	Input Symbol					
	id	+	*	(	)	$
E	E→TE'			E→TE'		
E'		E'→+TE'			E'→ϵ	E'→ϵ
T						
T'						
F						

NT	First	Follow
E	{ (,id }	{ $,) }
E'	{ +, ϵ }	{ $,) }
T	{ (,id }	{ +,$,) }
T'	{ *, ϵ }	{ +,$,) }
F	{ (,id }	{*,+,$,)}

T→FT'

a=FIRST(FT')={ (,id }

M[T,(]=T→FT'

M[T,id]=T→FT'

Rule: 2
A→ α
a = first(α)
M[A,a] = A→ α

Example-3: LL(1) parsing

Step 3: Construct predictive parsing table

NT	Input Symbol					
	id	+	*	(	)	$
E	E→TE'			E→TE'		
E'		E'→+TE'			E'→ϵ	E'→ϵ
T	T→FT'			T→FT'		
T'						
F						

E→TE'
E'→+TE' | ϵ
T→FT'
T'→*FT' | ϵ
F→(E) | id

NT	First	Follow
E	{ (,id }	{ $,) }
E'	{ +, ϵ }	{ $,) }
T	{ (,id }	{ +,$,) }
T'	{ *, ϵ }	{ +,$,) }
F	{ (,id }	{*,+,$,)}

T'→*FT'

a=FIRST(*FT')={ * }

M[T',*]=T'→*FT'

Rule: 2
A→ α
a = first(α)
M[A,a] = A→ α

Example-3: LL(1) parsing

Step 3: Construct predictive parsing table

NT	Input Symbol					
	id	+	*	(	)	$
E	E→TE'			E→TE'		
E'		E'→+TE'			E'→ϵ	E'→ϵ
T	T→FT'			T→FT'		
T'			T'→*FT'			
F						

T'→ϵ

b=FOLLOW(T')={ +,$,) }

M[T',+]=T'→ϵ

M[T',$]=T'→$\epsilon$

M[T',)]=T'→ϵ

Rule: 3
A→ α
b = follow(A)
M[A,b] = A→ α

E→TE'
E'→+TE' | ϵ
T→FT'
T'→*FT' | ϵ
F→(E) | id

NT	First	Follow
E	{ (,id }	{ $,) }
E'	{ +, ϵ }	{ $,) }
T	{ (,id }	{ +,$,) }
T'	{ *, ϵ }	{ +,$,) }
F	{ (,id }	{*,+,$,)}

Example-3: LL(1) parsing

Step 3: Construct predictive parsing table

NT	Input Symbol					
	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						

E→TE'
E'→+TE' | ϵ
T→FT'
T'→*FT' | ϵ
F→(E) | id

NT	First	Follow
E	{ (,id }	{ $,) }
E'	{ +, ϵ }	{ $,) }
T	{ (,id }	{ +,$,) }
T'	{ *, ϵ }	{ +,$,) }
F	{ (,id }	{*,+,$,)}

F→(E)

a=FIRST((E))={ (}

M[F,(]=F→(E)

Rule: 2
A→ α
a = first(α)
M[A,a] = A→ α

Example-3: LL(1) parsing

Step 3: Construct predictive parsing table

NT	Input Symbol					
	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→(E)		

E→TE'
E'→+TE' | ϵ
T→FT'
T'→*FT' | ϵ
F→(E) | id

NT	First	Follow
E	{ (,id }	{ $,) }
E'	{ +, ϵ }	{ $,) }
T	{ (,id }	{ +,$,) }
T'	{ *, ϵ }	{ +,$,) }
F	{ (,id }	{*,+,$,)}

F→id

a=FIRST(id)={ id }

M[F,id]=F→id

Rule: 2
A→ α
a = first(α)
M[A,a] = A→ α

Example-3: LL(1) parsing

- Step 4: Make each undefined entry of table be Error

NT	Input Symbol					
	id	+	*	(	)	$
E	E→TE'	Error	Error	E→TE'	Error	Error
E'	Error	E'→+TE'	Error	Error	E'→ϵ	E'→ϵ
T	T→FT'	Error	Error	T→FT'	Error	Error
T'	Error	T'→ϵ	T'→*FT'	Error	T'→ϵ	T'→ϵ
F	F→id	Error	Error	F→(E)	Error	Error

Example-3: LL(1) parsing

Step 4: Parse the string : id + id * id $

STACK	INPUT	OUTPUT
E$	id+id*id$	

NT	Input Symbol					
	id	+	*	(	)	$
E	E→TE'	Error	Error	E→TE'	Error	Error
E'	Error	E'→+TE'	Error	Error	E'→ϵ	E'→ϵ
T	T→FT'	Error	Error	T→FT'	Error	Error
T'	Error	T'→ϵ	T'→*FT'	Error	T'→ϵ	T'→ϵ
F	F→id	Error	Error	F→(E)	Error	Error