Parsing

Part III

Top Down Parsing

- A TDP tries to create a parse tree from the root towards the leafs scanning the input from left to right
- Find a left-most derivation
- Find (build) a parse tree
- Start building from the root and work down…
- As we search for a derivation
  - Must make choices:
    - Which rule to use
    - Where to use it
- May run into problems!!

String: id + id * id

$E \rightarrow TE'$

$E' \rightarrow +TE' \mid \varepsilon$

$T \rightarrow FT'$

$T' \rightarrow *FT' \mid \varepsilon$

$F \rightarrow (E) \mid \textbf{id}$

$E \rightarrow TE'$

$\rightarrow T\ E'$

$\rightarrow F\ T'\ E'$

$\rightarrow id + id * id$

Top-Down Parsing

- Recursive-Descent Parsing
  - Consists of a set of procedures, one for each non-terminal
  - Execution begins with the procedure for start symbol and ***descents*** from the start symbol to the final symbol (This is the reason it is called Recursive-Descent Parsing and procedures are **recursive** in nature)
    - Announces success if the procedure body scans the entire input
  - Backtracking is needed (If a choice of a production rule does not work, we backtrack to try other alternatives.)
  - It is a general parsing technique, but not widely used
  - Not efficient

Recursive-Descent Parsing Algorithm

```
void A(){
    for (j=1 to t){ /* assume there is t number of A-productions */
        Choose a A-production, Aj → X1X2…Xk;
        for (i=1 to k){
            if (Xi is a non-terminal)
                call procedure Xi();
            else if (Xi equals the current input symbol a)
                advance the input to the next symbol;
            else backtrack in input and reset the pointer
        }
    }
}
```

$A \rightarrow X_1X_2\ldots X_k \mid Y_1Y_2\ldots Y_k \mid \ldots\ \ldots$

Recursive-Descent Parsing (Cont.)

- Requires **backtracking** so the algorithm need to be modified
- Choosing an appropriate production is not easy as **we need to try all the alternatives**
- If a production fails, the input pointer needs to be reset and another alternate production is tried
- Not suitable for Left-Recursive grammars

Recursive Descent Parsing (Backtracking)

Input: aabbde

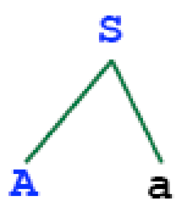
↑

S

1. S → Aa
2. → Ce
3. A → aaB
4. → aaba
5. B → bbb
6. C → aaD
7. D → bbd

Recursive Descent Parsing (Backtracking)

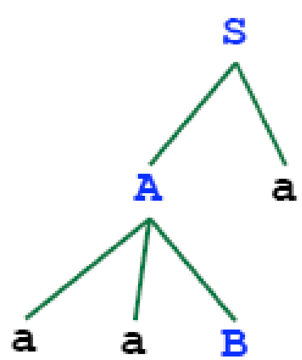
Input: aabbde



1. S → Aa
2. → Ce
3. A → aaB
4. → aaba
5. B → bbb
6. C → aaD
7. D → bbd

Recursive Descent Parsing (Backtracking)

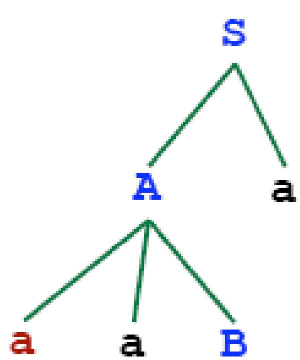
Input: aabbde



1. $S \rightarrow A$a
2. $\rightarrow C$e
3. $A \rightarrow$ aa$B$
4. $\rightarrow$ aaba
5. $B \rightarrow$ bbb
6. $C \rightarrow$ aa$D$
7. $D \rightarrow$ bbd

Recursive Descent Parsing (Backtracking)

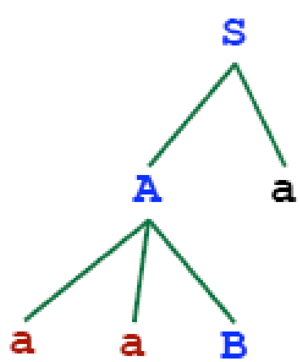
Input: aabbde



1. S → Aa
2. → Ce
3. A → aaB
4. → aaba
5. B → bbb
6. C → aaD
7. D → bbd

Recursive Descent Parsing (Backtracking)

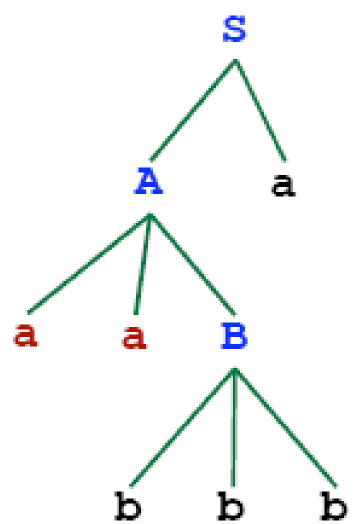
Input: aabbde



1. S → Aa
2.   → Ce
3. A → aaB
4.   → aaba
5. B → bbb
6. C → aaD
7. D → bbd

Recursive Descent Parsing (Backtracking)

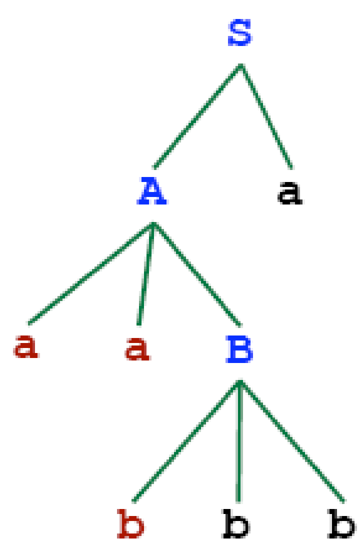
Input: `aabbde`



1. S → Aa
2. → Ce
3. A → aaB
4. → aaba
5. B → bbb
6. C → aaD
7. D → bbd

Recursive Descent Parsing (Backtracking)

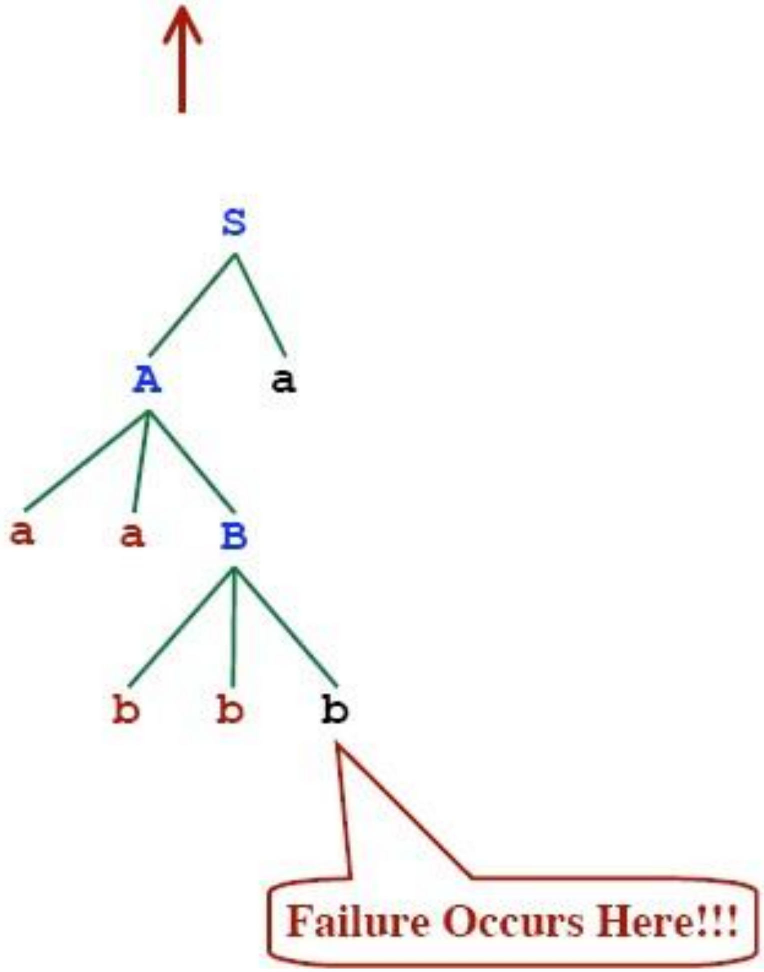
Input: aabbde



1.	S	→	Aa
2.		→	Ce
3.	A	→	aaB
4.		→	aaba
5.	B	→	bbb
6.	C	→	aaD
7.	D	→	bbd

Recursive Descent Parsing (Backtracking)

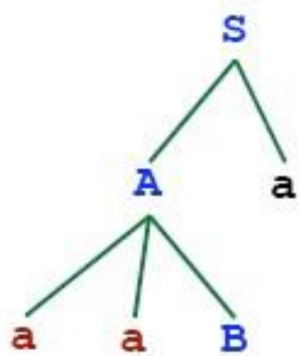
Input: **aabbde**



1. S → Aa
2. → Ce
3. A → aaB
4. → aaba
5. B → bbb
6. C → aaD
7. D → bbd

Recursive Descent Parsing (Backtracking)

Input: aabbde

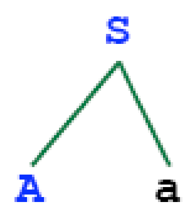


1. S → Aa
2. → Ce
3. A → aaB
4. → aaba
5. B → bbb
6. C → aaD
7. D → bbd

We need an ability to back up in the input!!!

Recursive Descent Parsing (Backtracking)

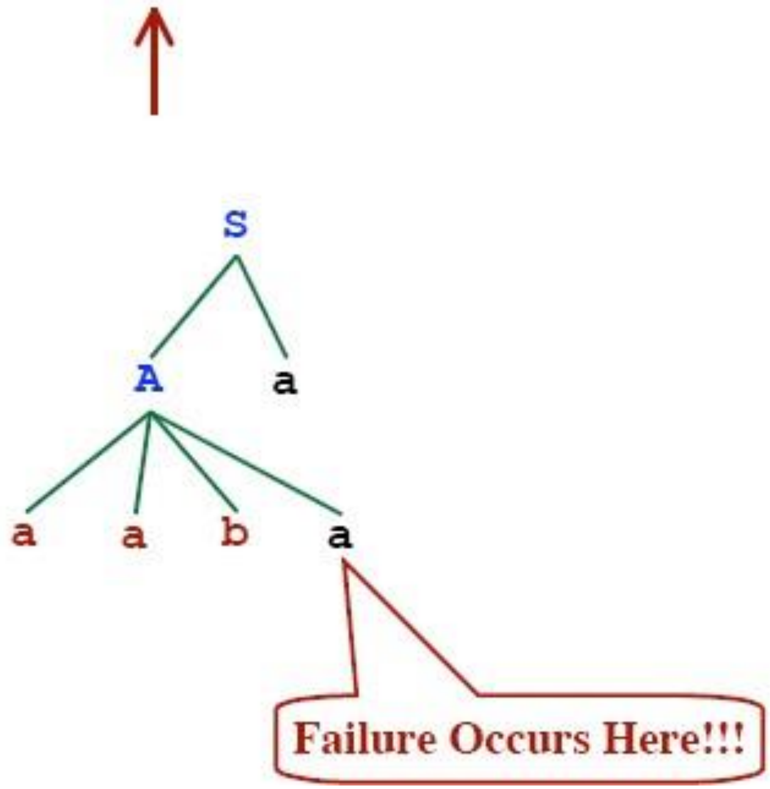
Input: aabbde



1. S → Aa
2. → Ce
3. A → aaB
4. → aaba
5. B → bbb
6. C → aaD
7. D → bbd

Recursive Descent Parsing (Backtracking)

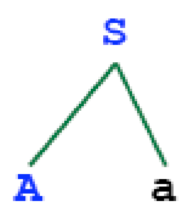
Input: `aabbde`



1. S → A`a`
2.   → C`e`
3. A → `aa`B
4.   → `aaba`
5. B → `bbb`
6. C → `aa`D
7. D → `bbd`

Recursive Descent Parsing (Backtracking)

Input: `aabbde`



1. S $\rightarrow$ Aa
2. $\rightarrow$ Ce
3. A $\rightarrow$ aaB
4. $\rightarrow$ aaba
5. B $\rightarrow$ bbb
6. C $\rightarrow$ aaD
7. D $\rightarrow$ bbd

Recursive Descent Parsing (Backtracking)

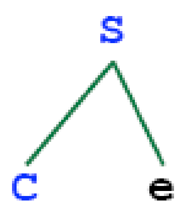
Input: aabbde

S

1. S → Aa
2. → Ce
3. A → aaB
4. → aaba
5. B → bbb
6. C → aaD
7. D → bbd

Recursive Descent Parsing (Backtracking)

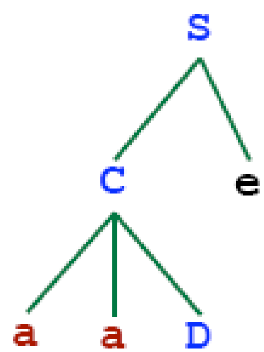
Input: aabbde



1. S → Aa
2. → Ce
3. A → aaB
4. → aaba
5. B → bbb
6. C → aaD
7. D → bbd

Recursive Descent Parsing (Backtracking)

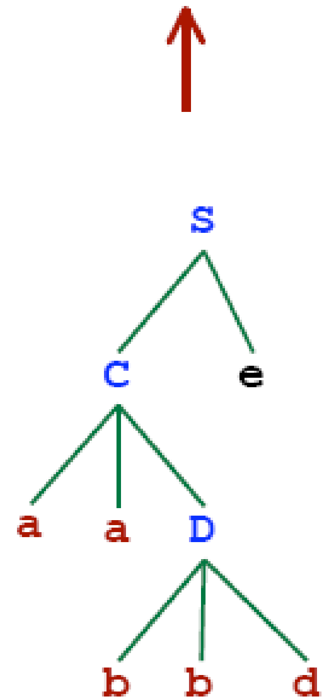
Input: `aabbde`



1. $S \rightarrow A$`a`
2. $\quad\ \rightarrow C$`e`
3. $A \rightarrow$ `aa`B
4. $\quad\ \rightarrow$ `aaba`
5. $B \rightarrow$ `bbb`
6. $C \rightarrow$ `aa`D
7. $D \rightarrow$ `bbd`

Recursive Descent Parsing (Backtracking)

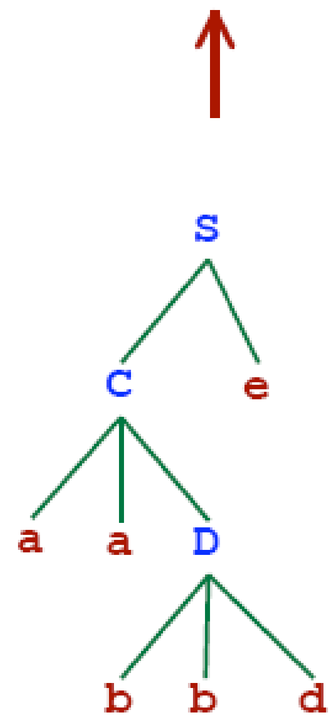
Input: `aabbde`



1. $S \rightarrow A$ `a`
2. $\quad \rightarrow C$ `e`
3. $A \rightarrow$ `aa`B
4. $\quad \rightarrow$ `aaba`
5. $B \rightarrow$ `bbb`
6. $C \rightarrow$ `aa`D
7. $D \rightarrow$ `bbd`

Recursive Descent Parsing (Backtracking)

Input: **aabbde**



1. S → Aa
2. → Ce
3. A → aaB
4. → aaba
5. B → bbb
6. C → aaD
7. D → bbd

Successfully parsed!!

Another example

$S \rightarrow cAd$
$A \rightarrow ab|a$

Input: cad

Predictive Parser

- no backtracking
- efficient
- needs a special form of grammars (LL(1) grammars).
- Recursive Predictive Parsing is a special form of Recursive Descent parsing without backtracking.
- Non-Recursive (Table Driven) Predictive Parser is also known as **LL(1)** parser.

When re-writing a non-terminal in a derivation step, a predictive parser can uniquely choose a production rule by just looking the current symbol in the input string.

$A \rightarrow \alpha_1 \mid \ldots \mid \alpha_n$ input: ... a ....

current token

Predictive Parsing Procedure

- Make a transition diagram (like NFA/DFA) for every rule of the grammar
- Optimize the DFA by reducing the number of states, yielding the final transition diagram
- To parse a string, simulate the string on the transition diagram
- If after consuming the input the transition diagram reaches an accept state, it is parsed

Simulation Method

- Start from the start state
- If a terminal comes and consume it, move to the next state
- If a non-terminal comes, go to the state of the DFA of the non-terminal and return on reaching the final state
- Return to the original DFA and continue parsing
- If on completion (completely reading the input string), you reach a final state, string is successfully parsed

Disadvantage

- It is a recursive parser, so it will consume a lot of memory as the stack grows
- To remove this recursion, we use LL parser, which uses a table for lookup

Predictive Parser (example)

```
stmt → if ......      |
       while ......   |
       begin ......   |
       for .....
```

- When we are trying to write the non-terminal *stmt*, if the current token is `if` we have to choose first production rule.
- When we are trying to write the non-terminal *stmt*, we can uniquely choose the production rule by just looking the current token.
- We eliminate the left recursion in the grammar, and left factor it. But it may not be suitable for predictive parsing (not LL(1) grammar).

Recursive Predictive Parsing

- Each non-terminal corresponds to a procedure.

Ex: $A \rightarrow aBb$ (This is only the production rule for A)

proc A {

- match the current token with a, and move to the next token;
- call 'B';
- match the current token with b, and move to the next token;

}

Recursive Predictive Parsing (cont.)

$A \rightarrow aBb \quad | \quad bAB$

```
proc A {
    case of the current token {
        'a':  - match the current token with a, and move to the next token;
              - call 'B';
              - match the current token with b, and move to the next token;

        'b':  - match the current token with b, and move to the next token;
              - call 'A';
              - call 'B';
    }
}
```

Recursive Predictive Parsing (cont.)

- When to apply ε-productions.

  $A \rightarrow aA \mid bB \mid \varepsilon$

- If all other productions fail, we should apply an ε-production. For example, if the current token is not a or b, we may apply the ε-production.
- Most correct choice: We should apply an ε-production for a non-terminal A when the current token is in the follow set of A (which terminals can follow A in the sentential forms).

Recursive Predictive Parsing (Example)

$A \rightarrow aBe \mid cBd \mid C$

$B \rightarrow bB \mid \varepsilon$

$C \rightarrow f$

proc C { match the current token with f,
and move to the next token; }

proc A {
case of the current token {
a: - match the current token with a, and move to the next token;
- call B;
- match the current token with e, and move to the next token;
c: - match the current token with c, and move to the next token;
- call B;
- match the current token with d, and move to the next token;
f: - call C
}
}

first set of C

proc B {
case of the current token {
b: - match the current token with b, and move to the next token;
- call B
e,d: do nothing
}
}

follow set of B

First Function

Let α be a string of symbols (terminals and nonterminals)

Define:

FIRST (α) = The set of terminals that could occur first in any string derivable from α

= $\{ a \mid \alpha \Rightarrow^* aw$, plus ε if $\alpha \Rightarrow^* \varepsilon \}$

Example:

$$\boxed{\begin{array}{l} E \rightarrow T\ E' \\ E' \rightarrow +\ T\ E' \mid \varepsilon \\ T \rightarrow F\ T' \\ T' \rightarrow *\ F\ T' \mid \varepsilon \\ F \rightarrow (\ E\ ) \mid \underline{\text{id}} \end{array}}$$

FIRST (F) = { (, id }

FIRST (T') = { *, ε }

FIRST (T) = { (, id }

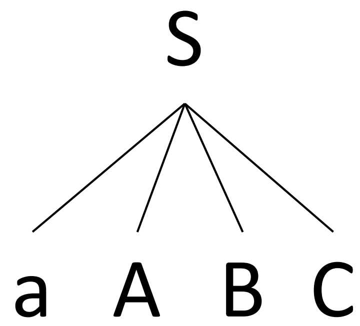
FIRST (E') = { +, ε }

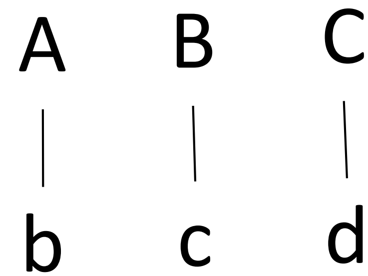
FIRST (E) = { (, id }

Example of FIRST

S → aABC
A → b
B → c
C → d

FIRST(S):
FIRST(A):
FIRST(B):
FIRST(C):

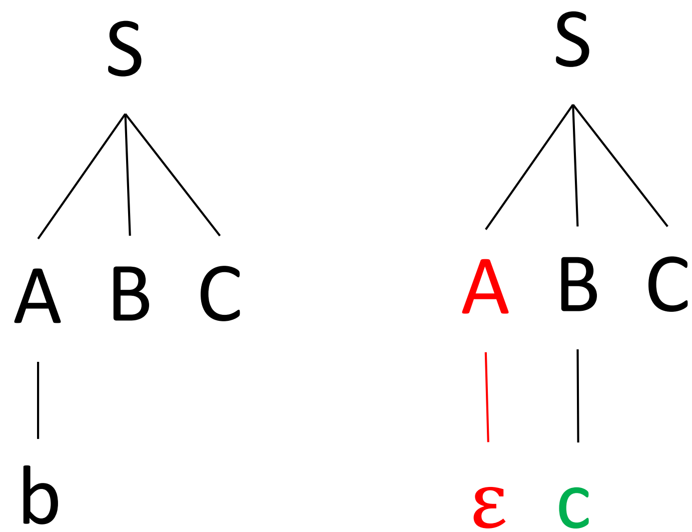




Example of FIRST

S → ABC
A → b | ε
B → c
C → d
D → e

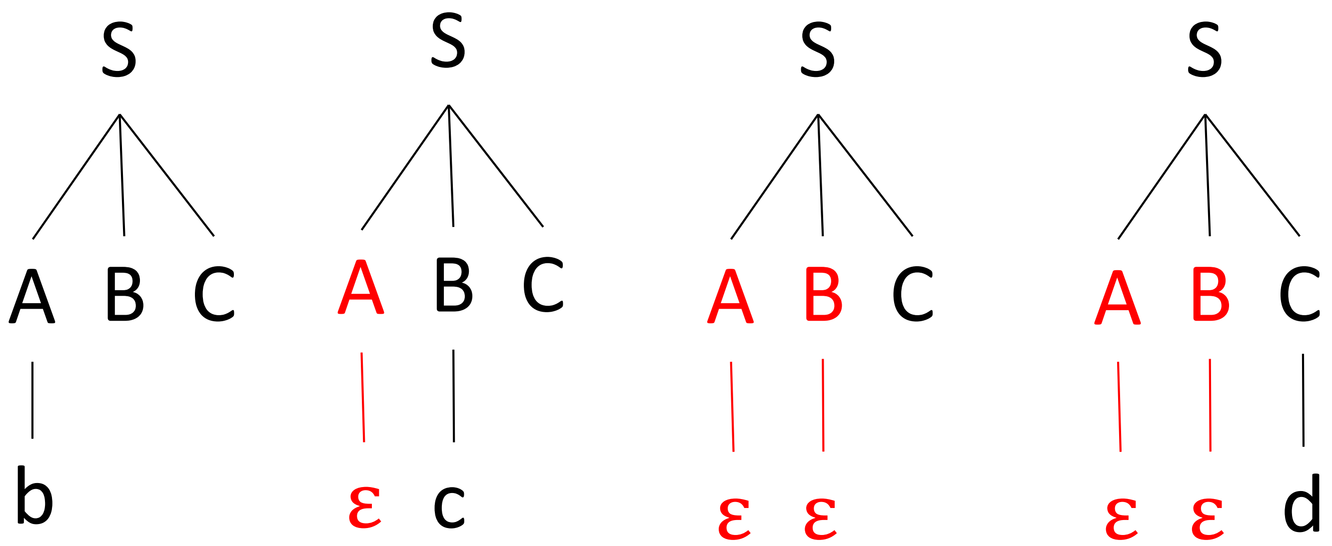
FIRST(S):
FIRST(A):
FIRST(B):
FIRST(C):



Example of FIRST

S → ABC
A → b | ε
B → c | ε
C → d

FIRST(S):
FIRST(A):
FIRST(B):
FIRST(C):



Computing the First Function

For all symbols X in the grammar...

```
if X is a terminal then
   FIRST(X) = { X }

if X → ε is a rule then
   add ε to FIRST(X)

if X → Y1 Y2 Y3 ... YK is a rule then
   if a ∈ FIRST(Y1) then
      add a to FIRST(X)
   if ε ∈ FIRST(Y1) and a ∈ FIRST(Y2) then
      add a to FIRST(X)
   if ε ∈ FIRST(Y1) and ε ∈ FIRST(Y2) and a ∈ FIRST(Y3) then
      add a to FIRST(X)
   ...
   if ε ∈ FIRST(Yi) for all Yi then
      add ε to FIRST(X)
```

Repeat until nothing more can be added to any sets.

To Compute the FIRST($X1X2X3...XN$)

```
Result = {}
Add everything in FIRST(X1), except ε, to result
if ε ∈ FIRST(X1) then
  Add everything in FIRST(X2), except ε, to result
  if ε ∈ FIRST(X2) then
    Add everything in FIRST(X3), except ε, to result
    if ε ∈ FIRST(X3) then
      Add everything in FIRST(X4), except ε, to result
      ...
        if ε ∈ FIRST(XN-1) then
          Add everything in FIRST(XN), except ε, to result
          if ε ∈ FIRST(XN) then
            // Then X1 ⇒* ε, X2 ⇒* ε, X3 ⇒* ε, ... XN ⇒* ε
            Add ε to result
          endIf
        endIf
      ...
    endIf
  endIf
endIf
```

To Compute FOLLOW Function

- What is the terminal which can follow a variable (non-terminal) in the process of variable
- A derivation always starts from \$. Because the input is followed by \$. So,

$$S\$$$

1. FOLLOW(S) = {\$}, S is the start symbol
2. If A → αBβ, where α and β are any string of terminals and non-terminals then,

   FOLLOW(B) = FIRST(β) except ε
3. If A → αB or A → αBβ where FIRST(β) contains ε (β=> ε) then

FOLLOW(B) = FOLLOW(A)

NOTE: \$ is ***always present*** in starting symbol of FOLLOW set

NOTE: ε is ***never present*** in FOLLOW sets

Example 1

$S \rightarrow ABCD$
$A \rightarrow b \mid \varepsilon$
$B \rightarrow c \mid \varepsilon$
$C \rightarrow d$
$D \rightarrow e$

FIRST(S): {b,c,d }
FIRST(A): { b, ε}
FIRST(B): {c, ε }
FIRST(C): {d}
FIRST(D): {e}

FOLLOW(S):
FOLLOW(A):
FOLLOW(B):
FOLLOW(C):
FOLLOW(D):

Example 1

$S \rightarrow ABCD$
$A \rightarrow b \mid \varepsilon$
$B \rightarrow c \mid \varepsilon$
$C \rightarrow d$
$D \rightarrow e$

FIRST(S): {b,c,d }
FIRST(A): { b, ε}
FIRST(B): {c, ε }
FIRST(C): {d}
FIRST(D): {e}

FOLLOW(S): {$}
FOLLOW(A): {c}
FOLLOW(B): {d}
FOLLOW(C): {e}
FOLLOW(D): {$}

$ is ***always present*** in starting symbol of FOLLOW set

ε is ***never present*** in FOLLOW sets

FOLLOW example

PRODUCTION	FIRST	FOLLOW
S→ABCDE A→a|ε B→b|ε C→c D→d|ε E→e|ε	FIRST(S)= FIRST(A)= FIRST(B)= FIRST(C)= FIRST(D)= FIRST(E)=	FOLLOW(S)= FOLLOW(A)= FOLLOW(B)= FOLLOW(C)= FOLLOW(D)= FOLLOW(E)=

1. FOLLOW(S) = {$}, S is the start symbol
2. If A → αBβ, where α and β are any string of terminals and non-terminals then,

   FOLLOW(B) = FIRST(β) except ε
3. If A → αB or A → αBβ where FIRST(β) contains ε (β=> ε) then

FOLLOW(B) = FOLLOW(A)

NOTE: $ is ***always present*** in starting symbol of FOLLOW set

NOTE: ε is ***never present*** in FOLLOW sets

FOLLOW example

PRODUCTION	FIRST	FOLLOW
S→ABCDE	FIRST(S) = {a, b, c}	FOLLOW(S) = {$}
A→a|ε	FIRST(A) = {a, ε}	FOLLOW(A) = {b, c}
B→b|ε	FIRST(B) = {b, ε}	FOLLOW(B) = {c}
C→c	FIRST(C) = {c}	FOLLOW(C) = {d, e, $}
D→d|ε	FIRST(D) = {d, ε}	FOLLOW(D) = {e, $}
E→e|ε	FIRST(E) = {e, ε}	FOLLOW(E) = {$}

FOLLOW example

PRODUCTION	FIRST	FOLLOW
$S \rightarrow B\mathbf{b} \mid C\mathbf{d}$ $B \rightarrow \mathbf{a}B \mid \varepsilon$ $C \rightarrow \mathbf{c}C \mid \varepsilon$	FIRST(S) = FIRST(B) = FIRST(C) =	FOLLOW(S) = FOLLOW(B) = FOLLOW(C) =

1. FOLLOW(S) = {$}, S is the start symbol
2. If $A \rightarrow \alpha B \beta$, where α and β are any string of terminals and non-terminals then,

   FOLLOW(B) = FIRST(β) except ε
3. If $A \rightarrow \alpha B$ or $A \rightarrow \alpha B \beta$ where FIRST(β) contains ε (β=> ε) then

FOLLOW(B) = FOLLOW(A)

NOTE: $ is ***always present*** in starting symbol of FOLLOW set

NOTE: ε is ***never present*** in FOLLOW sets

FOLLOW example

PRODUCTION	FIRST	FOLLOW
$S \rightarrow B\mathbf{b} \mid C\mathbf{d}$ $B \rightarrow \mathbf{a}B \mid \boldsymbol{\varepsilon}$ $C \rightarrow \mathbf{c}C \mid \boldsymbol{\varepsilon}$	FIRST(S) = {a,b,c,d} FIRST(B) = {a, ε} FIRST(C) = {c, ε}	FOLLOW(S) = {$} FOLLOW(B) = {b} FOLLOW(C) = {d}

FOLLOW example

One of the commonly used grammar!

PRODUCTION	FIRST	FOLLOW
E→ TE’ E’ → +TE’ | ε T→ FT’ T’→ * FT’ | ε F→ id | (E)	FIRST(E) = FIRST(E’) = FIRST(T) = FIRST(T’) = FIRST(F) =	FOLLOW(E) = FOLLOW(E’) = FOLLOW(T) = FOLLOW(T’) = FOLLOW(F) =

Do not write ε in Follow. Substitute it and check if it is the end or not.

FOLLOW example

One of the commonly used grammar!

PRODUCTION	FIRST	FOLLOW
E→TE' E' → +TE' | ε T→FT' T'→*FT' | ε F→id | (E)	FIRST(E) = { id,(} FIRST(E') = { +, ε} FIRST(T) = { id, (} FIRST(T') = { *, ε } FIRST(F) = { id, (}	FOLLOW(E) = { $,) } FOLLOW(E') = { $,) } FOLLOW(T) = { +, $,) } FOLLOW(T') = { +, $,) } FOLLOW(F) = { *, +, $,) }

FOLLOW example

PRODUCTION	FIRST	FOLLOW
S→ACB | CbB | Ba A→da | BC B→g | ε C→h | ε	FIRST(S)= FIRST(A)= FIRST(B)= FIRST(C)=	FOLLOW(S)= FOLLOW(A)= FOLLOW(B)= FOLLOW(C)=

FOLLOW example

PRODUCTION	FIRST	FOLLOW
S→ACB| CbB| Ba A→da| BC B→g| ε C→h| ε	FIRST(S)={d,g,h,ε,b,a} FIRST(A)={d,g,h,ε} FIRST(B)={g,ε} FIRST(C)={h,ε}	FOLLOW(S)={$} FOLLOW(A)={h,g,$} FOLLOW(B)={a,$,h,g} FOLLOW(C)={g,$,b,h}

In A, while generating the FIRST, we get completely ε.
As BC completely goes to ε.

Predictive Parsing

Will never backtrack!

Requirement:

For every rule:

$$A \rightarrow \alpha_1 \mid \alpha_2 \mid \alpha_3 \mid \ldots \mid \alpha_N$$

We must be able to choose the correct alternative by looking only at the next symbol

May peek ahead to the next symbol (token).

Example

$A \rightarrow aB$
$\rightarrow cD$
$\rightarrow E$

Assuming a,c $\notin$ FIRST (E)

Example

Stmt → `if` Expr ...
→ `for` LValue ...
→ `while` Expr ...
→ `return` Expr ...
→ ID ...

Predictive Parsing

- **LL(1) Grammars**
  - Can do predictive parsing
  - Can select the right rule
  - Looking at only the next 1 input symbol
    - First L : Left to Right Scanning
    - Second L: Leftmost derivation
    - 1 : one input symbol look-ahead for predictive decision

- **LL(k) Grammars**
  - Can do predictive parsing
  - Can select the right rule
  - Looking at only the next k input symbols

- **Techniques to modify the grammar:**
  - Left Factoring
  - Removal of Left Recursion

- **LL(k) Language**
  - Can be described with an LL(k) grammar

$E \rightarrow E+T|T$
$T \rightarrow T*F|F$
$F \rightarrow (E)|id$

Assume that the grammar is LL(1)

i.e., Backtracking will never be needed

Always know which righthand side to choose (with one look-ahead)

- No Left Recursion
- Grammar is Left-Factored.

Table Driven Predictive Parsing

After Eliminating Left Recursion:

Example

$E \rightarrow T\ E'$ Term ...

$E' \rightarrow +\ T\ E' \mid \varepsilon$ +Term +Term + ... +Term

$T \rightarrow F\ T'$ Factor ...

$T' \rightarrow *\ F\ T' \mid \varepsilon$ * Factor * Factor * ... * Factor

$F \rightarrow (\ E\ ) \mid \underline{\texttt{id}}$

Step 1: From grammar, construct table.

Step 2: Use table to parse strings.

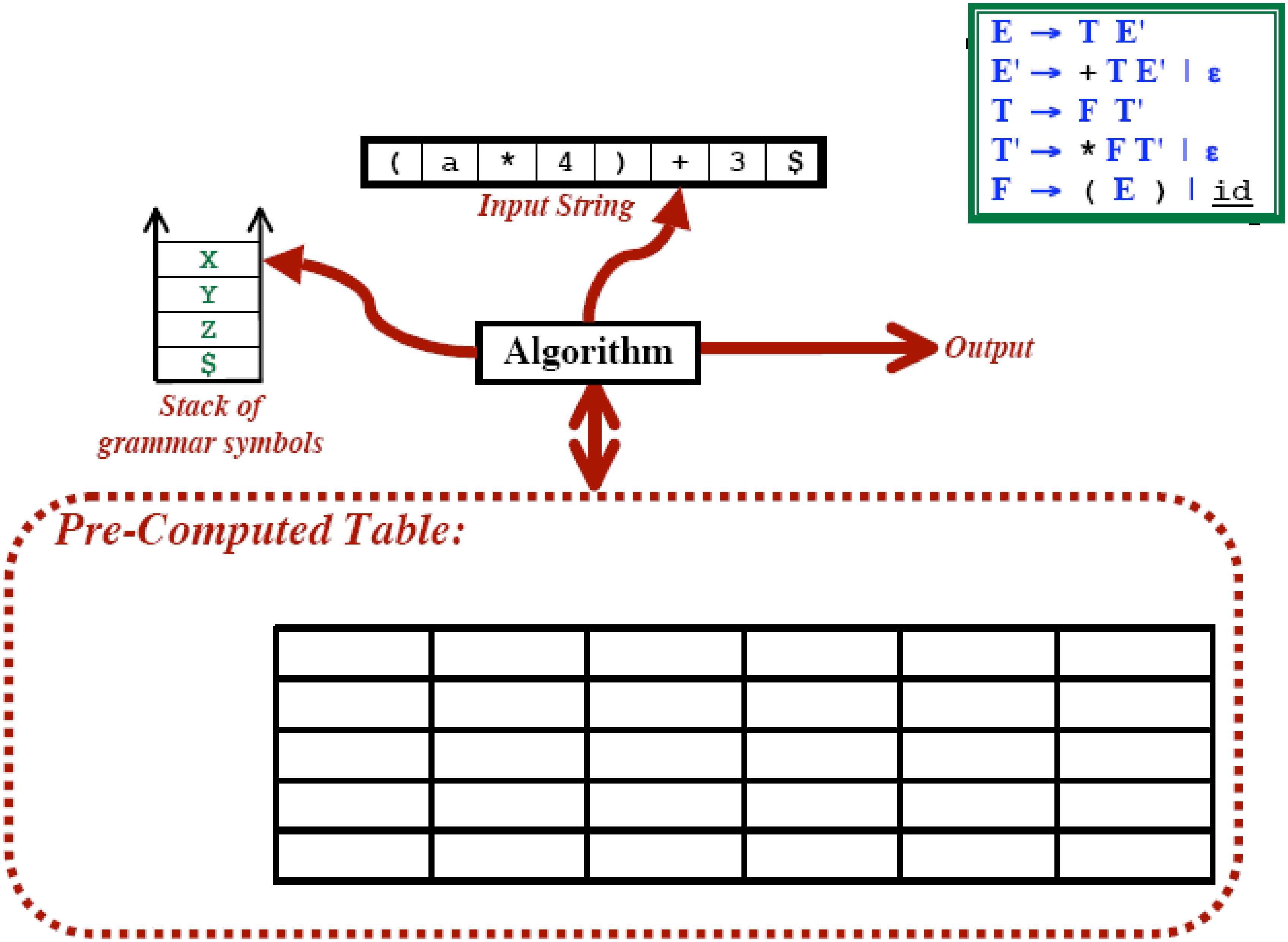
FIRST & FOLLOW

$E \rightarrow T\ E'$
$E' \rightarrow +\ T\ E' \mid \varepsilon$
$T \rightarrow F\ T'$
$T' \rightarrow *\ F\ T' \mid \varepsilon$
$F \rightarrow (\ E\ ) \mid \underline{\text{id}}$

FIRST(E) = FIRST(T) = FIRST(F) = {(, id}
FIRST(E') = {+, ε}
FIRST(T') = {*, ε}

FOLLOW(E) = {$,)}
FOLLOW(E') = FOLLOW(E) = {$,)}
FOLLOW(T) = FIRST(E') = {+, $,)} [As FIRST(E') contains ε we are using FOLLOW(E)]
FOLLOW(T') = FOLLOW(T) = { +, $,)}
FOLLOW(F) = FIRST(T') = {*, +, $,)} [FIRST(T') contains ε so using FOLLOW(T)]

Table Driven Predictive Parsing



Rules for Table Driven Predictive Parsing

For each production **A → α** (A tends to alpha):

1.Find First(α) and for each terminal in First(α), make entry A → α in the table.

2.If First(α) contains ε (→ ε in the table. epsilon) as terminal, then find the Follow(A) and for each terminal in Follow(A), make entry A

3.If the First(α) contains ε and Follow(A) contains $ as terminal, then make entry A → ε in the table for the $.

In the table, rows will contain the Non-Terminals and the column will contain the Terminal Symbols. All the **Null Productions** of the Grammars will go under the Follow elements and the remaining productions will lie under the elements of the First set.

Table Driven Predictive Parsing

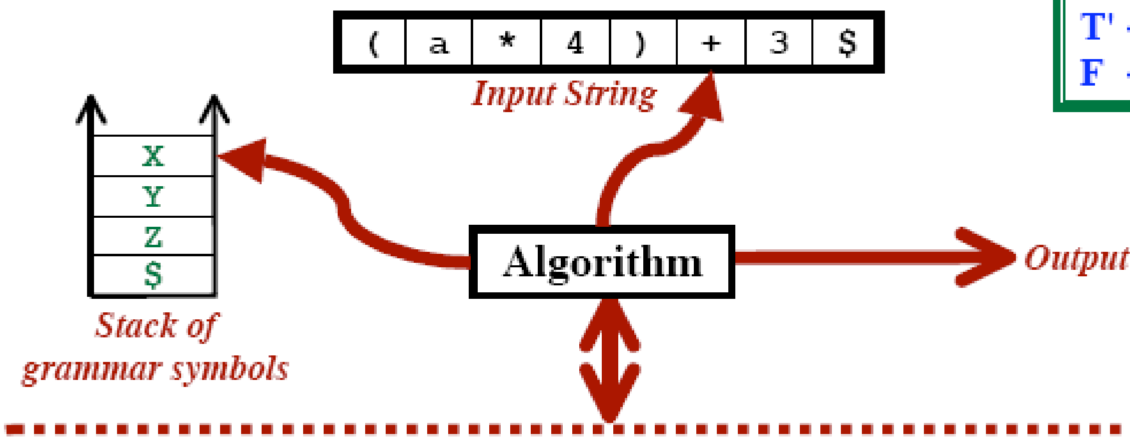
$E \rightarrow T\,E'$
$E' \rightarrow +\,T\,E' \mid \varepsilon$
$T \rightarrow F\,T'$
$T' \rightarrow *\,F\,T' \mid \varepsilon$
$F \rightarrow (\,E\,) \mid \underline{id}$

Non-Terminal	FIRST	FOLLOW
E	{(, id}	{), $}
E'	{+, ε}	{), $}
T	{(, id}	{+, $,)}
T'	{*, ε}	{ +, $,)}
F	{(, id}	{*, +, $,)}

NT ↓ | T →	+	*	(	)	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)		F→id	

Table Driven Predictive Parsing

$E \rightarrow T\ E'$
$E' \rightarrow +\ T\ E' \mid \varepsilon$
$T \rightarrow F\ T'$
$T' \rightarrow *\ F\ T' \mid \varepsilon$
$F \rightarrow (\ E\ ) \mid \underline{id}$



Pre-Computed Table:

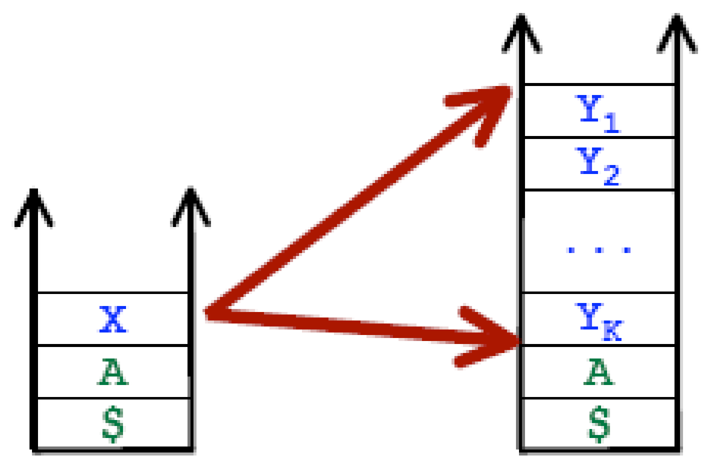
Input Symbols, plus $

Nonterminals

	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)		

Predictive Parsing Algorithm

```
Set input ptr to first symbol; Place $ after last input symbol
Push $
Push S
repeat
  X = stack top
  a = current input symbol
  if X is a terminal or X = $ then
    if X == a then
      Pop stack
      Advance input ptr
    else
      Error
    endIf
  elseIf Table[X,a] contains a rule then    // call it X → Y1 Y2 ... YK
    Pop stack
    Push YK
    ...
    Push Y2
    Push Y1
    Print ("X → Y1 Y2 ... YK")
  else    // Table[X,a] is blank
    Syntax Error
  endIf
until X == $
```



Predictive Parsing

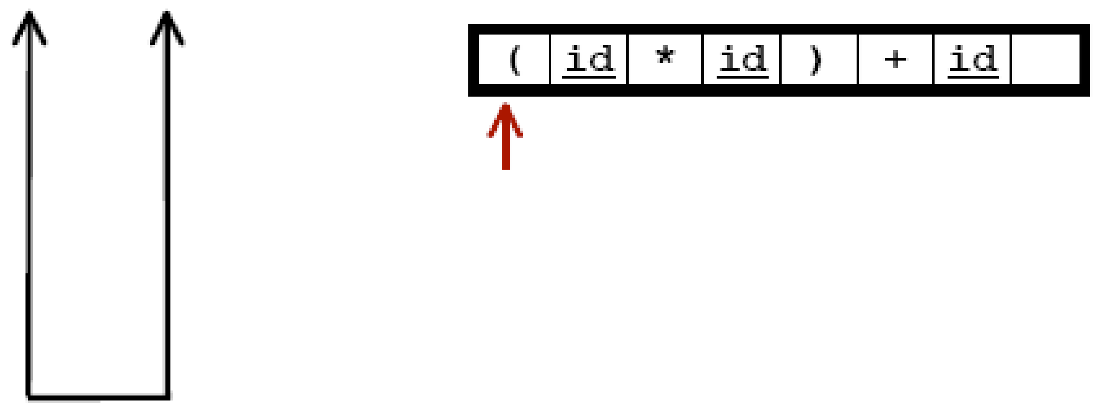
Example

Input:

(id*id)+id

Output:

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



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

Predictive Parsing

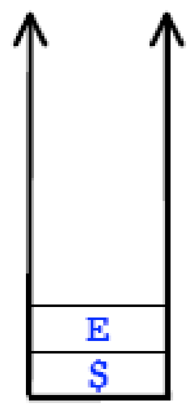
Example

Input:

(id*id)+id

Output:

$E \rightarrow T\ E'$
$E' \rightarrow +\ T\ E' \mid \varepsilon$
$T \rightarrow F\ T'$
$T' \rightarrow *\ F\ T' \mid \varepsilon$
$F \rightarrow (\ E\ ) \mid \underline{id}$



(	id	*	id	)	+	id	$

Add $ to end of input
Push $
Push E

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

Predictive Parsing

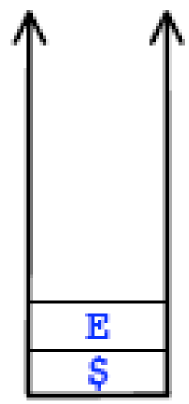
Example

$E \rightarrow T\ E'$
$E' \rightarrow +\ T\ E'\ |\ \varepsilon$
$T \rightarrow F\ T'$
$T' \rightarrow *\ F\ T'\ |\ \varepsilon$
$F \rightarrow (\ E\ )\ |\ \underline{id}$

Input:

(id*id)+id

Output:



(	id	*	id	)	+	id	$

Look at Table [E, '(']
Use rule E→TE'
Pop E
Push E'
Push T
Print E→TE'

	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)		

Predictive Parsing

Example

Input:

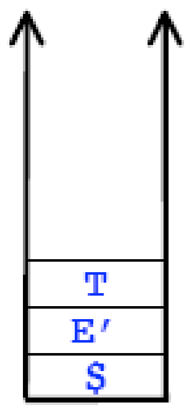
(id*id)+id

Output:

E → T E'

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

(	id	*	id	)	+	id	$



Look at Table [E, '(']
Use rule E→TE'
Pop E
Push E'
Push T
Print E→TE'

	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)		

Predictive Parsing

Example

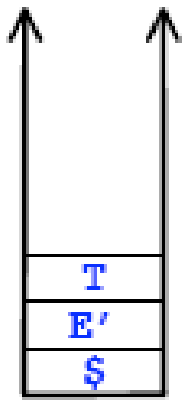
Input:

(id*id)+id

Output:

E → T E'

$E \rightarrow T\ E'$
$E' \rightarrow +\ T\ E' \mid \varepsilon$
$T \rightarrow F\ T'$
$T' \rightarrow *\ F\ T' \mid \varepsilon$
$F \rightarrow (\ E\ ) \mid \underline{id}$



(	id	*	id	)	+	id	$

Table [T, '('] = T→FT'
Pop T
Push T'
Push F
Print T→FT'

	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)		

Predictive Parsing

Example

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

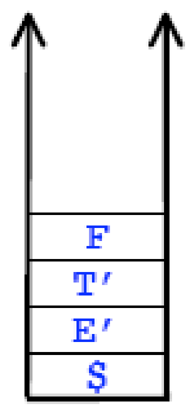
Input:

(id*id)+id

Output:

E → T E'

T → F T'



(	id	*	id	)	+	id	$

Table [T, '('] = T→FT'

Pop T

Push T'

Push F

Print T→FT'

	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)		

Predictive Parsing

Example

Input:

(id*id)+id

Output:

E → T E'
T → F T'

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

(	id	*	id	)	+	id	$

Stack:

F
T'
E'
$

Table [F, '('] = F→(E)
Pop F
Push (
Push E
Push)
Print F→(E)

	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)		

Predictive Parsing

Example

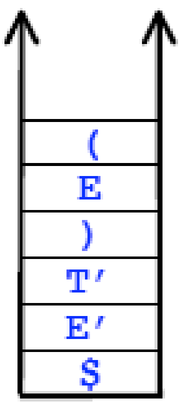
E	→	T E'		
E'	→	+ T E'	|	ε
T	→	F T'		
T'	→	* F T'	|	ε
F	→	(E)	|	id

Input:

(id*id)+id

Output:

E → T E'
T → F T'
F → (E)



(	id	*	id	)	+	id	$

Table [F, '('] = F→(E)
Pop F
Push)
Push E
Push (
Print F→(E)

	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)		

Predictive Parsing

Example

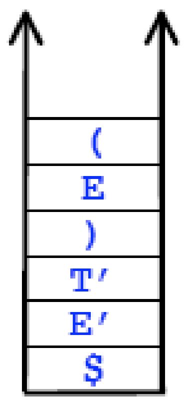
Input:

(id*id)+id

Output:

E → T E'
T → F T'
F → (E)

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



(	id	*	id	)	+	id	$

Top of Stack matches next input
Pop and Scan

	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)		

Predictive Parsing

Example

$E \rightarrow T\ E'$
$E' \rightarrow +\ T\ E' \mid \varepsilon$
$T \rightarrow F\ T'$
$T' \rightarrow *\ F\ T' \mid \varepsilon$
$F \rightarrow (\ E\ ) \mid \underline{id}$

Input:

(id*id)+id

Output:

$E \rightarrow T\ E'$
$T \rightarrow F\ T'$
$F \rightarrow (\ E\ )$

(	id	*	id	)	+	id	$

E
)
T'
E'
$

Table [E, id] = $E \rightarrow TE'$
Pop E
Push E'
Push T
Print $E \rightarrow TE'$

	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	$F \rightarrow \underline{id}$			$F \rightarrow (E)$		

Predictive Parsing

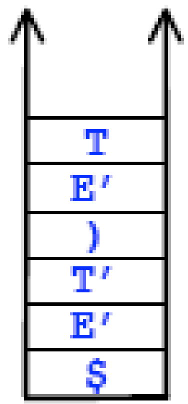
Example

Input:

(id*id)+id

Output:

E → T E'
T → F T'
F → (E)
E → T E'



(	id	*	id	)	+	id	$

Table [E, id] = E→TE'
Pop E
Push E'
Push T
Print E→TE'

E → T E'
E' → + T E' | ε
T → F T'
T' → * F T' | 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)		

Predictive Parsing

Example

Input:

`(id*id)+id`

Output:

$E \rightarrow T\ E'$
$T \rightarrow F\ T'$
$F \rightarrow (\ E\ )$
$E \rightarrow T\ E'$

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

Stack
T
E'
)
T'
E'
$

(	id	*	id	)	+	id	$

Table [T, id] = T→FT'
Pop T
Push T'
Push F
Print T→FT'

	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)		

Predictive Parsing

Example

Input:

(id*id)+id

Output:

E → T E'
T → F T'
F → (E)
E → T E'
T → F T'

F
T'
E'
)
T'
E'
$

(	id	*	id	)	+	id	$

Table [T, id] = T→FT'
Pop T
Push T'
Push F
Print T→FT'

E → T E'
E' → + T E' | ε
T → F T'
T' → * F T' | 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)		

Predictive Parsing

Example

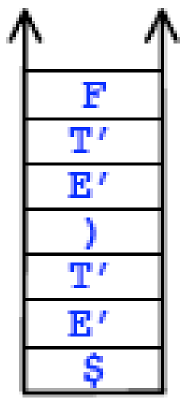
E → T E'
E' → + T E' | ε
T → F T'
T' → * F T' | ε
F → (E) | id

Input:

```
(id*id)+id
```

Output:

E → T E'
T → F T'
F → (E)
E → T E'
T → F T'



(	id	*	id	)	+	id	$

Table [F, id] = F→id
Pop F
Push id
Print F→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)		

Predictive Parsing

Example

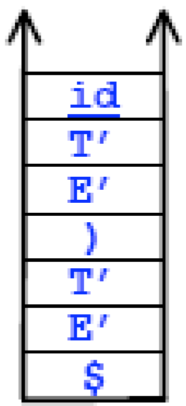
$E \rightarrow T\ E'$
$E' \rightarrow +\ T\ E' \mid \varepsilon$
$T \rightarrow F\ T'$
$T' \rightarrow *\ F\ T' \mid \varepsilon$
$F \rightarrow (\ E\ ) \mid id$

Input:

(id*id)+id

Output:

$E \rightarrow T\ E'$
$T \rightarrow F\ T'$
$F \rightarrow (\ E\ )$
$E \rightarrow T\ E'$
$T \rightarrow F\ T'$
$F \rightarrow id$



(	id	*	id	)	+	id	$

Table [F, id] = F→id
Pop F
Push id
Print F→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)		

Predictive Parsing

Example

Input:

(id*id)+id

Output:

E → T E'
T → F T'
F → (E)
E → T E'
T → F T'
F → id

id
T'
E'
)
T'
E'
$

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

(	id	*	id	)	+	id	$

Top of Stack matches next input
Pop and Scan

	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)		

Predictive Parsing

Example

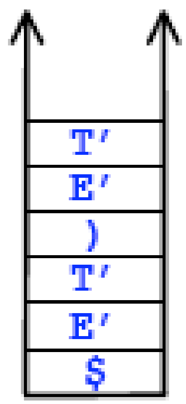
Input:

(id*id)+id

Output:

E → T E'
T → F T'
F → (E)
E → T E'
T → F T'
F → id

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



(	id	*	id	)	+	id	$

Table [T', ''] =* T'→*FT'
Pop T'
Push T'
Push F
Push ''*
Print T'→*FT'

	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)		

Predictive Parsing

Example

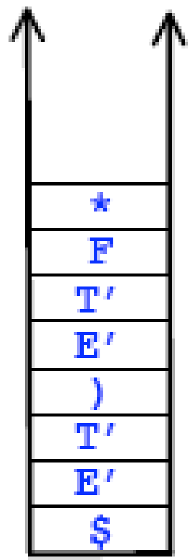
$E \rightarrow T\ E'$
$E' \rightarrow +\ T\ E' \mid \varepsilon$
$T \rightarrow F\ T'$
$T' \rightarrow *\ F\ T' \mid \varepsilon$
$F \rightarrow (\ E\ ) \mid \underline{id}$

Input:

(id*id)+id

Output:

$E \rightarrow T\ E'$
$T \rightarrow F\ T'$
$F \rightarrow (\ E\ )$
$E \rightarrow T\ E'$
$T \rightarrow F\ T'$
$F \rightarrow \underline{id}$
$T' \rightarrow *\ F\ T'$



(	id	*	id	)	+	id	$

Table [T', ''] =* $T' \rightarrow *FT'$
Pop T'
Push T'
Push F
Push ''*
Print $T' \rightarrow *FT'$

	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	$F \rightarrow \underline{id}$			$F \rightarrow (E)$		

Predictive Parsing

Example

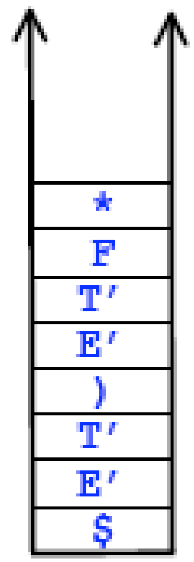
E → T E'
E' → + T E' | ε
T → F T'
T' → * F T' | ε
F → (E) | id

Input:

(id*id)+id

Output:

E → T E'
T → F T'
F → (E)
E → T E'
T → F T'
F → id
T' → * F T'



(	id	*	id	)	+	id	$

Top of Stack matches next input
Pop and Scan

	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)		

Predictive Parsing

Example

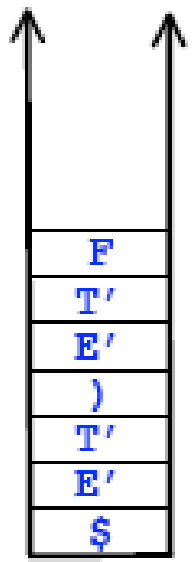
E → T E'
E' → + T E' | ε
T → F T'
T' → * F T' | ε
F → (E) | id

Input:

(id*id)+id

Output:

E → T E'
T → F T'
F → (E)
E → T E'
T → F T'
F → id
T' → * F T'



(	id	*	id	)	+	id	$

Top of Stack matches next input
Pop and Scan

	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)		

Predictive Parsing

Example

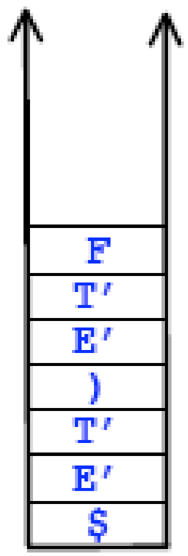
E	→	T E'		
E'	→	+ T E'	|	ε
T	→	F T'		
T'	→	* F T'	|	ε
F	→	(E)	|	id

Input:

(id*id)+id

Output:

E → T E'
T → F T'
F → (E)
E → T E'
T → F T'
F → id
T' → * F T'



(	id	*	id	)	+	id	$

Table [F, id] = F→id
Pop F
Push id
Print F→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)		

Predictive Parsing

Example

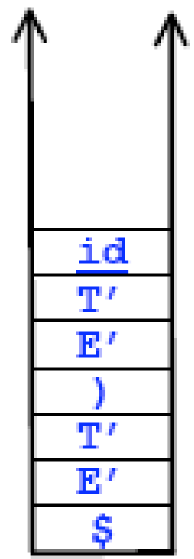
$E \rightarrow T\ E'$
$E' \rightarrow +\ T\ E' \mid \varepsilon$
$T \rightarrow F\ T'$
$T' \rightarrow *\ F\ T' \mid \varepsilon$
$F \rightarrow (\ E\ ) \mid id$

Input:

(id*id)+id

Output:

$E \rightarrow T\ E'$
$T \rightarrow F\ T'$
$F \rightarrow (\ E\ )$
$E \rightarrow T\ E'$
$T \rightarrow F\ T'$
$F \rightarrow id$
$T' \rightarrow *\ F\ T'$
$F \rightarrow id$



(	id	*	id	)	+	id	$

Table [F, id] = $F \rightarrow id$
Pop F
Push id
Print $F \rightarrow id$

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

Predictive Parsing

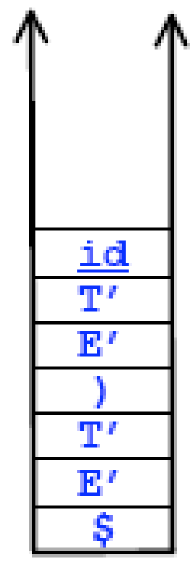
Example

Input:

(id*id)+id

Output:

E → T E'
T → F T'
F → (E)
E → T E'
T → F T'
F → id
T' → * F T'
F → id



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

(	id	*	id	)	+	id	$

Top of Stack matches next input
Pop and Scan

	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)		

Predictive Parsing

Example

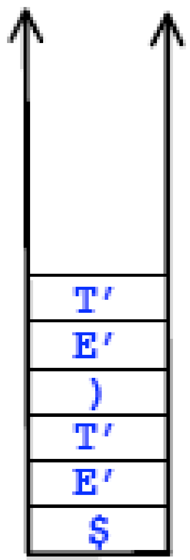
$E \rightarrow T\ E'$
$E' \rightarrow +\ T\ E' \mid \varepsilon$
$T \rightarrow F\ T'$
$T' \rightarrow *\ F\ T' \mid \varepsilon$
$F \rightarrow (\ E\ ) \mid \underline{\text{id}}$

Input:

(id*id)+id

Output:

$E \rightarrow T\ E'$
$T \rightarrow F\ T'$
$F \rightarrow (\ E\ )$
$E \rightarrow T\ E'$
$T \rightarrow F\ T'$
$F \rightarrow \underline{\text{id}}$
$T' \rightarrow *\ F\ T'$
$F \rightarrow \underline{\text{id}}$



(	id	*	id	)	+	id	$

Top of Stack matches next input
Pop and Scan

	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)		

Predictive Parsing

Example

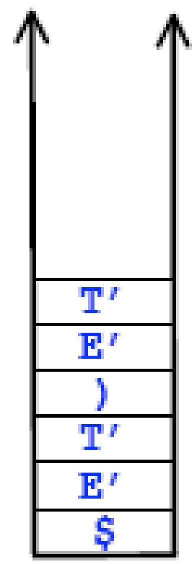
E → T E'
E' → + T E' | ε
T → F T'
T' → * F T' | ε
F → (E) | id

Input:

(id*id)+id

Output:

E → T E'
T → F T'
F → (E)
E → T E'
T → F T'
F → id
T' → * F T'
F → id



(	id	*	id	)	+	id	$

Table [T', ')'] = T'→ε
Pop T'
Push <nothing>
Print T'→ε

	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)		

Predictive Parsing

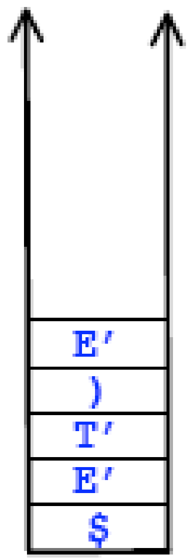
Example

Input:

(id*id)+id

Output:

E → T E'
T → F T'
F → (E)
E → T E'
T → F T'
F → id
T' → * F T'
F → id
T' → ε



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

(	id	*	id	)	+	id	$

Table [T', ')'] = T'→ε
Pop T'
Push <nothing>
Print T'→ε

	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)		

Predictive Parsing

Example

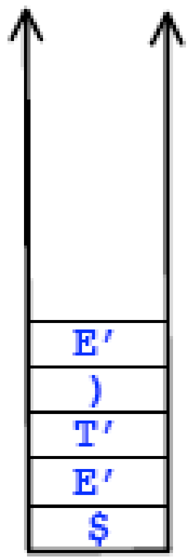
E	→	T E'		
E'	→	+ T E'	|	ε
T	→	F T'		
T'	→	* F T'	|	ε
F	→	(E)	|	id

Input:

(id*id)+id

Output:

E	→	T E'
T	→	F T'
F	→	(E)
E	→	T E'
T	→	F T'
F	→	id
T'	→	* F T'
F	→	id
T'	→	ε



(	id	*	id	)	+	id	$

Table [E', ')'] = E'→ε

Pop E'

Push <nothing>

Print E'→ε

	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)		

Predictive Parsing

Example

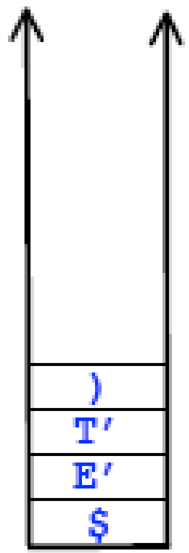
$E \rightarrow T\ E'$
$E' \rightarrow +\ T\ E'\ |\ \varepsilon$
$T \rightarrow F\ T'$
$T' \rightarrow *\ F\ T'\ |\ \varepsilon$
$F \rightarrow (\ E\ )\ |\ \underline{id}$

Input:

(id*id)+id

Output:

$E \rightarrow T\ E'$
$T \rightarrow F\ T'$
$F \rightarrow (\ E\ )$
$E \rightarrow T\ E'$
$T \rightarrow F\ T'$
$F \rightarrow \underline{id}$
$T' \rightarrow *\ F\ T'$
$F \rightarrow \underline{id}$
$T' \rightarrow \varepsilon$
$E' \rightarrow \varepsilon$



(	id	*	id	)	+	id	$

Table [E', ')'] = $E' \rightarrow \varepsilon$
Pop E'
Push <nothing>
Print $E' \rightarrow \varepsilon$

	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)		

Predictive Parsing

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

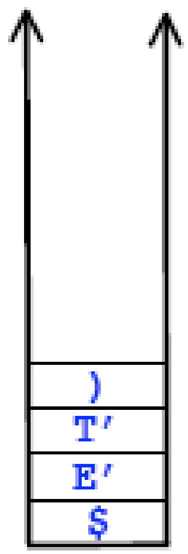
Example

Input:

(id*id)+id

Output:

E → T E'
T → F T'
F → (E)
E → T E'
T → F T'
F → id
T' → * F T'
F → id
T' → ε
E' → ε



(	id	*	id	)	+	id	$

Top of Stack matches next input
Pop and Scan

	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)		

Predictive Parsing

Example

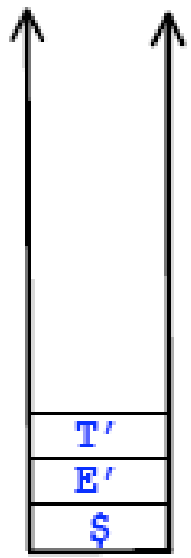
E	$\rightarrow$	$T\ E'$										
E'	$\rightarrow$	$+\ T\ E' \mid \varepsilon$										
T	$\rightarrow$	$F\ T'$										
T'	$\rightarrow$	$*\ F\ T' \mid \varepsilon$										
F	$\rightarrow$	$(\ E\ ) \mid$ id										

Input:

(id*id)+id

Output:

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



(	id	*	id	)	+	id	$

Top of Stack matches next input
Pop and Scan

	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			$F \rightarrow (E)$		

Predictive Parsing

Example

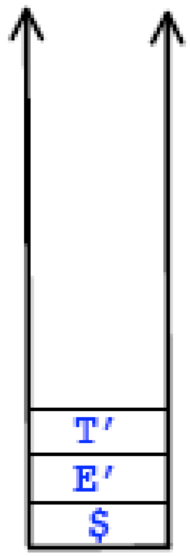
E	→	T E'
E'	→	+ T E' | ε
T	→	F T'
T'	→	* F T' | ε
F	→	(E) | id

Input:

(id*id)+id

Output:

E → T E'
T → F T'
F → (E)
E → T E'
T → F T'
F → id
T' → * F T'
F → id
T' → ε
E' → ε



(	id	*	id	)	+	id	$

Table [T', '+'] = T'→ε
Pop T'
Push <nothing>
Print T'→ε

	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)		

Predictive Parsing

Example

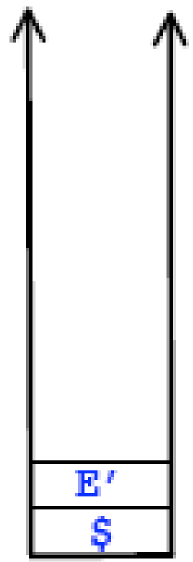
E	→	T E'
E'	→	+ T E' | ε
T	→	F T'
T'	→	* F T' | ε
F	→	(E) | id

Input:

(id*id)+id

Output:

E	→	T E'
T	→	F T'
F	→	(E)
E	→	T E'
T	→	F T'
F	→	id
T'	→	* F T'
F	→	id
T'	→	ε
E'	→	ε
T'	→	ε



(	id	*	id	)	+	id	$

Table [T', '+'] = T'→ε

Pop T'

Push <nothing>

Print T'→ε

	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)		

Predictive Parsing

Example

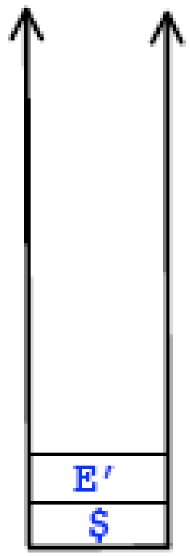
E → T E'
E' → + T E' | ε
T → F T'
T' → * F T' | ε
F → (E) | id

Input:

(id*id)+id

Output:

E → T E'
T → F T'
F → (E)
E → T E'
T → F T'
F → id
T' → * F T'
F → id
T' → ε
E' → ε
T' → ε



(	id	*	id	)	+	id	$

Table [E', '+'] = E'→+TE'
Pop E'
Push E'
Push T
Push '+'
Print E'→+TE'

	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)		

Predictive Parsing

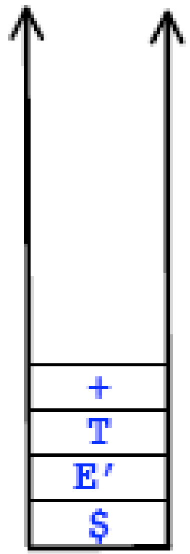
Example

Input:

(id*id)+id

Output:

E → T E'
T → F T'
F → (E)
E → T E'
T → F T'
F → id
T' → * F T'
F → id
T' → ε
E' → ε
T' → ε
E' → + T E'



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

(	id	*	id	)	+	id	$

Table [E', '+'] = E'→+TE'
Pop E'
Push E'
Push T
Push '+'
Print E'→+TE'

	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)		

Predictive Parsing

Example

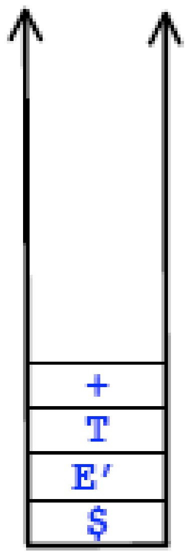
E → T E'
E' → + T E' | ε
T → F T'
T' → * F T' | ε
F → (E) | id

Input:

(id*id)+id

Output:

E → T E'
T → F T'
F → (E)
E → T E'
T → F T'
F → id
T' → * F T'
F → id
T' → ε
E' → ε
T' → ε
E' → + T E'



(	id	*	id	)	+	id	$

Top of Stack matches next input
Pop and Scan

	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)		

Predictive Parsing

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

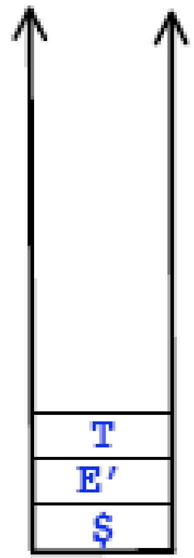
Example

Input:

(id*id)+id

Output:

E → T E'
T → F T'
F → (E)
E → T E'
T → F T'
F → id
T' → * F T'
F → id
T' → ε
E' → ε
T' → ε
E' → + T E'



(	id	*	id	)	+	id	$

Top of Stack matches next input
Pop and Scan

	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)		

Predictive Parsing

Example

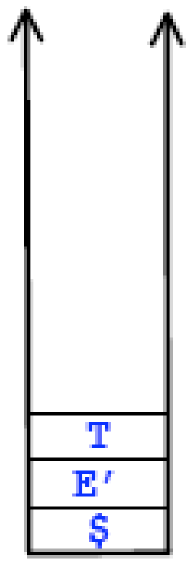
$E \rightarrow T\ E'$
$E' \rightarrow +\ T\ E'\ |\ \varepsilon$
$T \rightarrow F\ T'$
$T' \rightarrow *\ F\ T'\ |\ \varepsilon$
$F \rightarrow (\ E\ )\ |\ \underline{id}$

Input:

(id*id)+id

Output:

$E \rightarrow T\ E'$
$T \rightarrow F\ T'$
$F \rightarrow (\ E\ )$
$E \rightarrow T\ E'$
$T \rightarrow F\ T'$
$F \rightarrow \underline{id}$
$T' \rightarrow *\ F\ T'$
$F \rightarrow \underline{id}$
$T' \rightarrow \varepsilon$
$E' \rightarrow \varepsilon$
$T' \rightarrow \varepsilon$
$E' \rightarrow +\ T\ E'$



(	id	*	id	)	+	id	$

Table [T, id] = $T \rightarrow FT'$
Pop T
Push T'
Push F
Print $T \rightarrow FT'$

	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	$F \rightarrow \underline{id}$			$F \rightarrow (E)$		

Predictive Parsing

Example

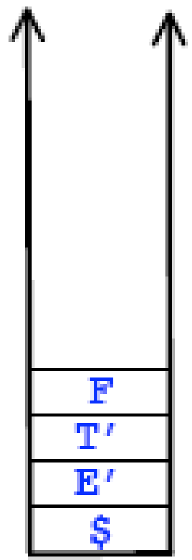
E → T E'
E' → + T E' | ε
T → F T'
T' → * F T' | ε
F → (E) | id

Input:

(id*id)+id

Output:

E → T E'
T → F T'
F → (E)
E → T E'
T → F T'
F → id
T' → * F T'
F → id
T' → ε
E' → ε
T' → ε
E' → + T E'
T → F T'



(	id	*	id	)	+	id	$

Table [T, id] = T→FT'
Pop T
Push T'
Push F
Print T→FT'

	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)		

Predictive Parsing

Example

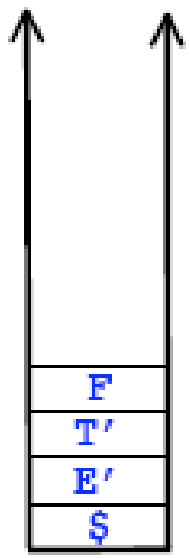
$E \rightarrow T\ E'$
$E' \rightarrow +\ T\ E'\ |\ \varepsilon$
$T \rightarrow F\ T'$
$T' \rightarrow *\ F\ T'\ |\ \varepsilon$
$F \rightarrow (\ E\ )\ |\ \underline{id}$

Input:

(id*id)+id

Output:

$E \rightarrow T\ E'$
$T \rightarrow F\ T'$
$F \rightarrow (\ E\ )$
$E \rightarrow T\ E'$
$T \rightarrow F\ T'$
$F \rightarrow \underline{id}$
$T' \rightarrow *\ F\ T'$
$F \rightarrow \underline{id}$
$T' \rightarrow \varepsilon$
$E' \rightarrow \varepsilon$
$T' \rightarrow \varepsilon$
$E' \rightarrow +\ T\ E'$
$T \rightarrow F\ T'$



(	id	*	id	)	+	id	$

Table [F, id] = $F \rightarrow \underline{id}$
Pop F
Push id
Print $F \rightarrow \underline{id}$

	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)		

Predictive Parsing

Example

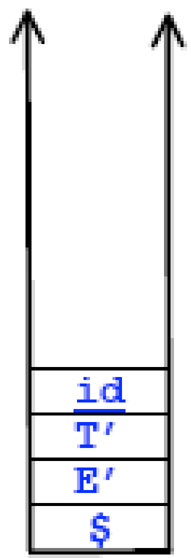
E	→	T	E'					
E'	→	+	T	E'	|	ε		
T	→	F	T'					
T'	→	*	F	T'	|	ε		
F	→	(	E	)	|	id		

Input:

(id*id)+id

Output:

E → T E'
T → F T'
F → (E)
E → T E'
T → F T'
F → id
T' → * F T'
F → id
T' → ε
E' → ε
T' → ε
E' → + T E'
T → F T'
F → id



(	id	*	id	)	+	id	$

Table [F, id] = F→id
Pop F
Push id
Print F→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)		

Predictive Parsing

Example

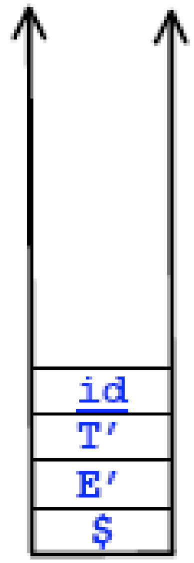
$E \rightarrow T\ E'$
$E' \rightarrow +\ T\ E' \mid \varepsilon$
$T \rightarrow F\ T'$
$T' \rightarrow *\ F\ T' \mid \varepsilon$
$F \rightarrow (\ E\ ) \mid \underline{id}$

Input:

(id*id)+id

Output:

$E \rightarrow T\ E'$
$T \rightarrow F\ T'$
$F \rightarrow (\ E\ )$
$E \rightarrow T\ E'$
$T \rightarrow F\ T'$
$F \rightarrow \underline{id}$
$T' \rightarrow *\ F\ T'$
$F \rightarrow \underline{id}$
$T' \rightarrow \varepsilon$
$E' \rightarrow \varepsilon$
$T' \rightarrow \varepsilon$
$E' \rightarrow +\ T\ E'$
$T \rightarrow F\ T'$
$F \rightarrow \underline{id}$



(	id	*	id	)	+	id	$

Top of Stack matches next input
Pop and Scan

	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	$F \rightarrow \underline{id}$			$F \rightarrow (E)$		

Predictive Parsing

Example

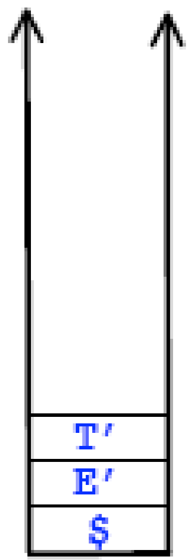
E → T E'
E' → + T E' | ε
T → F T'
T' → * F T' | ε
F → (E) | id

Input:

(id*id)+id

Output:

E → T E'
T → F T'
F → (E)
E → T E'
T → F T'
F → id
T' → * F T'
F → id
T' → ε
E' → ε
T' → ε
E' → + T E'
T → F T'
F → id



(	id	*	id	)	+	id	$

Top of Stack matches next input
Pop and Scan

	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)		

Predictive Parsing

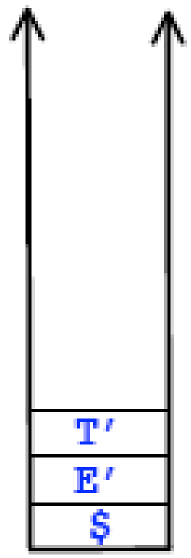
Example

Input:

(id*id)+id

Output:

E → T E'
T → F T'
F → (E)
E → T E'
T → F T'
F → id
T' → * F T'
F → id
T' → ε
E' → ε
T' → ε
E' → + T E'
T → F T'
F → id



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

(	id	*	id	)	+	id	$

Table [T', $] = T'→ε
Pop T'
Push <nothing>
Print T'→ε

	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)		

Predictive Parsing

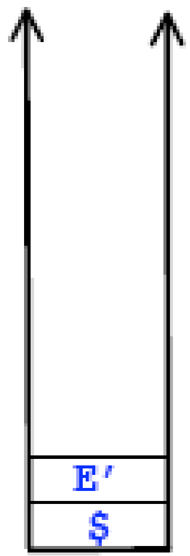
Example

Input:

`(id*id)+id`

Output:

E → T E'
T → F T'
F → (E)
E → T E'
T → F T'
F → id
T' → * F T'
F → id
T' → ε
E' → ε
T' → ε
E' → + T E'
T → F T'
F → id
T' → ε



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

(	id	*	id	)	+	id	$

Table [T', $] = T'→ε
Pop T'
Push <nothing>
Print T'→ε

	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)		

Predictive Parsing

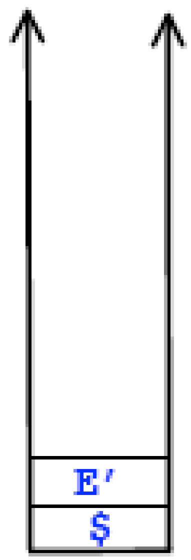
Example

Input:

(id*id)+id

Output:

E → T E'
T → F T'
F → (E)
E → T E'
T → F T'
F → id
T' → * F T'
F → id
T' → ε
E' → ε
T' → ε
E' → + T E'
T → F T'
F → id
T' → ε



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

(	id	*	id	)	+	id	$

Table [E', $] = E'→ε
Pop E'
Push <nothing>
Print E'→ε

	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)		

Predictive Parsing

Example

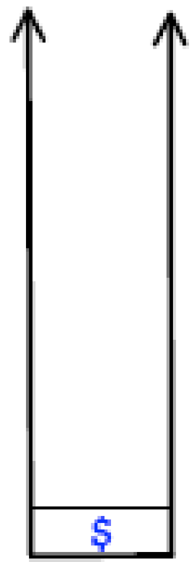
$E \rightarrow T\ E'$
$E' \rightarrow +\ T\ E' \mid \varepsilon$
$T \rightarrow F\ T'$
$T' \rightarrow *\ F\ T' \mid \varepsilon$
$F \rightarrow (\ E\ ) \mid$ <u>id</u>

Input:

(id*id)+id

Output:

E → T E'
T → F T'
F → (E)
E → T E'
T → F T'
F → id
T' → * F T'
F → id
T' → ε
E' → ε
T' → ε
E' → + T E'
T → F T'
F → id
T' → ε
E' → ε



(	id	*	id	)	+	id	$

Table [E', $] = E'→ε
Pop E'
Push <nothing>
Print E'→ε

	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)		

Predictive Parsing

Example

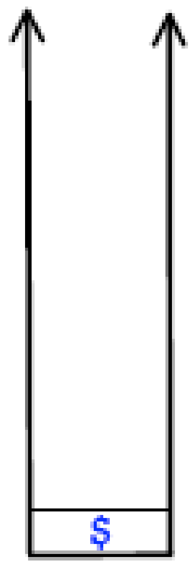
E → T E'
E' → + T E' | ε
T → F T'
T' → * F T' | ε
F → (E) | id

Input:

(id*id)+id

Output:

E → T E'
T → F T'
F → (E)
E → T E'
T → F T'
F → id
T' → * F T'
F → id
T' → ε
E' → ε
T' → ε
E' → + T E'
T → F T'
F → id
T' → ε
E' → ε



(	id	*	id	)	+	id	$

Input symbol == $
Top of stack == $
Loop terminates with success

	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)		

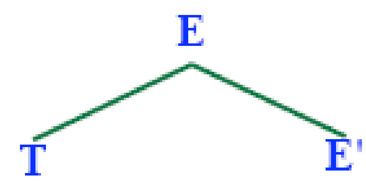
Reconstructing the Parse Tree

Input:

`(id*id)+id`

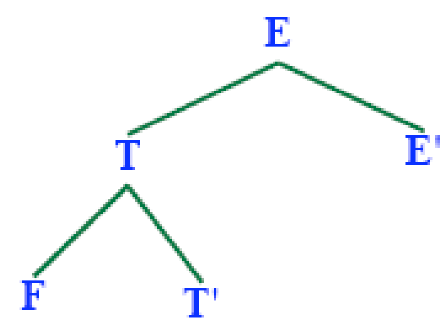
Output:

E → T E'



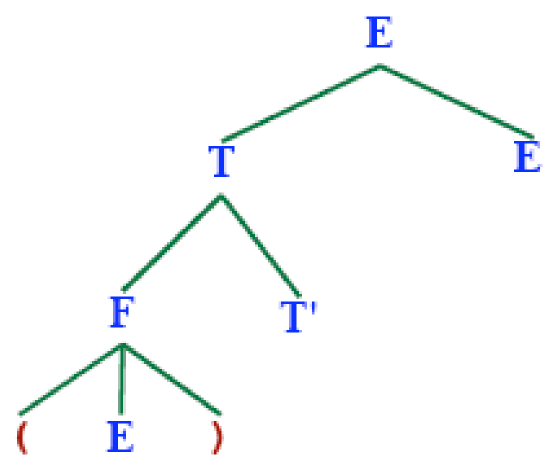
Output:

E → T E'
T → F T'



Output:

E → T E'
T → F T'
F → (E)



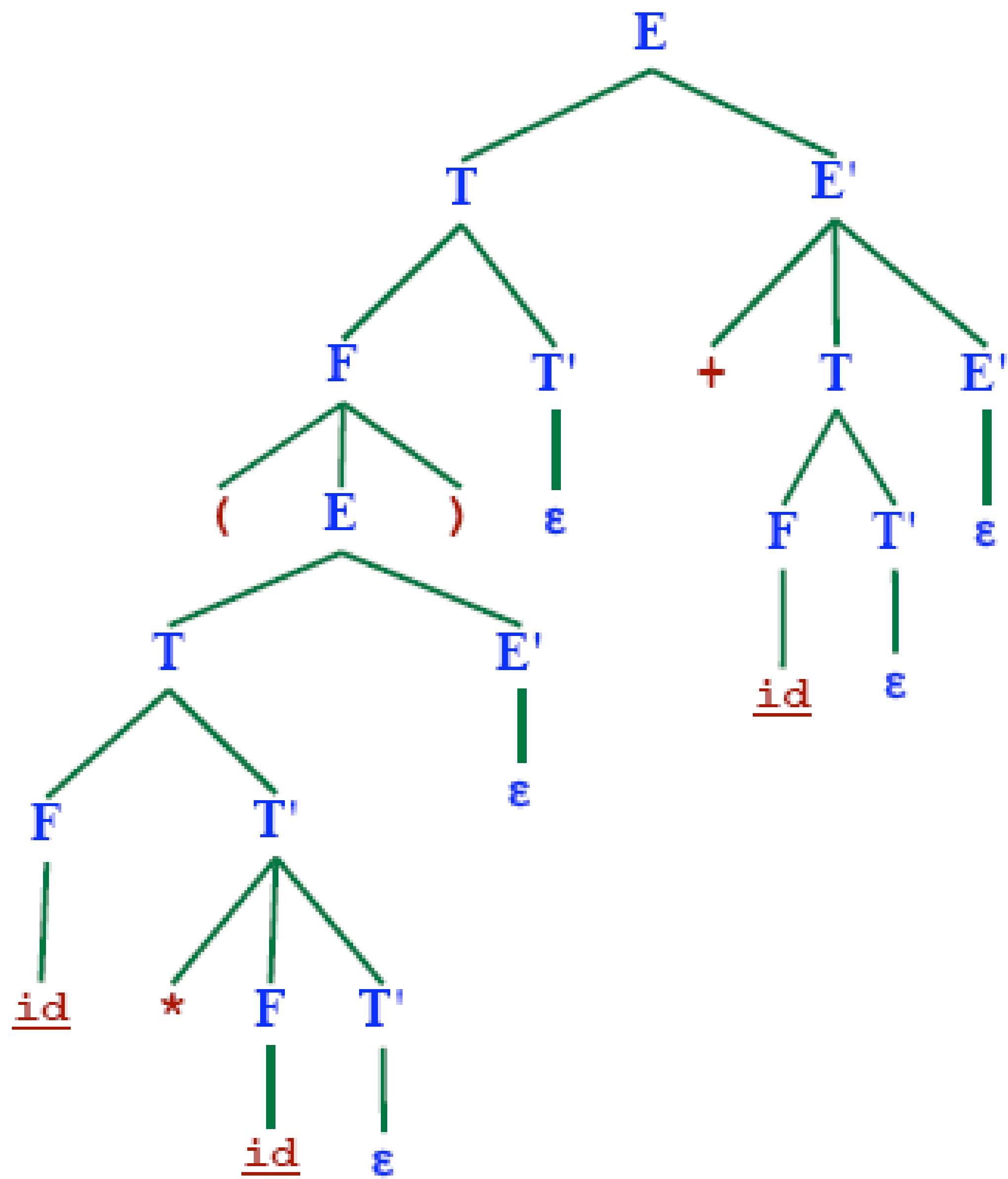
Reconstructing the Parse Tree

Input:

(id*id)+id

Output:

E → T E'
T → F T'
F → (E)
E → T E'
T → F T'
F → id
T' → * F T'
F → id
T' → ε
E' → ε
T' → ε
E' → + T E'
T → F T'
F → id
T' → ε
E' → ε



Reconstructing the Parse Tree

Input:

(id*id)+id

Output:

$E \rightarrow T\ E'$
$T \rightarrow F\ T'$
$F \rightarrow (\ E\ )$
$E \rightarrow T\ E'$
$T \rightarrow F\ T'$
$F \rightarrow \underline{id}$
$T' \rightarrow *\ F\ T'$
$F \rightarrow \underline{id}$
$T' \rightarrow \varepsilon$
$E' \rightarrow \varepsilon$
$T' \rightarrow \varepsilon$
$E' \rightarrow +\ T\ E'$
$T \rightarrow F\ T'$
$F \rightarrow \underline{id}$
$T' \rightarrow \varepsilon$
$E' \rightarrow \varepsilon$

Leftmost Derivation:

E
$T\ E'$
$F\ T'\ E'$
$(\ E\ )\ T'\ E'$
$(\ T\ E'\ )\ T'\ E'$
$(\ F\ T'\ E'\ )\ T'\ E'$
$(\ \underline{id}\ T'\ E'\ )\ T'\ E'$
$(\ \underline{id} * F\ T'\ E'\ )\ T'\ E'$
$(\ \underline{id} * \underline{id}\ T'\ E'\ )\ T'\ E'$
$(\ \underline{id} * \underline{id}\ E'\ )\ T'\ E'$
$(\ \underline{id} * \underline{id}\ )\ T'\ E'$
$(\ \underline{id} * \underline{id}\ )\ E'$
$(\ \underline{id} * \underline{id}\ ) + T\ E'$
$(\ \underline{id} * \underline{id}\ ) + F\ T'\ E'$
$(\ \underline{id} * \underline{id}\ ) + \underline{id}\ T'\ E'$
$(\ \underline{id} * \underline{id}\ ) + \underline{id}\ E'$
$(\ \underline{id} * \underline{id}\ ) + \underline{id}$

$S \rightarrow iEtS \mid iEtSeS \mid a$

$E \rightarrow c$

- Left recursive: The grammar is not left recursive
- Left factoring: The grammar after eliminating left factoring is:

$S \rightarrow iEtSS' \mid a$

$S' \rightarrow \varepsilon \mid eS$

$E \rightarrow c$

Non-Term	FIRST	FOLLOW
S		
S’		
E		

S → iEtS | iEtSeS | a

E → c

- Left recursive: The grammar is not left recursive
- Left factoring: The grammar after eliminating left factoring is:

S → iEtSS' | a

S' → ε |eS

E → c

Non-Term	FIRST	FOLLOW
S	{i, a}	{$, e, FOLLOW(S')} = {$,e}
S'	{ε, e}	{$,e}
E	{c}	{t}

S → iEtSS' | a
S' → ε |eS
E → c

NT	FIRST	FOLLOW
S	{i, a}	{$,e}
S'	{ε, e}	{$,e}
E	{c}	{t}

Predictive Parsing Table:

NT ↓ T →	i	t	a	e	c	$
S						
S'						
E						

$S \rightarrow iEtSS' \mid a$

$S' \rightarrow \varepsilon \mid eS$

$E \rightarrow c$

NT	FIRST	FOLLOW
S	{i, a}	{$,e}
S’	{ε, e}	{$,e}
E	{c}	{t}

Predictive Parsing Table:

NT ↓ / T →	i	t	a	e	c	$
S	S→iEtSS’		S→a			
S’				S’ →ε S’ →eS		S’ →ε
E					E→c	

From the predictive parsing table, there are multiple entries of S’ on e. So, this grammar is ambiguous thus this will not be LL(1)

Not LL(1)

- A left recursive grammar cannot be LL(1). Elimination of Left Recursion is mandatory
- Non-Determinism is not allowed in LL(1). We eliminate non-determinism by using Left Factoring is
- If the predictive parsing table has multiple entries, it cannot be a LL(1) grammar
- So, ambiguous grammars cannot be LL(1)

S→ aABb
A→ c/ ε
B→d/ ε

Check if the above grammar is LL(1) or not

NT	FIRST	FOLLOW
S		
A		
B		

NT↓ T→	a	b	c	d	$
S					
A					
B					

$$
\begin{aligned}
S \rightarrow\ & aSbS \\
|\ & bSaS \\
|\ & \varepsilon
\end{aligned}
$$

NT	FIRST	FOLLOW
S		

Check if the above grammar is LL(1) or not

NT ↓ T →	a	b	$
S			

S→ aB | ε

B→ bC | ε

C→cS | ε

Check if the above grammar is LL(1) or not

NT	FIRST	FOLLOW
S		
B		
C		

NT↓ T→	a	b	c	$
S				
A				
B				

Transition Diagram for Predictive Parsers

- Useful for visualizing predictive parsers.
- To construct Transition Diagram from a grammar
  - Eliminate left recursion
  - Left factor the grammar
  - Then for each nonterminal A
    - Create an initial and final state
    - For each production $A \rightarrow X_1X_2 \ldots X_k$, create a path from the initial to the final state, with edges labeled $X_1, X_2, \ldots, X_k$. If $\rightarrow \varepsilon$, the path is an edge labeled ε.

Transition Diagram for Predictive Parsers

- Predictive parser begins in the start state for the start symbol
- Suppose at any time it is in state s with an edge
  - labeled by a terminal a to state t

    - If the next input is a the parser advances in input and moves to state t
    - If the edge from s to t is labeled by ε, then the parser moves immediately to state t without advancing the input
  - labeled by a nonterminal A

    - Parser goes to the start state for A
    - If it ever reaches the final state of A it will immediately go back to state t

Transition Diagram for Predictive Parsers

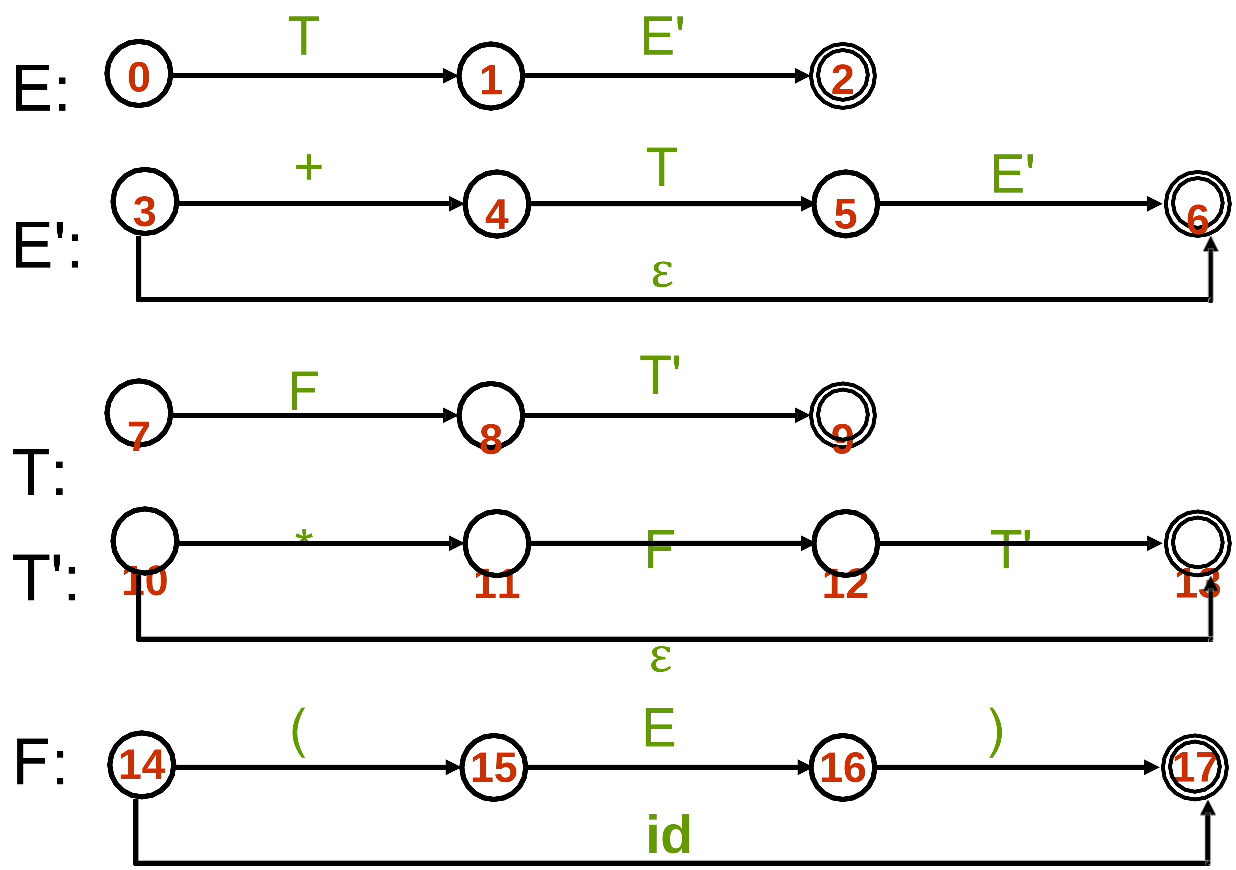
E → TE'

E' → +TE' | ε

T → FT'

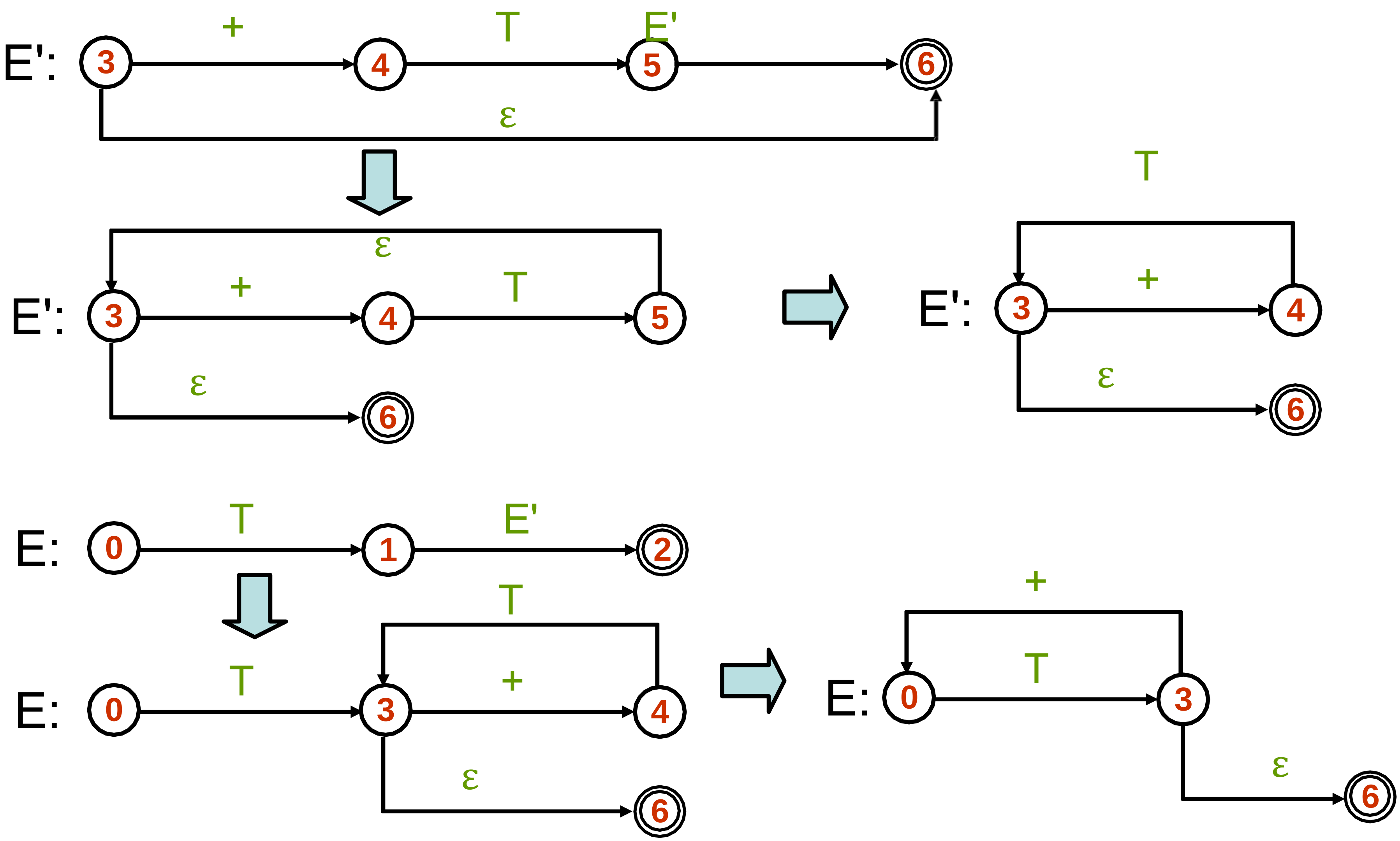
T' → *FT' | ε

F → (E) | **id**



Simplification of Transition Diagrams

- TDs can be simplified by substituting one in another



Simplification of Transition Diagrams

- Complete set of TDs
- A C implementation of this simplified version of the parser runs 20-25% faster than the original version