Parsing

Part III

Top Down Parsing

- 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!!

Top-Down Parsing

- Recursive-Descent Parsing
  - 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
- Predictive Parsing
  - 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.

Recursive Descent Parsing (Backtracking)

Input: **aabbde**

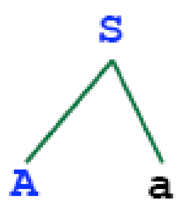
↑

S

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)

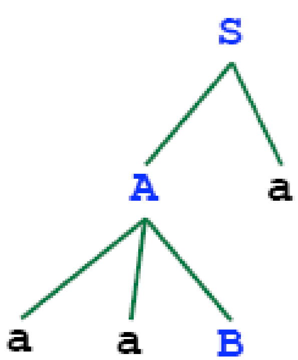
Input: aabbde



1. S $\rightarrow$ 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)

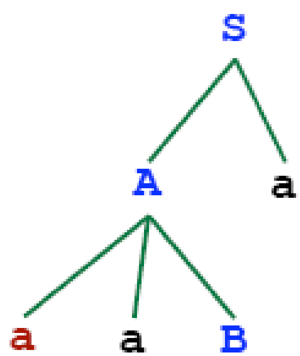
Input: aabbde



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

Recursive Descent Parsing (Backtracking)

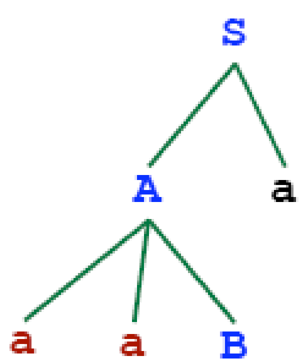
Input: `aabbde`



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

Recursive Descent Parsing (Backtracking)

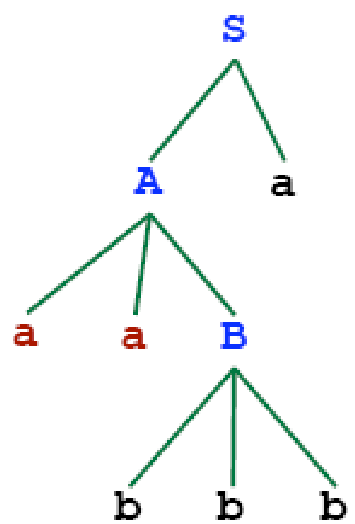
Input: `aabbde`



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

Recursive Descent Parsing (Backtracking)

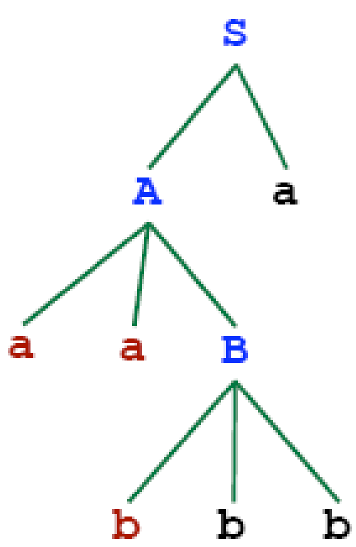
Input: `aabbde`



1.	S	$\rightarrow$	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)

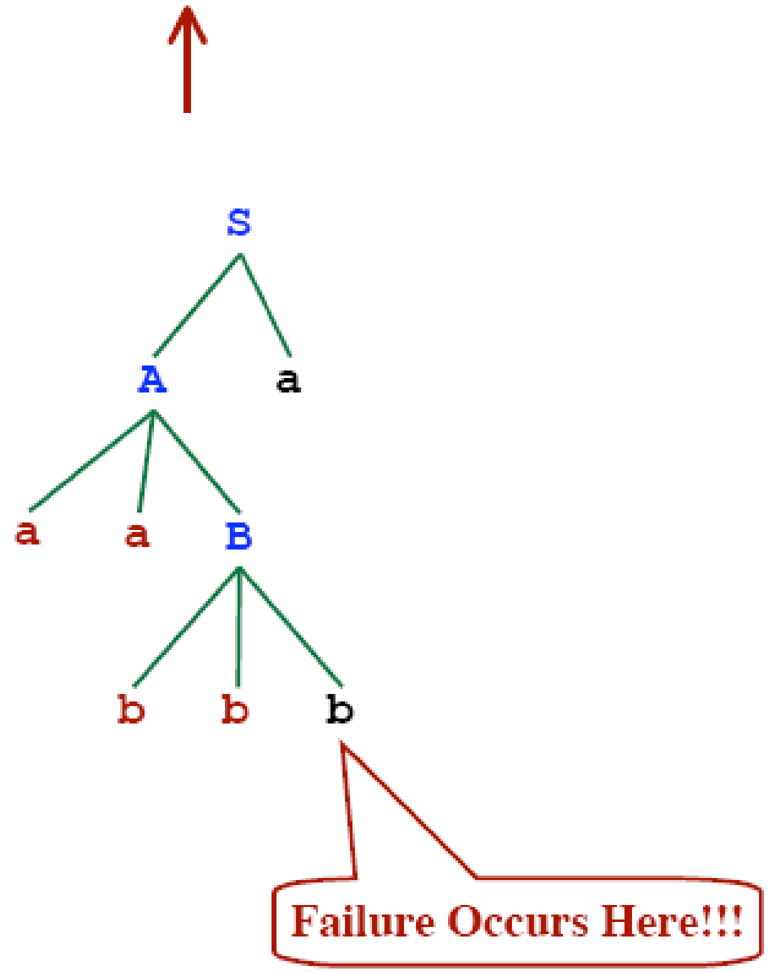
Input: **aabbde**



1. S → 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)

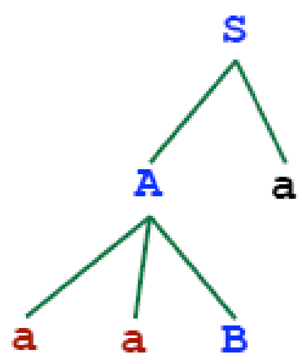
Input: `aabbde`



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

Recursive Descent Parsing (Backtracking)

Input: `aabbde`

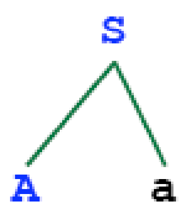


1.	S	$\rightarrow$	A`a`
2.		$\rightarrow$	C`e`
3.	A	$\rightarrow$	`aa`B
4.		$\rightarrow$	`aaba`
5.	B	$\rightarrow$	`bbb`
6.	C	$\rightarrow$	`aa`D
7.	D	$\rightarrow$	`bbd`

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

Recursive Descent Parsing (Backtracking)

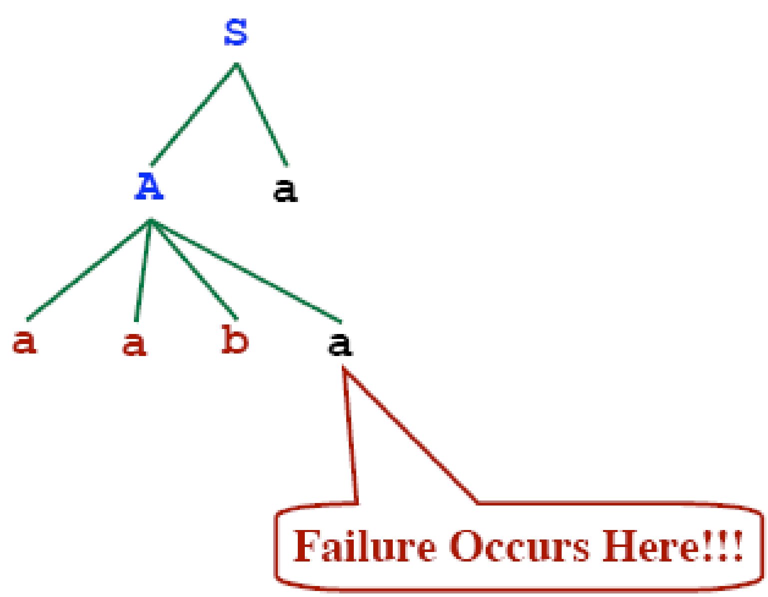
Input: `aabbde`



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)

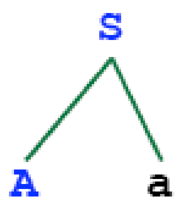
Input: aabbde



1. S $\rightarrow$ 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



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`

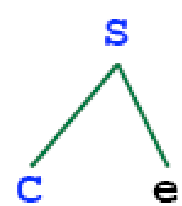
↑

S

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

Recursive Descent Parsing (Backtracking)

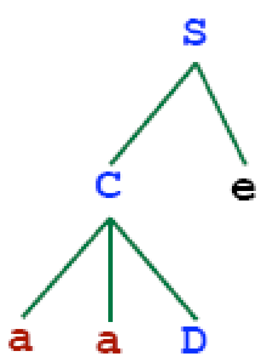
Input: aabbde



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

Recursive Descent Parsing (Backtracking)

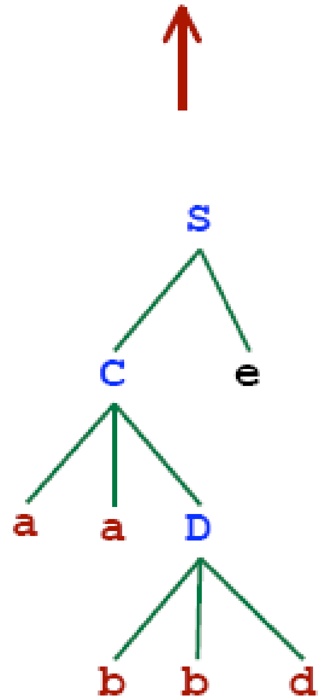
Input: `aabbde`



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

Recursive Descent Parsing (Backtracking)

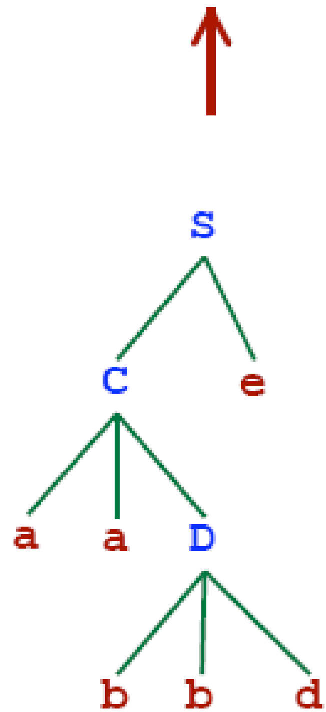
Input: `aabbde`



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

Successfully parsed!!

Recursive-Descent Parsing Algorithm

- A recursive-descent parsing program consists of a set of procedures – one for each non-terminal
- Execution begins with the procedure for the start symbol
  - Announces success if the procedure body scans the entire input

```
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
        }
    }
}
```

Predictive 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 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.)

$A \rightarrow aBb \mid bAB$

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

```
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 | $\alpha \Rightarrow^*$ aw, plus ε if $\alpha \Rightarrow^* \varepsilon$ }

Example:

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

FIRST (F) = { (, <u>id</u> }

FIRST (T') = { *, ε}

FIRST (T) = { (, <u>id</u> }

FIRST (E') = { +, ε}

FIRST (E) = { (, <u>id</u> }

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
```

First - Example

- P → i | c | n T S
- Q → P | a S | b S c S T
- R → b | ε
- S → c | R n | ε
- T → R S q

- FIRST(P) = {i,c,n}
- FIRST(Q) = {i,c,n,a,b}
- FIRST(R) = {b,ε}
- FIRST(S) = {c,b,n,ε}
- FIRST(T) = {b,c,n,q}

First - Example

- $S \rightarrow a\ S\ e \mid S\ T\ S$
- $T \rightarrow R\ S\ e \mid Q$
- $R \rightarrow r\ S\ r \mid \varepsilon$
- $Q \rightarrow S\ T \mid \varepsilon$

- $\text{FIRST}(S) = \{a\}$
- $\text{FIRST}(R) = \{r, \varepsilon\}$
- $\text{FIRST}(T) = \{r, a, \varepsilon\}$
- $\text{FIRST}(Q) = \{a, \varepsilon\}$

FOLLOW Sets

- FOLLOW(A) is the set of terminals (including end marker of input - \$) that may follow non-terminal A in some sentential form.
- FOLLOW(A) = $\{c \mid S \Rightarrow^{+} \ldots Ac\ldots\} \cup \{\$\}$ if $S \Rightarrow^{+} \ldots A$
- For example, consider $L \Rightarrow^{+}$ (())(L)L

  Both ‘)’ and end of file can follow L
- NOTE: ε is ***never*** in FOLLOW sets

Computing FOLLOW(A)

1. If A is start symbol, put \$ in FOLLOW(A)
2. Productions of the form $B \rightarrow \alpha\ A\ \beta$,

   Add FIRST(β) – {ε} to FOLLOW(A)

INTUITION: Suppose $B \rightarrow AX$ and FIRST(X) = {c}

$$S \Rightarrow^{+} \alpha\ B\ \beta \Rightarrow \alpha\ A\ X\ \beta \Rightarrow^{+} \alpha\ A\ c\ \delta\ \beta$$

= FIRST(X)

3. Productions of the form $B \rightarrow \alpha\ A$ or

$$B \rightarrow \alpha\ A\ \beta \text{ where } \beta \Rightarrow^{*} \varepsilon$$

Add FOLLOW(B) to FOLLOW(A)

INTUITION:

- Suppose $B \rightarrow Y\ A$

$$S \Rightarrow^{+} \alpha\ B\ \beta \Rightarrow \alpha\ Y\ A\ \beta$$

FOLLOW(B)

- Suppose $B \rightarrow A\ X$ and $X \Rightarrow^{*} \lambda$

$$S \Rightarrow^{+} \alpha\ B\ \beta \Rightarrow \alpha\ A\ X\ \beta \Rightarrow^{*} \alpha\ A\ \beta$$

FOLLOW(B)

Example

- $S \rightarrow a\ S\ e \mid B$
- $B \rightarrow b\ B\ C\ f \mid C$
- $C \rightarrow c\ C\ g \mid d \mid \varepsilon$

- FIRST(C) = $\{c,d,\varepsilon\}$
- FIRST(B) = $\{b,c,d,\varepsilon\}$
- FIRST(S) = $\{a,b,c,d,\varepsilon\}$

- FOLLOW(C) =
- FOLLOW(B) =
- FOLLOW(S) = $\{\$\}$

Using rule #1

Assume the first non-terminal is the start symbol

Example

- $S \rightarrow a \, \underline{S \, e} \mid B$
- $B \rightarrow b \, \underline{B \, C \, f} \mid C$
- $C \rightarrow c \, \underline{C \, g} \mid d \mid \varepsilon$

- FIRST(C) = {c,d,ε}
- FIRST(B) = {b,c,d,ε}
- FIRST(S) = {a,b,c,d,ε}

- FOLLOW(C) = {f,g}

- FOLLOW(B) = {c,d,f}

- FOLLOW(S) = {$,e}

Using rule #2

Example

- $S \rightarrow a\ S\ e \mid \underline{B}$
- $B \rightarrow b\ B\ C\ f \mid \underline{C}$
- $C \rightarrow c\ C\ g \mid d \mid \varepsilon$

- FIRST(C) = {c,d,ε}
- FIRST(B) = {b,c,d,ε}
- FIRST(S) = {a,b,c,d,ε}

- FOLLOW(C) =

  $\{f,g\} \cup \text{FOLLOW}(B)$
  $= \{c,d,e,f,g,\$\}$

- FOLLOW(B) =

  $\{c,d,f\} \cup \text{FOLLOW}(S)$
  $= \{c,d,e,f,\$\}$

- FOLLOW(S) = {$, e }

Using rule #3

Example

- $S \rightarrow (A) \mid \varepsilon$
- $A \rightarrow T\ E$
- $E \rightarrow \&\ T\ E \mid \varepsilon$
- $T \rightarrow (A) \mid a \mid b \mid c$

- FIRST(T) = {(,a,b,c}
- FIRST(E) = {&, ε }
- FIRST(A) = {(,a,b,c}
- FIRST(S) = {(, ε}

- FOLLOW(S) =
- FOLLOW(A) =
- FOLLOW(E) =
- FOLLOW(T) =

Example

- S → (A) | ε
- A → T E
- E → & T E | ε
- T → (A) | a | b | c

- FIRST(T) = {(,a,b,c}
- FIRST(E) = {&, ε }
- FIRST(A) = {(,a,b,c}
- FIRST(S) = {(, ε}

- FOLLOW(S) = {$}
- FOLLOW(A) = {) }
- FOLLOW(E) =
FOLLOW(A) = {) }
- FOLLOW(T) =
FIRST(E) ∪ FOLLOW(A) ∪ FOLLOW(E) = {&,)}

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 → aB
→ cD
→ 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

Table Driven Predictive Parsing

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.

Example

$E \rightarrow T\ E'$

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

$T \rightarrow F\ T'$

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

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

Term ...

+Term +Term + ... +Term

Factor ...

*** Factor * Factor * ... * Factor**

Step 1: From grammar, construct table.

Step 2: Use table to parse strings.

Table Driven Predictive Parsing

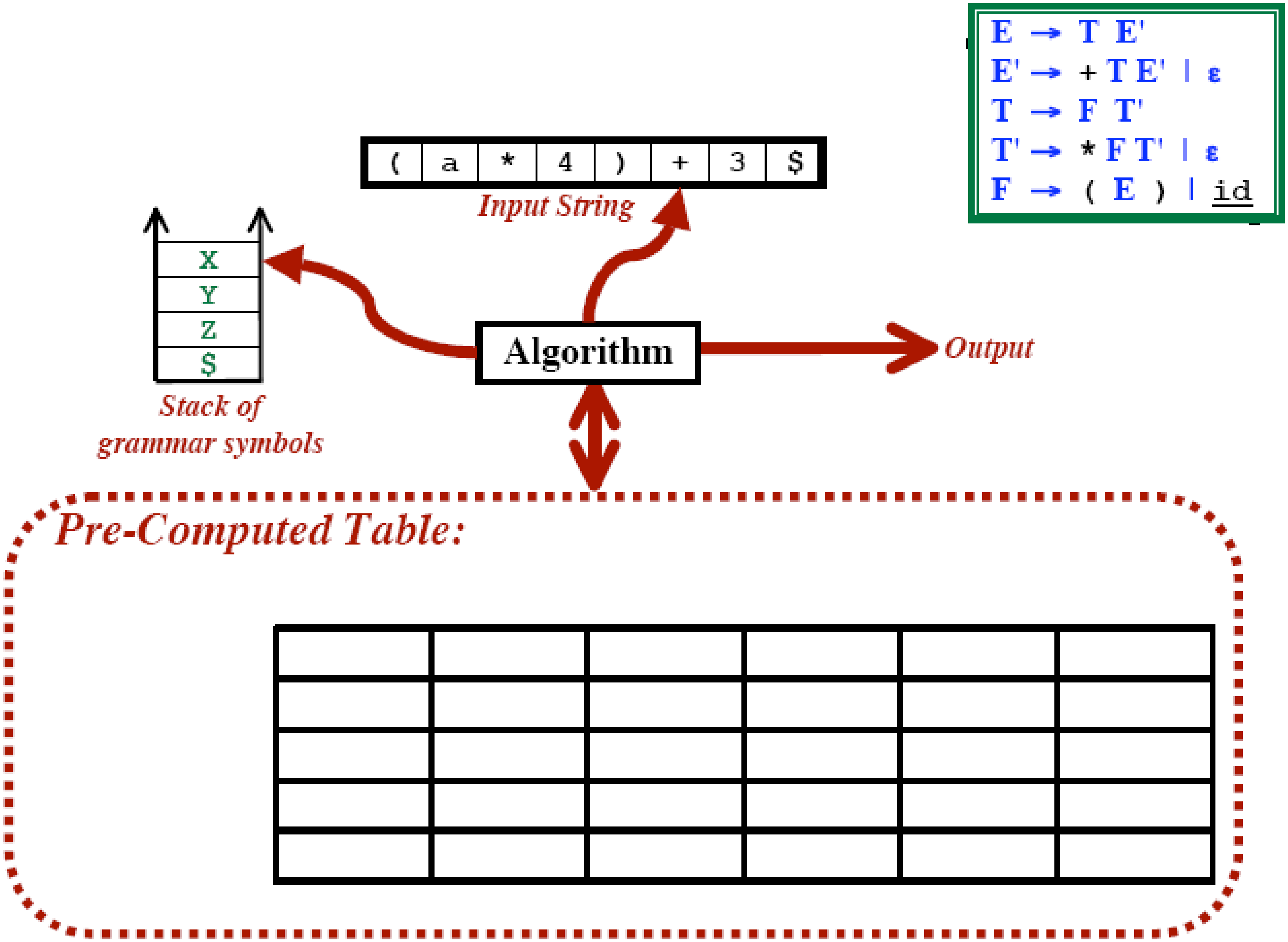
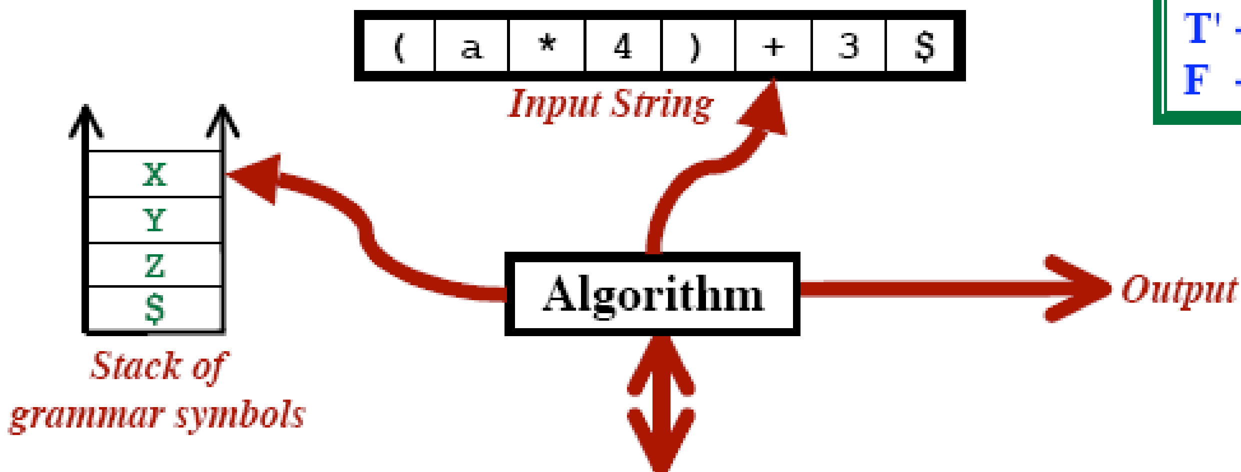


Table Driven Predictive Parsing

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



Pre-Computed Table:

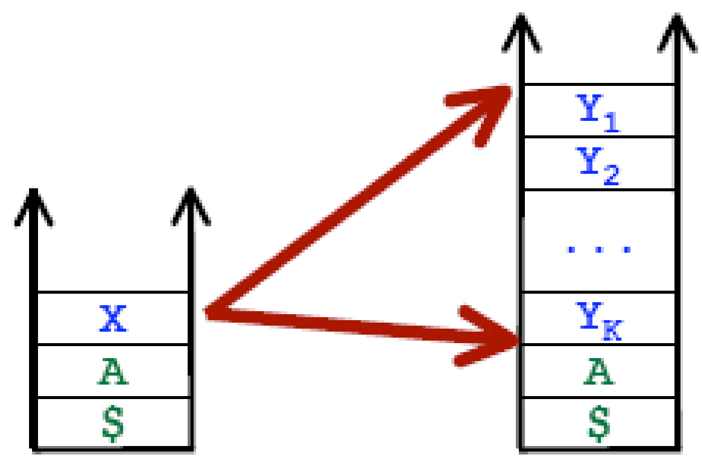
Input Symbols, plus $

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

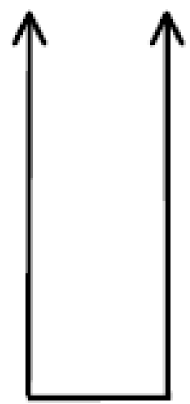
Example

Input:

(id*id)+id

Output:

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



(	id	*	id	)	+	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

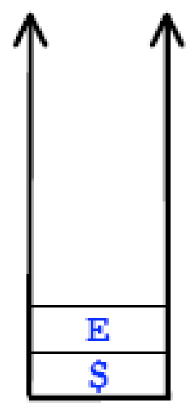
Example

Input:

`(id*id)+id`

Output:

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



(	id	*	id	)	+	id	$

Add $ to end of input
Push $
Push 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

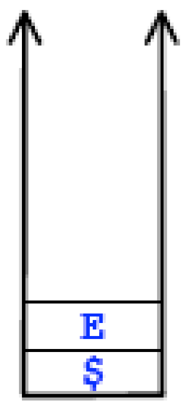
Example

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

Input:

(id*id)+id

Output:



(	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

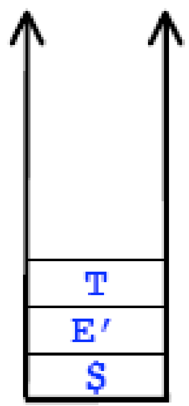
E → T E'
E' → + T E' | ε
T → F T'
T' → * F T' | ε
F → (E) | id

Input:

(id*id)+id

Output:

E → T E'



(	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

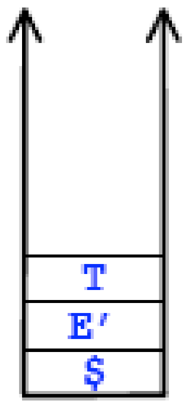
Input:

(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

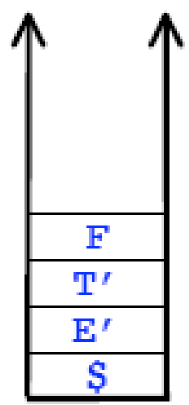
Input:

(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

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

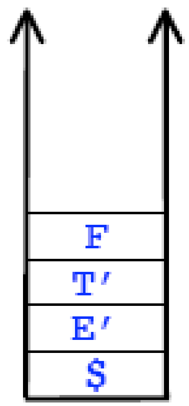
Input:

(id*id)+id

Output:

E → T E'

T → F T'



(	id	*	id	)	+	id	$

Table [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

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

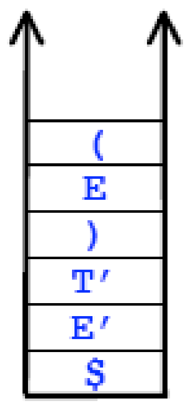
Example

Input:

(id*id)+id

Output:

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



(	id	*	id	)	+	id	$

Table [F, '('] = $F \rightarrow (E)$
Pop F
Push)
Push E
Push (
Print $F \rightarrow (E)$

	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

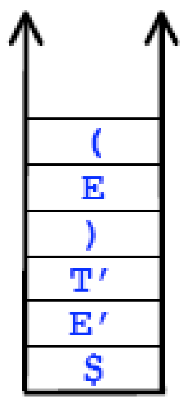
Example

Input:

(id*id)+id

Output:

E → T E'
T → F T'
F → (E)



(	id	*	id	)	+	id	$

Top of Stack matches next input
Pop and Scan

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

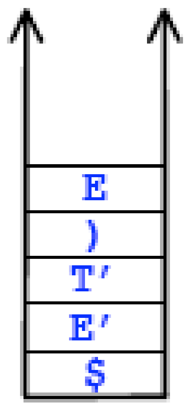
E → T E'
E' → + T E' | ε
T → F T'
T' → * F T' | ε
F → (E) | id

Input:

(id*id)+id

Output:

E → T E'
T → F T'
F → (E)



(	id	*	id	)	+	id	$

Table [E, id] = 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

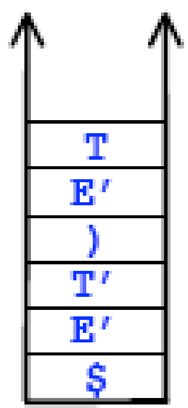
Input:

(id*id)+id

Output:

E → T E'
T → F T'
F → (E)
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, id] = 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

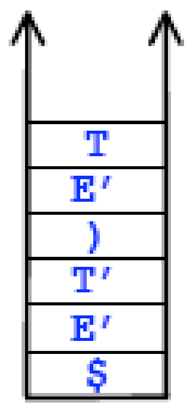
E	→	T E'
E'	→	+ T E' | ε
T	→	F T'
T'	→	* F T' | ε
F	→	(E) | id

Input:

(id*id)+id

Output:

E	→	T E'
T	→	F T'
F	→	(E)
E	→	T E'



(	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

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

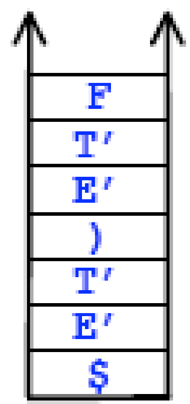
Example

Input:

(id*id)+id

Output:

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



(	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

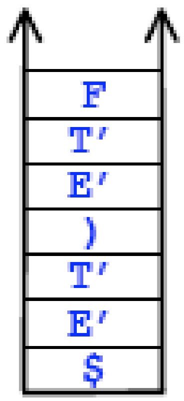
$E \to T\ E'$
$E' \to +\ T\ E' \mid \varepsilon$
$T \to F\ T'$
$T' \to *\ F\ T' \mid \varepsilon$
$F \to (\ E\ ) \mid \underline{id}$

Input:

(id*id)+id

Output:

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



(	id	*	id	)	+	id	$

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

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

Predictive Parsing

Example

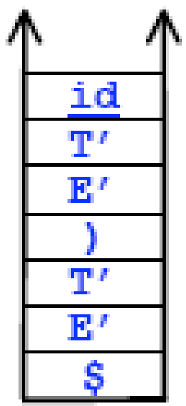
$E \rightarrow T\ E'$
$E' \rightarrow +\ T\ E'\ |\ \varepsilon$
$T \rightarrow F\ T'$
$T' \rightarrow *\ F\ T'\ |\ \varepsilon$
$F \rightarrow (\ E\ )\ |\ \underline{id}$

Input:

(id*id)+id

Output:

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



(	id	*	id	)	+	id	$

Table [F, id] = $F \rightarrow \underline{id}$
Pop F
Push <u>*id*</u>
Print $F \rightarrow \underline{id}$

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

Predictive Parsing

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

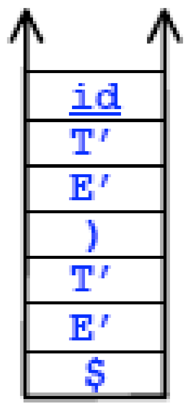
Example

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	$

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

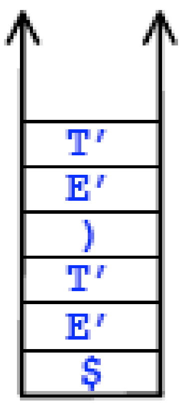
E → T E'
E' → + T E' | ε
T → F T'
T' → * F T' | ε
F → (E) | id

Input:

(id*id)+id

Output:

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



(	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

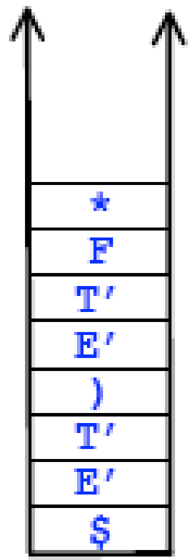
E → T E'
E' → + T E' | ε
T → F T'
T' → * F T' | ε
F → (E) | id

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 [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

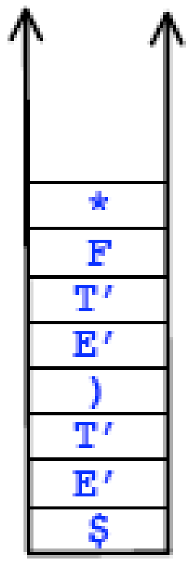
Example

Input:

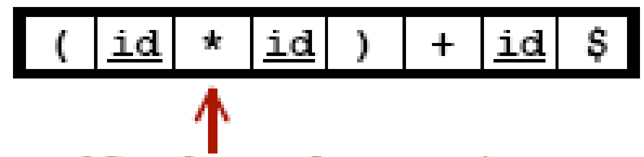
(id*id)+id

Output:

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



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

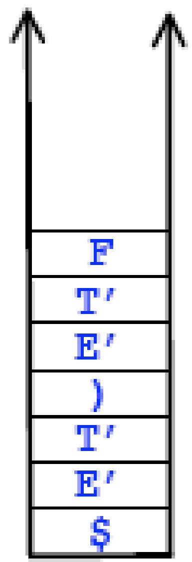
$E \rightarrow T\ E'$
$E' \rightarrow +\ T\ E' \mid \varepsilon$
$T \rightarrow F\ T'$
$T' \rightarrow *\ F\ T' \mid \varepsilon$
$F \rightarrow (\ E\ ) \mid \underline{id}$

Input:

(id*id)+id

Output:

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



(	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

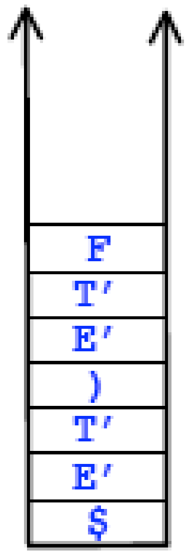
$E \to T\,E'$
$E' \to +\,T\,E' \mid \varepsilon$
$T \to F\,T'$
$T' \to *\,F\,T' \mid \varepsilon$
$F \to (\,E\,) \mid \underline{id}$

Input:

(id*id)+id

Output:

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



(	id	*	id	)	+	id	$

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

	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

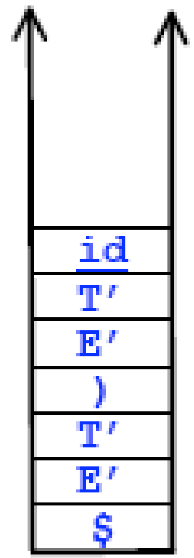
$E \rightarrow T\ E'$
$E' \rightarrow +\ T\ E'\ |\ \varepsilon$
$T \rightarrow F\ T'$
$T' \rightarrow *\ F\ T'\ |\ \varepsilon$
$F \rightarrow (\ E\ )\ |\ \underline{id}$

Input:

(id*id)+id

Output:

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

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

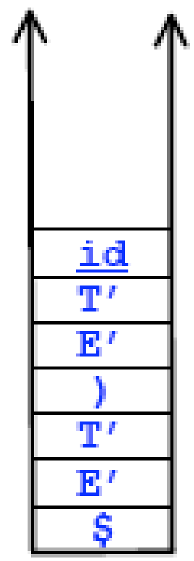
Example

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



(	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

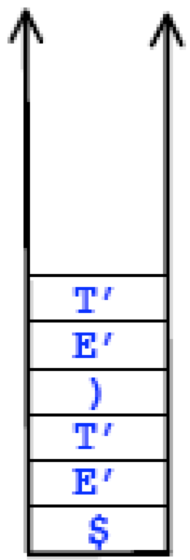
Example

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



(	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

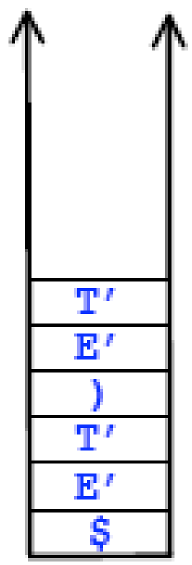
E	→	T	E'	
E'	→	+ T E'	|	ε
T	→	F	T'	
T'	→	* F T'	|	ε
F	→	(E)	|	id

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

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

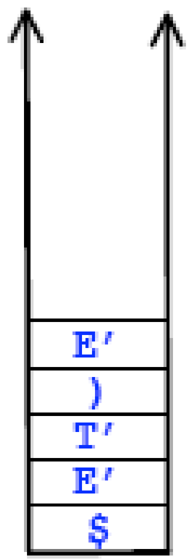
Example

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



(	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

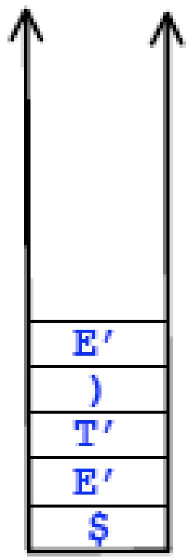
E → T E'
E' → + T E' | ε
T → F T'
T' → * F T' | ε
F → (E) | id

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

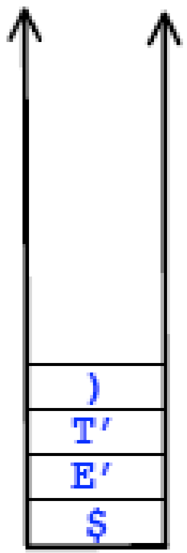
$E \rightarrow T\ E'$
$E' \rightarrow +\ T\ E'\ |\ \varepsilon$
$T \rightarrow F\ T'$
$T' \rightarrow *\ F\ T'\ |\ \varepsilon$
$F \rightarrow (\ E\ )\ |\ \underline{id}$

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	$E \rightarrow TE'$			$E \rightarrow TE'$		
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

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

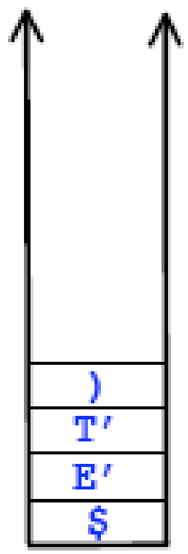
Example

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	$

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

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

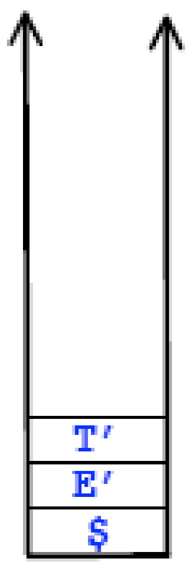
Example

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	$

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

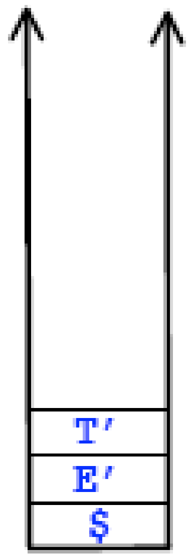
E	→	T E'
E'	→	+ T E' | ε
T	→	F T'
T'	→	* F T' | ε
F	→	(E) | id

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
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

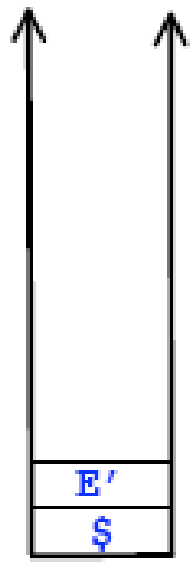
E → T E'
E' → + T E' | ε
T → F T'
T' → * F T' | ε
F → (E) | id

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

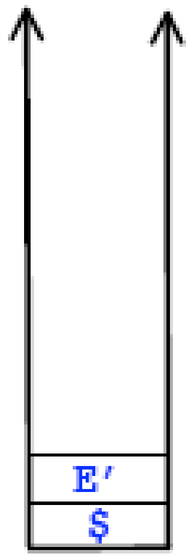
E → T E'
E' → + T E' | ε
T → F T'
T' → * F T' | ε
F → (E) | id

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

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

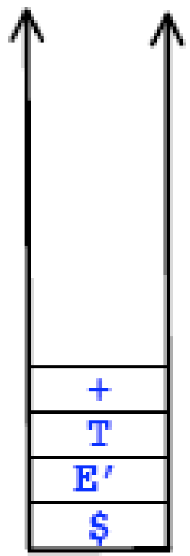
Example

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	$

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

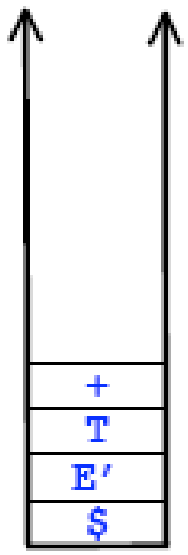
$E \rightarrow T\ E'$
$E' \rightarrow +\ T\ E' \mid \varepsilon$
$T \rightarrow F\ T'$
$T' \rightarrow *\ F\ T' \mid \varepsilon$
$F \rightarrow (\ E\ ) \mid \underline{id}$

Input:

(id*id)+id

Output:

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



(	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

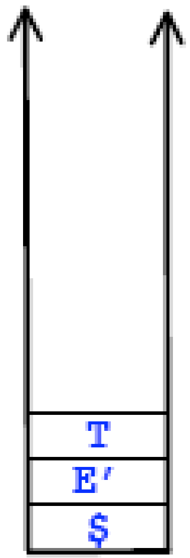
$E \to T\ E'$
$E' \to +\ T\ E'\ \vert\ \varepsilon$
$T \to F\ T'$
$T' \to *\ F\ T'\ \vert\ \varepsilon$
$F \to (\ E\ )\ \vert$ id

Input:

(id*id)+id

Output:

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



(	id	*	id	)	+	id	$

Top of Stack matches next input
Pop and Scan

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

Predictive Parsing

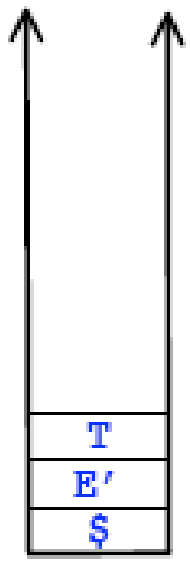
Example

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 [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

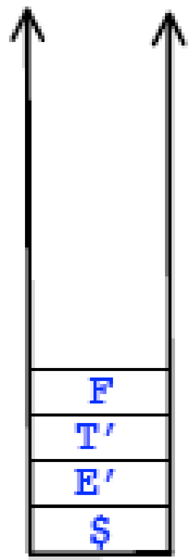
$E \rightarrow T\ E'$
$E' \rightarrow +\ T\ E' \mid \varepsilon$
$T \rightarrow F\ T'$
$T' \rightarrow *\ F\ T' \mid \varepsilon$
$F \rightarrow (\ E\ ) \mid \underline{id}$

Input:

(id*id)+id

Output:

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



(	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

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

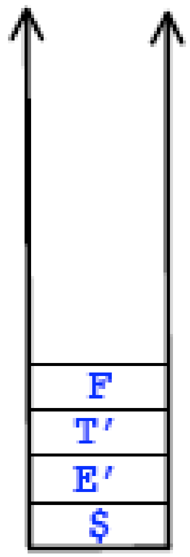
Example

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'
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

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

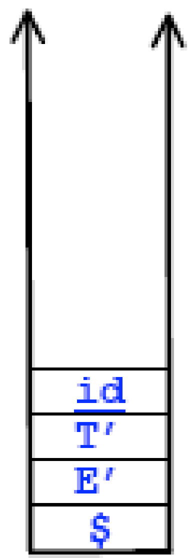
Example

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'
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

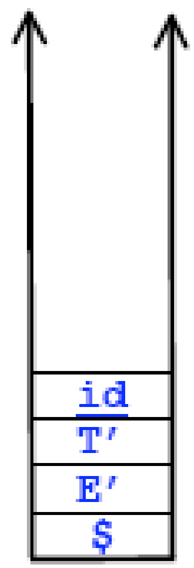
Input:

(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	$

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

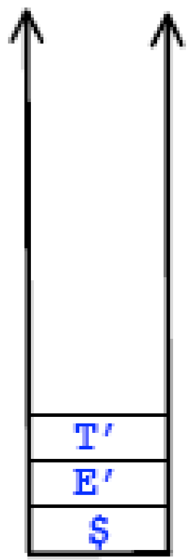
E → T E'
E' → + T E' | ε
T → F T'
T' → * F T' | ε
F → (E) | id

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
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

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

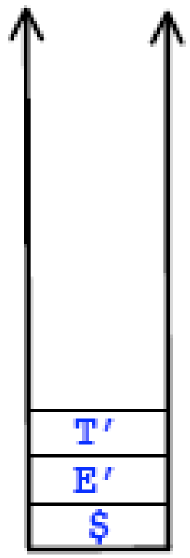
Example

Input:

(id*id)+id

Output:

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



(	id	*	id	)	+	id	$

Table [T', $] = $T' \rightarrow \varepsilon$
Pop T'
Push <nothing>
Print $T' \rightarrow \varepsilon$

	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

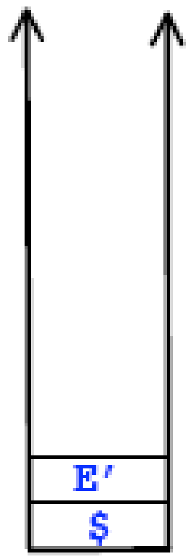
$E \rightarrow T\ E'$
$E' \rightarrow +\ T\ E' \mid \varepsilon$
$T \rightarrow F\ T'$
$T' \rightarrow *\ F\ T' \mid \varepsilon$
$F \rightarrow (\ E\ ) \mid \underline{id}$

Input:

(id*id)+id

Output:

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



(	id	*	id	)	+	id	$

Table [T', $] = $T' \rightarrow \varepsilon$
Pop T'
Push <nothing>
Print $T' \rightarrow \varepsilon$

	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

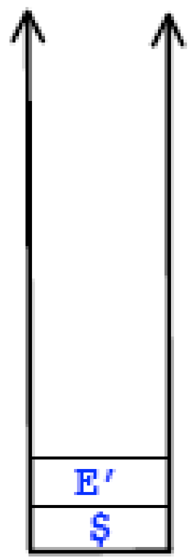
$E \to T\ E'$
$E' \to +\ T\ E' \mid \varepsilon$
$T \to F\ T'$
$T' \to *\ F\ T' \mid \varepsilon$
$F \to (\ E\ ) \mid$ id

Input:

(id*id)+id

Output:

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



(	id	*	id	)	+	id	$

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

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

Predictive Parsing

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

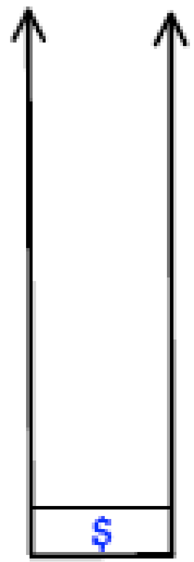
Example

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'
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

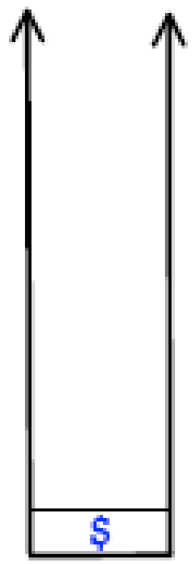
E → T E'
E' → + T E' | ε
T → F T'
T' → * F T' | ε
F → (E) | id

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)		

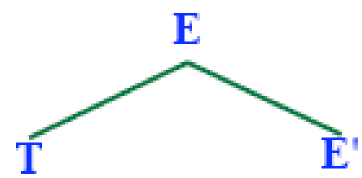
Reconstructing the Parse Tree

Input:

`(id*id)+id`

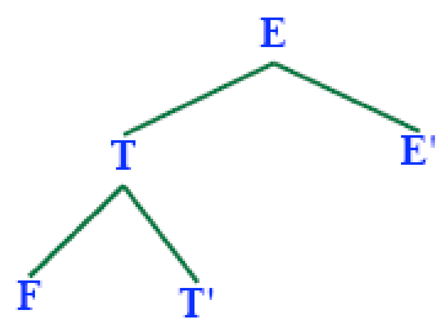
Output:

E → T E'



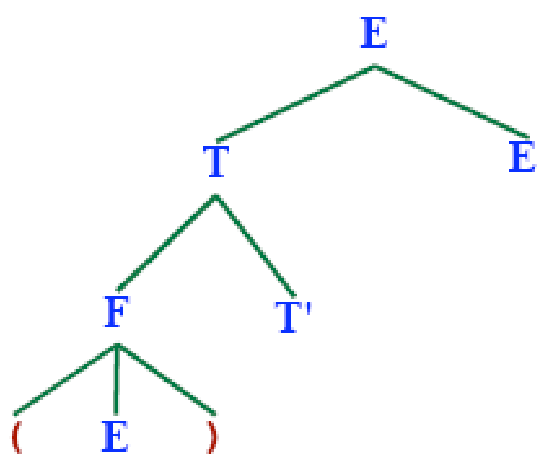
Output:

E → T E'
T → F T'



Output:

E → T E'
T → F T'
F → (E)



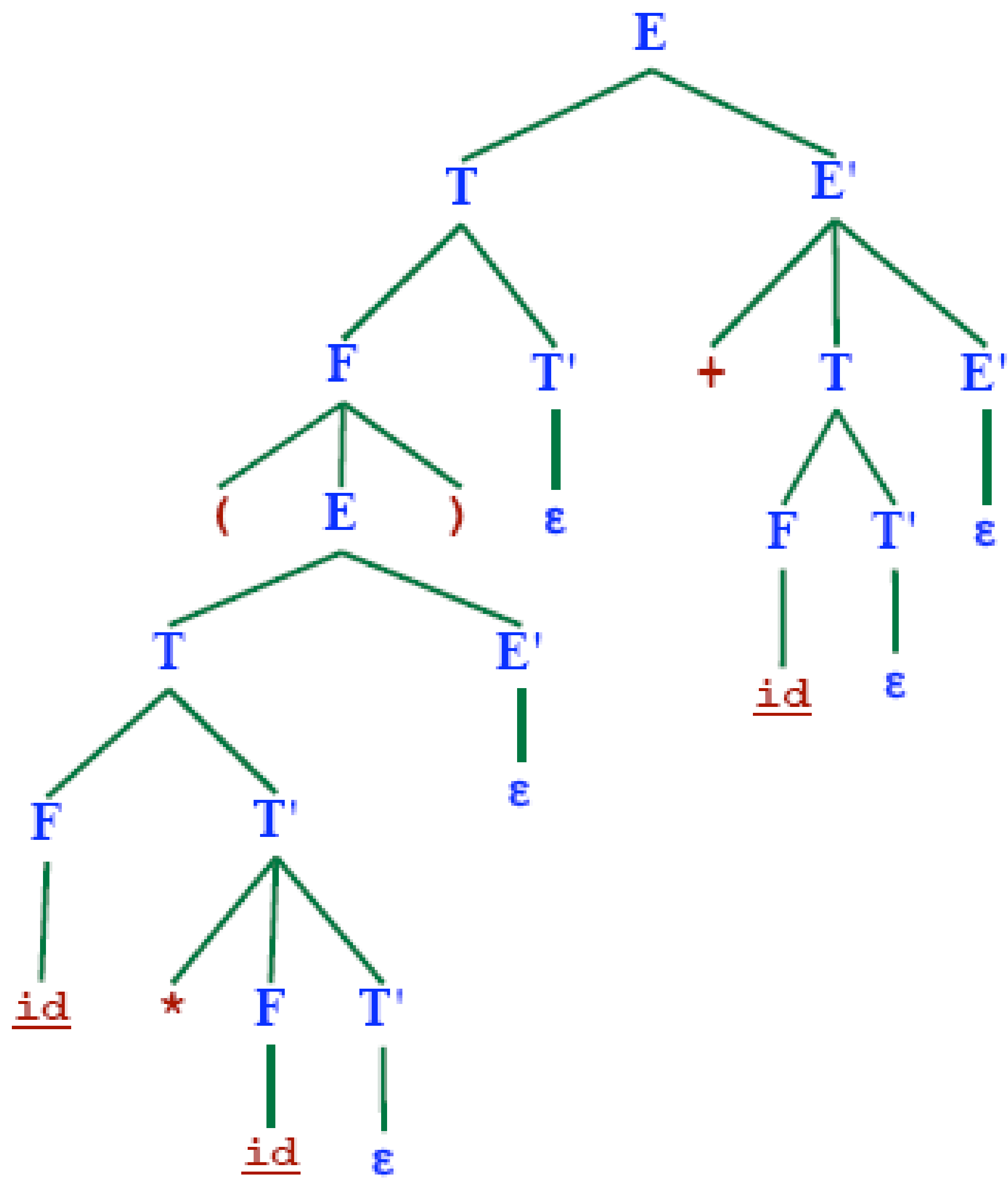
Reconstructing the Parse Tree

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	→	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

Leftmost Derivation:

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

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 $A \rightarrow X_1 X_2 \ldots X_k$, create a path from the initial to the final state, with edges labeled X_1, X_2, …, X_k. If $A \rightarrow \varepsilon$, 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

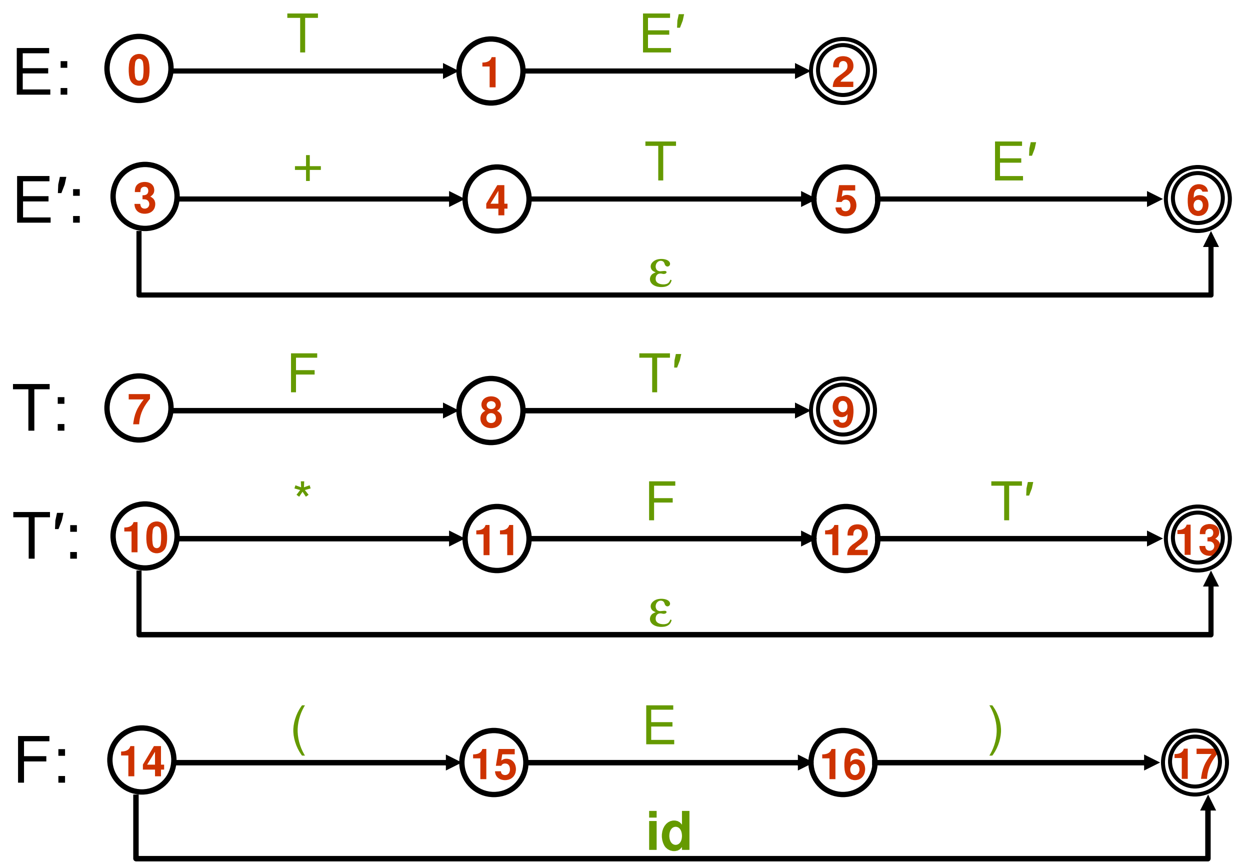
E → TE′

E′ → +TE′ | ε

T → FT′

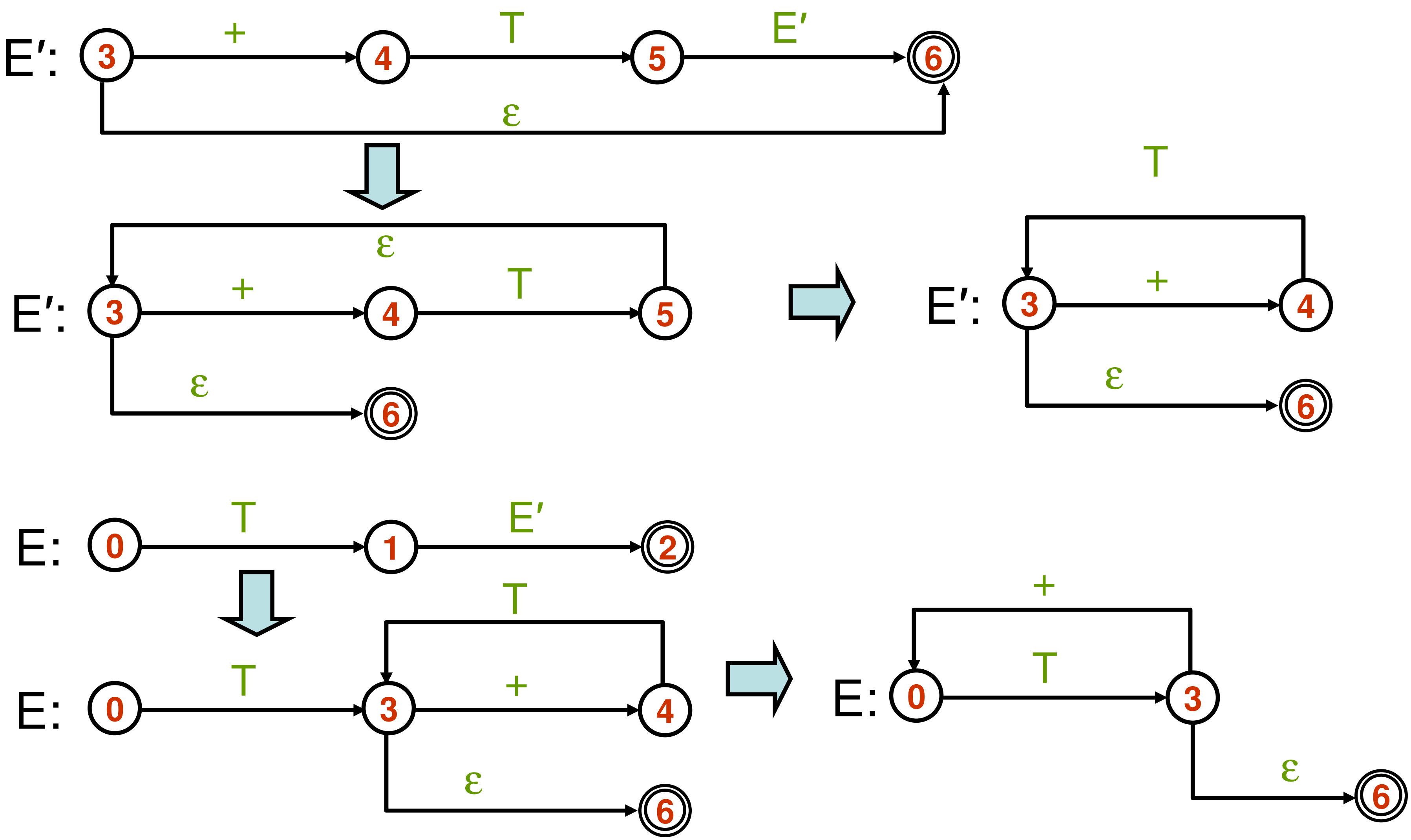
T′ → *FT′ | ε

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