40-414 Compiler Design

Introduction to Parsing

Lecture 4

Exercise

Question?

- Consider the grammar

  E → T X　　　　X → + E | ε

  T → (E) | int Y　　　　Y → * T | ε

- Write Recursive Descent Procedures including panic mode error recovery for all non-terminals.

- Write a step-by-step parsing of input 'int * int'

- Draw the parse tree of the input

Procedure E

Follow(E) = {), $}

- Consider the grammar

$E \rightarrow T\ X$ $\qquad$ $X \rightarrow +\ E \mid \varepsilon$

$T \rightarrow (\ E\ ) \mid \mathrm{int}\ Y$ $\qquad$ $Y \rightarrow *\ T \mid \varepsilon$

```
procedure E ;
   { if lookahead is in { ( , int }
          then { call T; call X }
          else  if lookahead is in { $, ) }
                 then { print ( 'missing E on line …' ); exit }
                 else {  print ( 'illegal lookahead on line …' );
                         lookahed := get_next_token;
                         call E
                      }
   }
```

Procedure T

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

- Consider the grammar

$E \rightarrow T\ X \qquad\qquad X \rightarrow +\ E \mid \varepsilon$

$T \rightarrow (\ E\ ) \mid int\ Y \qquad\qquad Y \rightarrow *\ T \mid \varepsilon$

```
procedure T ;
   { if lookahead  = ‘(‘
          then { call Match ( ‘(‘ ); call E; call Match( ‘)’ ); }
            else if lookahead  = int
                    then { call Match ( int ); call Y }
                    else  if lookahead is in { + , $, ) }
                    then { print ( ‘missing T on line …’ ); exit }
                    else {   print ( ‘illegal lookahead on line …’ ) ;
                             lookahed := get_next_token;
                              call T }
   }
```

Procedure X

Follow(X) = {), $}

- Consider the grammar

$E \rightarrow T\ X$ $\qquad$ $X \rightarrow +\ E \mid \varepsilon$

$T \rightarrow (\ E\ ) \mid \text{int}\ Y$ $\qquad$ $Y \rightarrow *\ T \mid \varepsilon$

```
procedure X ;
   { if lookahead = '+'
            then { call Match ( '+' ); call E }
            else  if lookahead is in { $, ) }
                     then exit;
                     else {   print ( 'illegal lookahead on line …' ) ;
                              lookahed := get_next_token;
                               call  X }
}
```

Procedure Y

Follow(Y) = {+,) , $}

- Consider the grammar

$E \to T\ X$ $\qquad$ $X \to +\ E \mid \varepsilon$

$T \to (\ E\ ) \mid \text{int}\ Y$ $\qquad$ $Y \to *\ T \mid \varepsilon$

```
procedure Y ;
   { if lookahead = ‘*‘
            then { call Match ( ‘*’ ); call T }
            else  if lookahead is in { $, ), + }
                       then exit;
                       else {    print ( ‘illegal lookahead on line …’ ) ;
                                 lookahed := get_next_token;
                                  call  Y }
 }
```

R.D. Parsing Example

$E \rightarrow T\ X$
$T \rightarrow (\ E\ )\ |\ \text{int}\ Y$
$X \rightarrow +\ E\ |\ \varepsilon$
$Y \rightarrow *\ T\ |\ \varepsilon$

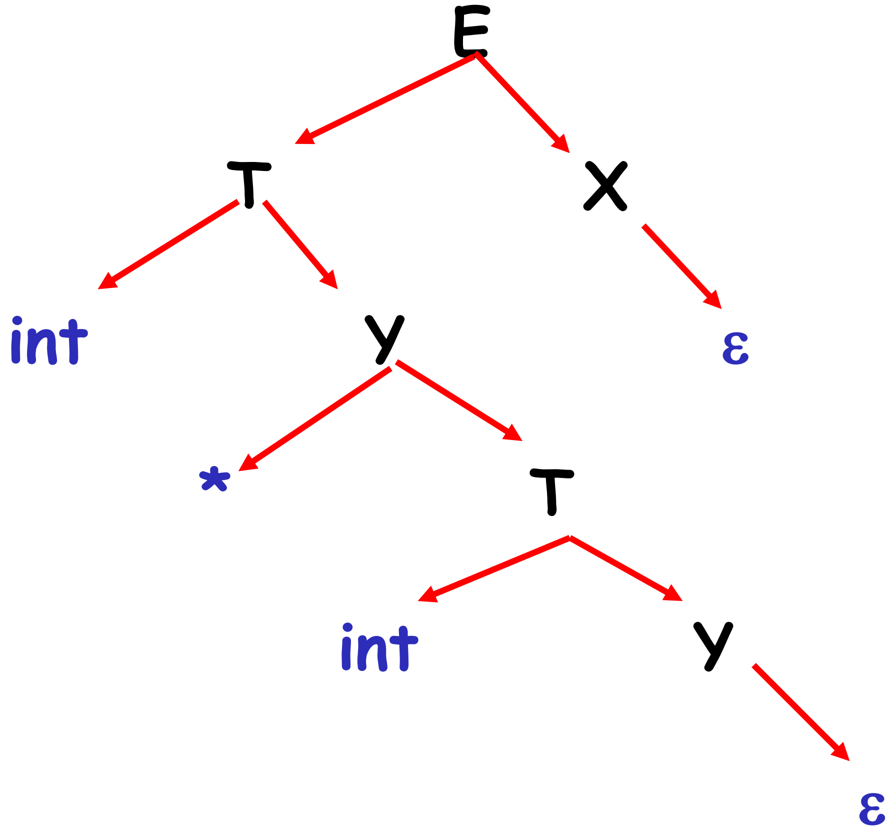
Current Routin		Input	Action
E	→	int * int $	call T
T	→	int * int $	Match (int)
Match	→	int * int $	Call Lexer
T	←	* int $	call Y
Y	→	* int $	Match (*)
Match	→	int $	Call Lexer
Y	←	int $	call T
T	→	int $	Match (int)
Match	→	int $	Call Lexer
T	←	$	call Y
Y	→	$	exit
T	←	$	exit
Y	←	$	exit
T	←	$	exit
E	←	$	exit
X	→	$	exit
E	←	$	ACCEPT

lookaheads

.
.
.

Calls: →
Returns: ←

Parsing Tree of int * int $



Current Routine	
E	→
T	→
Match	→
T	←
Y	→
Match	→
Y	←
T	→
Match	→
T	←
Y	→
T	←
Y	←
T	←
E	←
X	→
E	←

Question?

How many strings does the following grammar generate?

$A \rightarrow B\ B$
$B \rightarrow C\ C$
$C \rightarrow 1 \mid 2$

- ○ 7
- ○ 15
- ○ 2
- ○ 8
- ○ 16
- ○ 4

Answer!

How many strings does the following grammar generate?

A → B B
B → C C
C → 1| 2

- ○ 7
- ○ 15
- ○ 2
- ○ 8
- [✓ ○ 16]
- ○ 4

Question?

How many strings does the following grammar generate?

$$A \rightarrow B\ B$$
$$B \rightarrow C\ C$$
$$C \rightarrow 1 \mid 2 \mid \varepsilon$$

- ○ 16
- ○ 31
- ○ 15
- ○ 12
- ○ 64
- ○ 63
- ○ 32
- ○ 11

Answer!

How many strings does the following grammar generate?

$A \rightarrow B\ B$
$B \rightarrow C\ C$
$C \rightarrow 1\,|\,2\,|\,\varepsilon$

○ 16
[✓ ○ 31]
○ 15
○ 12
○ 64
○ 63
○ 32
○ 11

Question?

Which of the following is a valid derivation of the given grammar?

$S \rightarrow aXa$
$X \rightarrow \varepsilon \mid bY$
$Y \rightarrow \varepsilon \mid cXc \mid d$

○ S
aXa
abYa
acXca
acca

○ S
aa

○ S
aXa
abYa
abcXca
abcbYca
abcbdca

○ S
aXa
abYa
abcXcda
abccda

Answer!

Which of the following is a valid derivation of the given grammar?

S → aXa
X → ε | bY
Y → ε | cXc | d

○ S
aXa
abYa
acXca
acca

○ S
aa

[✓ ○ S
aXa
abYa
abcXca
abcbYca
abcbdca]

○ S
aXa
abYa
abcXcda
abccda

Derivation:

S → aXa → abYa
→ abcXca → abcbYca
→ abcbdca

Question?

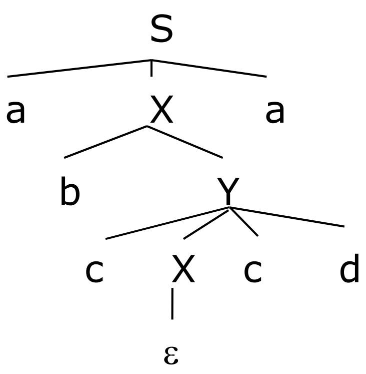
Which of the following is a valid parse tree for the given grammar?

S → aXa

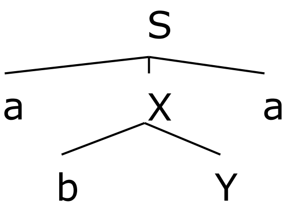
X → ε | bY

Y → ε | cXc | d

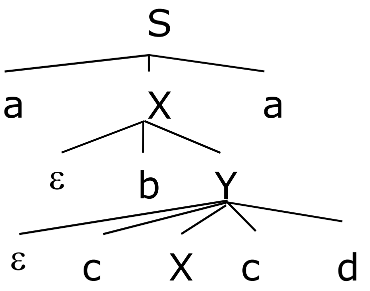
○



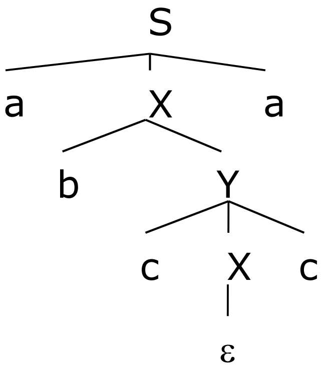
○



○



○



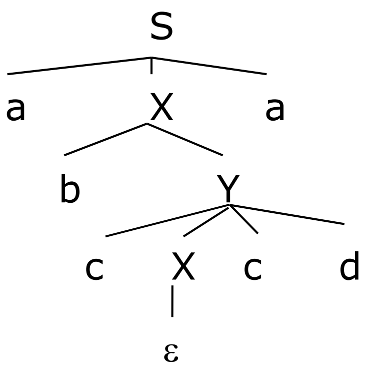
Answer!

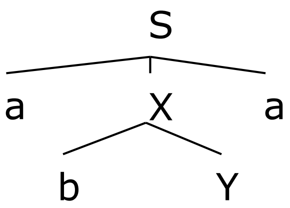
Which of the following is a valid parse tree for the given grammar?

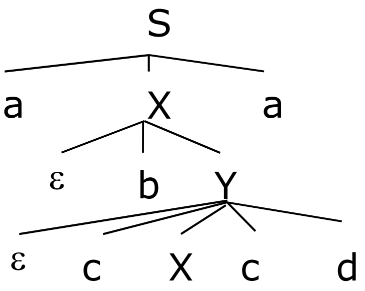
S → aXa

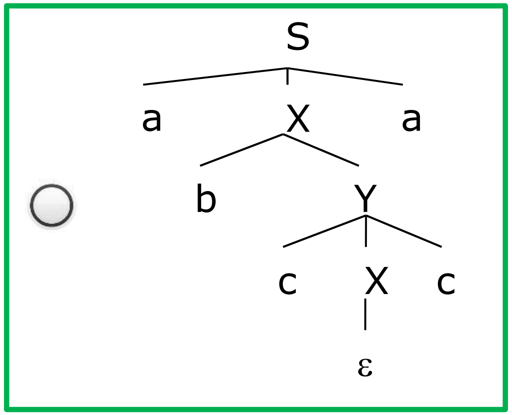
X → ε | bY

Y → ε | cXc | d

○ 

○ 

○ 

○ 

Question?

Choose the grammar that correctly eliminates left recursion from the given grammar: $E \rightarrow E + T \mid T$

$T \rightarrow id \mid (E)$

○ $E \rightarrow E + id \mid E + (E)$

$\mid id \mid (E)$

○ $E \rightarrow TE'$

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

$T \rightarrow id \mid (E)$

○ $E \rightarrow E' + T \mid T$

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

$T \rightarrow id \mid (E)$

○ $E \rightarrow id + E \mid E + T \mid T$

$T \rightarrow id \mid (E)$

Answer!

Choose the grammar that correctly eliminates left recursion from the given grammar: $E \rightarrow E + T \mid T$

$T \rightarrow id \mid (E)$

○ $E \rightarrow E + id \mid E + (E) \mid id \mid (E)$

○ [✓ $E \rightarrow TE'$
$E' \rightarrow + TE' \mid \varepsilon$
$T \rightarrow id \mid (E)$]

○ $E \rightarrow E' + T \mid T$
$E' \rightarrow id \mid (E)$
$T \rightarrow id \mid (E)$

○ $E \rightarrow id + E \mid E + T \mid T$
$T \rightarrow id \mid (E)$

Question?

Consider the following grammar. Adding which one of the following rules will cause the grammar to be left-recursive? [Choose all that apply]

$S \rightarrow A$

$A \rightarrow B | C$

$B \rightarrow (C)$

$C \rightarrow B + C | D$

$D \rightarrow 1 | 0$

- ○ $D \rightarrow A$
- ○ $A \rightarrow D$
- ○ $B \rightarrow C$
- ○ $D \rightarrow B$
- ○ $C \rightarrow 1\ C$

Answer!

Consider the following grammar. Adding which one of the following rules will cause the grammar to be left-recursive? [Choose all that apply]

S → A

A → B | C

B → (C)

C → B + C | D

D → 1 | 0

- [✓ ○ D → A]
- ○ A → D
- [✓ ○ B → C]
- ○ D → B
- ○ C → 1 C

Question?

Which of the following grammars are ambiguous?

□ $S \to SS \mid a \mid b$

□ $E \to E + E \mid id$

□ $S \to Sa \mid Sb$

□ $E \to E' \mid E' + E$

$E' \to -E' \mid id \mid (E)$

Answer!

Which of the following grammars are ambiguous?

- □ $S \to SS \mid a \mid b$
- □ $E \to E + E \mid id$
- □ $S \to Sa \mid Sb$
- □ $E \to E' \mid E' + E$
  $E' \to -E' \mid id \mid (E)$

Question?

Choose the unambiguous version of the given ambiguous grammar: $S \rightarrow SS \mid a \mid b \mid \varepsilon$

○ $S \rightarrow Sa \mid Sb \mid \varepsilon$

○ $S \rightarrow SS'$
$S' \rightarrow a \mid b$

○ $S \rightarrow S \mid S'$
$S' \rightarrow a \mid b$

○ $S \rightarrow Sa \mid Sb$

Answer!

Choose the unambiguous version
of the given ambiguous grammar: $S \rightarrow SS \mid a \mid b \mid \varepsilon$

[✓ ○ $S \rightarrow Sa \mid Sb \mid \varepsilon$]

○ $S \rightarrow SS'$
$S' \rightarrow a \mid b$

○ $S \rightarrow S \mid S'$
$S' \rightarrow a \mid b$

○ $S \rightarrow Sa \mid Sb$

Question?

Consider the following grammar. How many unique parse trees are there for the string 5 * 3 + (2 * 7) + 4?

○ 2

○ 1

○ 7

○ 8

○ 5

○ 4

E → E * E | E + E | (E) | int

Answer!

Consider the following grammar. How many unique parse trees are there for the string 5 * 3 + (2 * 7) + 4?

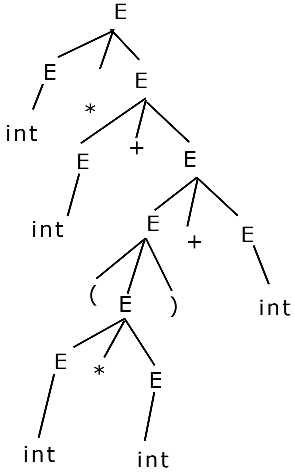
- ○ 2
- ○ 1
- ○ 7
- ○ 8
- [✓ ○ 5]
- ○ 4

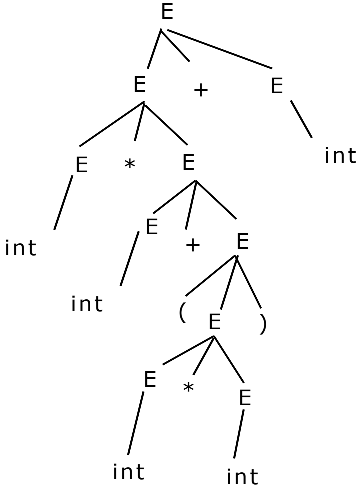
E → E * E | E + E | (E) | int

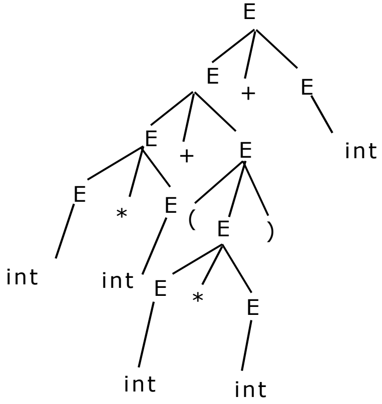
Answer!

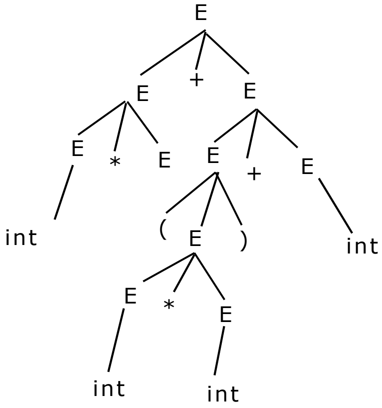
E→ E * E | E + E | (E) | int

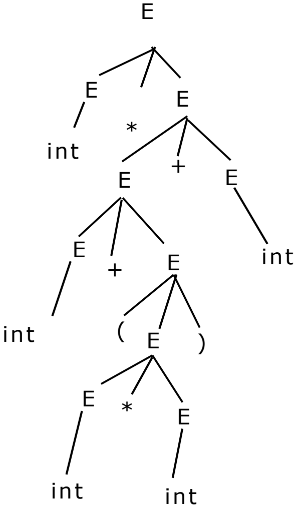
5 * 3 + (2 * 7) + 4











Question?

Which of the following statements are true about the given grammar?

$S \rightarrow a\ T\ U\ b \mid \varepsilon$
$T \rightarrow c\ U\ c \mid b\ U\ b \mid a\ U\ a$
$U \rightarrow S\ b \mid c\ c$

Choose all that are correct.

○ The follow set of S is { $, b }

○ The first set of U is { a, b, c }

○ The first set of S is {ε, a, b }

○ The follow set of T is { a, b, c }

Answer!

Which of the following statements are true about the given grammar?

$S \rightarrow a\ T\ U\ b \mid \varepsilon$
$T \rightarrow c\ U\ c \mid b\ U\ b \mid a\ U\ a$
$U \rightarrow S\ b \mid c\ c$

Choose all that are correct.

- ○ [✓ The follow set of S is { \$, b }]
- ○ [✓ The first set of U is { a, b, c }]
- ○ The first set of S is {ε, a, b }
- ○ [✓ The follow set of T is { a, b, c }]

Question?

Consider the following grammar:

S→ A (S) B | ε
A→ S | S B x | ε
B → S B | y

What are the first and follow sets of S

○ First: { x, y, (, ε} Follow: { y, x, (,) }

○ First: { x, ε} Follow: { $, y, x, (,) }

○ First: { y, (, ε} Follow: { $, y, (,)}

○ First: { x, y, (, ε} Follow: { $, y, x, (,) }

○ First: { x, y, (} Follow: { $, y, x, (,) }

○ First: { x, (} Follow: { $, y, x }

Answer!

Consider the following grammar:

S → A (S) B | ε
A → S | S B x | ε
B → S B | y

What are the first and follow sets of S

○ First: { x, y, (, ε } Follow: { y, x, (,) }

○ First: { x, ε } Follow: { $, y, x, (,) }

[✓ ○ First: { y, (, ε } Follow: { $, y, (,)}]

○ First: { x, y, (, ε } Follow: { $, y, x, (,) }

○ First: { x, y, (} Follow: { $, y, x, (,) }

○ First: { x, (} Follow: { $, y, x }

Question?

Choose the derivation that is a valid recursive descent parse for the string id + id in the given grammar. Moves that are followed by backtracking are given in red.

E → E’ | E’ + E

E’ → - E’ | id | (E)

○

E
E’
E’ + E
id + E
id + E’
id +id

○

E
E’
-E’
id
(E)
E’ + E
-E’ + E
id + E
id + E’
id + -E’
id + id

○

E
E’ + E
id + E
id + E’
id +id

[✓ ○]

E
E’
id
E’ + E
id + E
id + E’
id +id

Answer!

Choose the derivation that is a valid recursive descent parse for the string id + id in the given grammar. Moves that are followed by backtracking are given in red.

E → E’ | E’ + E

E’ → - E’ | id | (E)

○

E
E’
E’ + E
id + E
id + E’
id +id

○

E
E’
-E’
id
(E)
E’ + E
-E’ + E
id + E
id + E’
id + -E’
id + id

○

E
E’ + E
id + E
id + E’
id +id

○

E
E’
id
E’ + E
id + E
id + E’
id +id

Question?

Choose the alternative that correctly left factors “if” statements in the given grammar

```
EXPR →   if BOOL then { EXPR }
       | if BOOL then { EXPR } else { EXPR }
       | …
BOOL →   true | false
```

○
```
EXPR →   if true then { EXPR }
       | if false then { EXPR }
       | if true then { EXPR } else { EXPR }
       | if false then { EXPR } else { EXPR }
       | …
```

○
```
EXPR →   EXPR' | EXPR' else { EXPR }
EXPR' →  if BOOL then { EXPR }
       | …
BOOL →   true | false
```

○
```
EXPR →   if BOOL EXPR'
       | …
EXPR' →  then { EXPR }
       | then { EXPR } else { EXPR }
BOOL →   true | false
```

○
```
EXPR →   if BOOL then { EXPR } EXPR'
       | …
EXPR' →  else { EXPR } | ε
BOOL →   true | false
```

Answer!

Choose the alternative that correctly left factors "if" statements in the given grammar

EXPR → if BOOL then { EXPR }
| if BOOL then { EXPR } else { EXPR }
| …
BOOL → true | false

○
EXPR → if true then { EXPR }
| if false then { EXPR }
| if true then { EXPR } else { EXPR }
| if false then { EXPR } else { EXPR }
| …

○
EXPR → EXPR' | EXPR' else { EXPR }
EXPR' → if BOOL then { EXPR }
| …
BOOL → true | false

○
EXPR → if BOOL EXPR'
| …
EXPR' → then { EXPR }
| then { EXPR } else { EXPR }
BOOL → true | false

[✓ ○
EXPR → if BOOL then { EXPR } EXPR'
| …
EXPR' → else { EXPR } | ε
BOOL → true | false]