

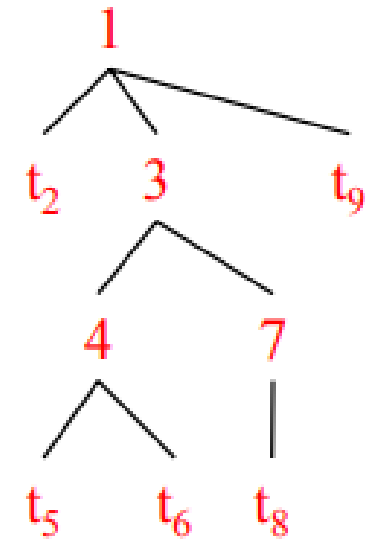


# Top-down Parsing

Professor: Suman Saha

# Top-down Parser

- The parse tree is constructed
  - From the top
  - From left to right
- Terminal are seen in order of appearance in the token stream:
  - $t_2 t_5 t_6 t_8 t_9$
- Top-down parsing methods
  - Recursive descent
  - Predictive parsing



# Recursive Descent Parsing



- It is built from a set of mutually recursive procedures (or a non-recursive equivalent) where each such procedure implements one of the nonterminals of the grammar.
- Recursive descent parsing suffers from backtracking
  - **Backtracking**: It means one derivation of a production fails; the syntax analyzer restart the process using different rules of same production.
  - This technique may process the input string more than once to determine the right production

# Backtracking

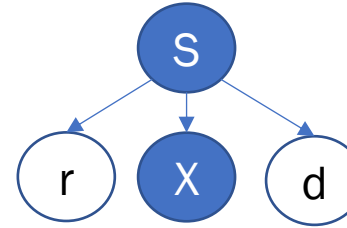
- Say, we have grammar

$$S \rightarrow rXd \mid rZd$$
$$X \rightarrow oa \mid ea$$
$$Z \rightarrow ai$$

- For input string read

# Backtracking

- Say, we have grammar

$$S \rightarrow rXd \mid rZd$$
$$X \rightarrow oa \mid ea$$
$$Z \rightarrow ai$$


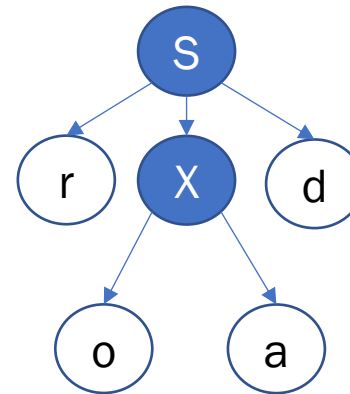
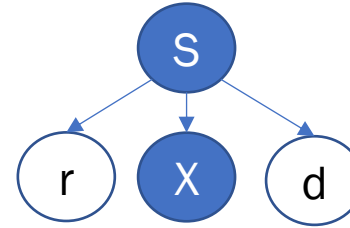
- For input string read

# Backtracking

- Say, we have grammar

$$S \rightarrow rXd \mid rZd$$
$$X \rightarrow oa \mid ea$$
$$Z \rightarrow ai$$

- For input string read

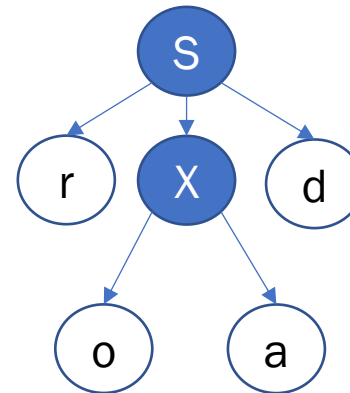
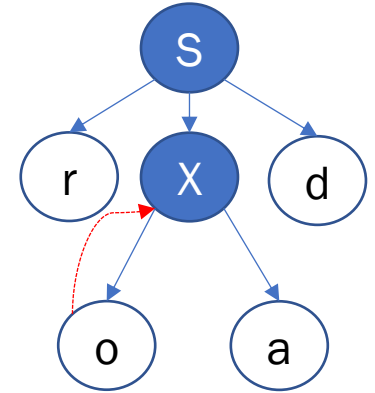
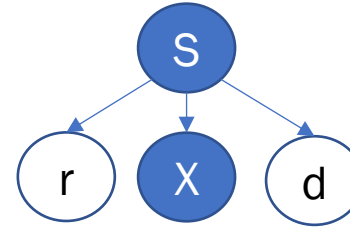


# Backtracking

- Say, we have grammar

$$S \rightarrow rXd \mid rZd$$
$$X \rightarrow oa \mid ea$$
$$Z \rightarrow ai$$

- For input string read

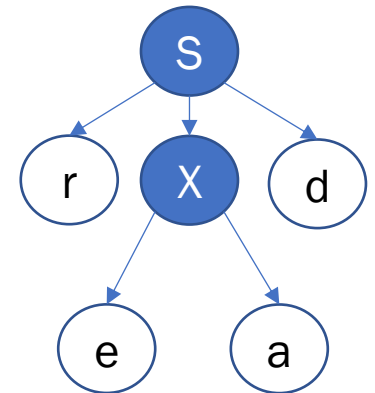
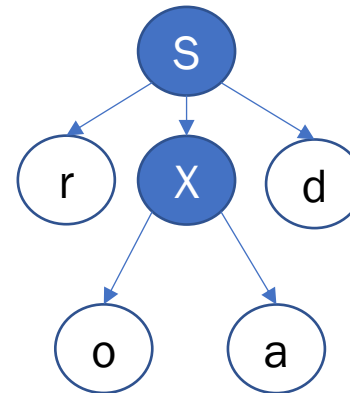
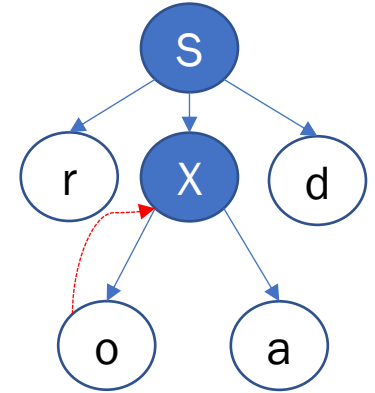
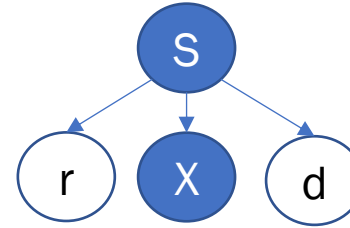


# Backtracking

- Say, we have grammar

$$S \rightarrow rXd \mid rZd$$
$$X \rightarrow oa \mid ea$$
$$Z \rightarrow ai$$

- For input string read





# Recursive Descent Parsing



- Left-recursive grammar has a non-terminal  $S$

$$S \rightarrow^+ S\alpha \text{ for some } \alpha$$

- Recursive descent does not work in such case. the starting symbol is repeated immediately after the arrow; making the derivation refer to itself infinitely times.
- While the parser may be partially correct, it may not terminate.

# Elimination of Left Recursion

- Consider the left-recursive grammar

$$S \rightarrow S \alpha \mid \beta$$

- $S$  generates all string starting with a  $\beta$  and followed by a number of  $\alpha$
- Can rewrite using right-recursion

$$\begin{aligned} S &\rightarrow \beta S' \\ S' &\rightarrow \alpha S' \mid \epsilon \end{aligned}$$

# Elimination of Left Recursion

- The grammar

$$\begin{aligned} S &\rightarrow A \alpha \mid \delta \\ A &\rightarrow S \beta \end{aligned}$$

is also left-recursive because

$$S \rightarrow^+ S \beta \alpha$$

- This left-recursion can also be eliminated? (Exercise)

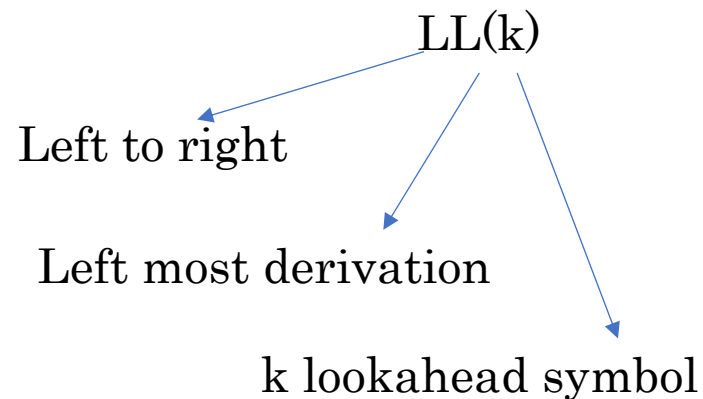
# Predictive Parsing



- It is like a recursive descent parser does not have left recursion but has the capability to predict which production is to be used to replace the input string. It does not suffer from backtracking.
- It uses a look-ahead pointer, which points to the next input symbols. To make the parser back-tracking free, the predictive parser accepts only a class of grammar known as **LL(k) grammar**
  - LL(k) grammar is a subset of context-free grammar

# Predictive Parsing

- It is like a recursive descent parser does not have left recursion but has the capability to predict which production is to be used to replace the input string. It does not suffer from backtracking.
- It uses a look-ahead pointer, which points to the next input symbols. To make the parser back-tracking free, the predictive parser accepts only a class of grammar known as **LL(k) grammar**
  - LL(k) grammar is a subset of context-free grammar



# Predictive Parsing and Left Factoring



- Given grammar

$$\begin{aligned} E &\rightarrow T + E \mid T \\ T &\rightarrow \text{int} \mid \text{int} * T \mid ( E ) \end{aligned}$$

- Hard to predict because
  - For  $T$  two productions start with  $\text{int}$
  - For  $E$  it is not clear how to predict
- A grammar must be left-factored before use for predictive parsing

# Predictive Parsing and Left Factoring



- Given grammar

$$\begin{aligned} E &\rightarrow T + E \mid T \\ T &\rightarrow \text{int} \mid \text{int} * T \mid ( E ) \end{aligned}$$

- Factor out common prefixes of productions

$$\begin{aligned} E &\rightarrow T X \\ X &\rightarrow + E \mid \varepsilon \\ T &\rightarrow ( E ) \mid \text{int} Y \\ Y &\rightarrow * T \mid \varepsilon \end{aligned}$$





# LL(1) Parsing Table

- LL(1) means that for each non-terminal and token there is only one production
- Can be specified via 2D tables
  - One dimension for current non-terminal to expand
  - One dimension for next token
  - A table entry contains one production (**We will talk about how to find table entry later**)

## Left-factored Grammar

$$\begin{aligned}
 E &\rightarrow T X \\
 X &\rightarrow + E \mid \epsilon \\
 T &\rightarrow ( E ) \mid \text{int } Y \\
 Y &\rightarrow * T \mid \epsilon
 \end{aligned}$$

	int	*	+	(	)	\$
E	<b><math>T X</math></b>			<b><math>T X</math></b>		
X			<b><math>+ E</math></b>		<b><math>\epsilon</math></b>	<b><math>\epsilon</math></b>
T	<b><math>\text{int } Y</math></b>			<b><math>( E )</math></b>		
Y		<b><math>* T</math></b>	<b><math>\epsilon</math></b>		<b><math>\epsilon</math></b>	<b><math>\epsilon</math></b>

# LL(1) Parsing Table

	int	*	+	(	)	\$
E	$TX$			$TX$		
X			$+E$		$\epsilon$	$\epsilon$
T	$int Y$			$(E)$		
Y		$*T$	$\epsilon$		$\epsilon$	$\epsilon$

## Left-factored Grammar

$E \rightarrow TX$

$X \rightarrow +E \mid \epsilon$

$T \rightarrow (E) \mid int Y$

$Y \rightarrow *T \mid \epsilon$

- Consider the  $[E, int]$  entry
  - When current non-terminal is  $E$  and next input is  $int$ , use production  $E \rightarrow TX$

# LL(1) Parsing Table

	int	*	+	(	)	\$
E	$TX$			$TX$		
X			$+E$		$\epsilon$	$\epsilon$
T	$int Y$			$(E)$		
Y		$*T$	$\epsilon$		$\epsilon$	$\epsilon$

## Left-factored Grammar

$E \rightarrow TX$

$X \rightarrow +E \mid \epsilon$

$T \rightarrow (E) \mid int Y$

$Y \rightarrow *T \mid \epsilon$

- Consider the  $[Y,+]$  entry
  - When current non-terminal is  $Y$  and current token is  $+$ , get rid of  $Y$

# LL(1) Parsing Table

	int	*	+	(	)	\$
E	$TX$			$TX$		
X			$+E$		$\epsilon$	$\epsilon$
T	$\text{int } Y$			$(E)$		
Y		$*T$	$\epsilon$		$\epsilon$	$\epsilon$

## Left-factored Grammar

$E \rightarrow TX$

$X \rightarrow +E \mid \epsilon$

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

$Y \rightarrow *T \mid \epsilon$

- Blank entries indicate error situations
  - Consider the  $[E, *]$  entry
  - There is no way to derive a string starting with  $*$  from non-terminal  $E$

# LL(1) Parsing Table

- We use a stack to keep track of pending non-terminals
- We reject when we encounter an error state
- We accept when we encounter end-of-input

	int	*	+	(	)	\$
E	<b>T X</b>			<b>T X</b>		
X			<b>+ E</b>		$\epsilon$	$\epsilon$
T	<b>int Y</b>			<b>( E )</b>		
Y		<b>* T</b>	$\epsilon$		$\epsilon$	$\epsilon$

Stack	Input	Action
E \$	int * int \$	T X
T X \$	int * int \$	int Y
int Y X \$	int * int \$	terminal
Y X \$	* int \$	* T
* T X \$	* int \$	terminal
T X \$	int \$	int Y
int Y X \$	int \$	terminal
Y X \$	\$	$\epsilon$
X \$	\$	$\epsilon$
\$	\$	ACCEPT

# FIRST() and FOLLOW()

- FIRST(C)** for a grammar symbol C is the set of terminals that begin the strings derivable from C

$$\text{First}(C) = \{ t \mid C \xrightarrow{*} t\alpha \} \cup \{ \epsilon \mid C \xrightarrow{*} \epsilon \}$$

- First of terminal is terminal itself

- $\text{FIRST}(E) = \{ (, \text{int} \}$
- $\text{FIRST}(X) = \{ +, \epsilon \}$
- $\text{FIRST}(T) = \{ (, \text{int} \}$
- $\text{FIRST}(Y) = \{ *, \epsilon \}$

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

$$X \rightarrow +E \mid \epsilon$$

## Left-factored Grammar

$$\begin{aligned} E &\rightarrow TX \\ X &\rightarrow +E \mid \epsilon \\ T &\rightarrow (E) \mid \text{int } Y \\ Y &\rightarrow *T \mid \epsilon \end{aligned}$$

First (E)

$$E \rightarrow TX$$

$$\rightarrow (E)X$$

$$\rightarrow \text{int } YX$$

# FIRST() and FOLLOW()

• **Follow(C)** to be the set of terminals that can appear immediately to the right of Non-Terminal C in some sentential form.

## Rules:

- If C is the start symbol then  $\$ \in \text{Follow}(C)$
- If  $C \rightarrow A B$  then  $\text{First}(B) \subseteq \text{Follow}(A)$  and  $\text{Follow}(C) \subseteq \text{Follow}(B)$
- Also, if  $B \rightarrow^* \epsilon$  then  $\text{Follow}(C) \subseteq \text{Follow}(A)$

- $\text{FOLLOW}(E) = \{ ), \$ \}$
- $\text{FOLLOW}(X) = \{ ), \$ \}$
- $\text{FOLLOW}(T) = \{ +, ), \$ \}$
- $\text{FOLLOW}(Y) = \{ +, ), \$ \}$

$\text{Follow}(T)$

$T \rightarrow \text{int } Y$

$C \rightarrow A B$

$\text{Follow}(C) \subseteq \text{Follow}(B)$

$\text{Follow}(T) \subseteq \text{Follow}(Y)$

### Left-factored Grammar

$E \rightarrow T X$   
 $X \rightarrow + E \mid \epsilon$   
 $T \rightarrow ( E ) \mid \text{int } Y$   
 $Y \rightarrow * T \mid \epsilon$

$\text{FIRST}(E) = \{ (, \text{int} \}$   
 $\text{FIRST}(X) = \{ +, \epsilon \}$   
 $\text{FIRST}(T) = \{ (, \text{int} \}$   
 $\text{FIRST}(Y) = \{ *, \epsilon \}$

# FIRST() and FOLLOW()

- **Follow(C)** to be the set of terminals that can appear immediately to the right of Non-Terminal C in some sentential form.

- Rules:

- If C is the start symbol then  $\$ \in \text{Follow}(C)$
- If  $C \rightarrow A B$  then  $\text{First}(B) \subseteq \text{Follow}(A)$  and  $\text{Follow}(C) \subseteq \text{Follow}(B)$
- Also, if  $B \rightarrow^* \epsilon$  then  $\text{Follow}(C) \subseteq \text{Follow}(A)$

- $\text{FOLLOW}(E) = \{ \text{), } \$ \}$
- $\text{FOLLOW}(X) = \{ \text{), } \$ \}$
- $\text{FOLLOW}(T) = \{ +, \text{), } \$ \}$
- $\text{FOLLOW}(Y) = \{ +, \text{), } \$ \}$

$\text{Follow}(E)$

$\text{Follow}(T)$

$E \rightarrow TX$

$C \rightarrow AB$

$\text{Follow}(C) \subseteq \text{Follow}(B)$

$\text{Follow}(E) \subseteq \text{Follow}(X)$

Left-factored Grammar	
$E \rightarrow TX$	✓
$X \rightarrow +E \mid \epsilon$	
$T \rightarrow (E) \mid \text{int } Y$	
$Y \rightarrow *T \mid \epsilon$	

$\text{FIRST}(E) = \{ (, \text{int} \}$
$\text{FIRST}(X) = \{ +, \epsilon \}$
$\text{FIRST}(T) = \{ (, \text{int} \}$
$\text{FIRST}(Y) = \{ *, \epsilon \}$



# FIRST() and FOLLOW()

- **Follow(C)** to be the set of terminals that can appear immediately to the right of Non-Terminal C in some sentential form.

- Rules:

- If C is the start symbol then  $\$ \in \text{Follow}(C)$
- If  $C \rightarrow A B$  then  $\text{First}(B) \subseteq \text{Follow}(A)$  and  $\text{Follow}(C) \subseteq \text{Follow}(B)$
- Also, if  $B \rightarrow^* \epsilon$  then  $\text{Follow}(C) \subseteq \text{Follow}(A)$

$F \rightarrow TX$   
 $C \rightarrow AB$   
 $\text{First}(B) \subseteq \text{Follow}(A)$   
 $\text{First}(X) \subseteq \text{Follow}(T)$

- $\text{FOLLOW}(E) = \{ ), \$ \}$
- $\text{FOLLOW}(X) = \{ ), \$ \}$
- $\text{FOLLOW}(T) = \{ +, ), \$ \}$
- $\text{FOLLOW}(Y) = \{ +, ), \$ \}$

$\text{Follow}(E)$   
 $\{ + \} \cup \text{Follow}(E)$   
 $\text{Follow}(T)$

$Y \rightarrow * T$   
 $A \rightarrow AB$   
 $\text{Follow}(C) \subseteq \text{Follow}(B)$   
 $\text{Follow}(Y) \subseteq \text{Follow}(T)$   
 $\text{Follow}(Y)$

Left-factored Grammar

$E \rightarrow TX$   
 $X \rightarrow +E \mid \epsilon$   
 $T \rightarrow (E) \mid \text{int } Y$   
 $Y \rightarrow *T \mid \epsilon$

$\text{FIRST}(E) = \{ (, \text{int} \}$   
 $\text{FIRST}(X) = \{ +, \epsilon \}$   
 $\text{FIRST}(T) = \{ (, \text{int} \}$   
 $\text{FIRST}(Y) = \{ *, \epsilon \}$

# FIRST() and FOLLOW()

- Follow(C) to be the set of terminals that can appear immediately to the right of Non-Terminal C in some sentential form.

- Rules:

- If C is the start symbol then  $\$ \in \text{Follow}(C)$
- If  $C \rightarrow A B$  then  $\text{First}(B) \subseteq \text{Follow}(A)$  and  $\text{Follow}(C) \subseteq \text{Follow}(B)$
- Also, if  $B \rightarrow^* \epsilon$  then  $\text{Follow}(C) \subseteq \text{Follow}(A)$

$X \rightarrow +E$   
 $C \rightarrow AB$   
 $\text{Follow}(X) \supseteq \text{Follow}(E)$

$T \rightarrow (E)$

- FOLLOW(E) = { }, \$
- FOLLOW(X) = { }, \$
- FOLLOW(T) = { +, ), \$ }
- FOLLOW(Y) = { +, ), \$ }

$\{ \$ \} \cup \text{follow}(X) \cup \{ \epsilon \}$   
 $\text{Follow}(E)$   
 $\{ + \} \cup \text{Follow}(E) \cup \text{Follow}(T)$   
 $\text{Follow}(T)$

## Left-factored Grammar

$E \rightarrow T X$   
 $X \rightarrow + E \mid \epsilon$   
 $T \rightarrow (E) \mid \text{int } Y$   
 $Y \rightarrow * T \mid \epsilon$

$\text{FIRST}(E) = \{ (, \text{int} \}$   
 $\text{FIRST}(X) = \{ +, \epsilon \}$   
 $\text{FIRST}(T) = \{ (, \text{int} \}$   
 $\text{FIRST}(Y) = \{ *, \epsilon \}$

# Constructing LL(1) Parsing Table



PennState

- Construct a parsing table  $T$  for CFG  $G$
- For each production  $A \rightarrow \alpha$  in  $G$  do:
  - For each terminal  $t \in \text{First}(\alpha)$  do
    - $T[A, t] = \alpha$
  - If  $\epsilon \in \text{First}(\alpha)$ , for each  $t \in \text{Follow}(A)$  do
    - $T[A, t] = \alpha$
  - If  $\epsilon \in \text{First}(\alpha)$  and  $\$ \in \text{Follow}(A)$  do
    - $T[A, \$] = \alpha$

## Left-factored Grammar

$E \rightarrow T X$   
 $X \rightarrow + E \mid \epsilon$   
 $T \rightarrow ( E ) \mid \text{int } Y$   
 $Y \rightarrow * T \mid \epsilon$

$\text{FOLLOW}(E) = \{ ), \$ \}$   
 $\text{FOLLOW}(X) = \{ ), \$ \}$   
 $\text{FOLLOW}(T) = \{ +, ), \$ \}$   
 $\text{FOLLOW}(Y) = \{ +, ), \$ \}$

$\text{FIRST}(E) = \{ (, \text{int} \}$   
 $\text{FIRST}(X) = \{ +, \epsilon \}$   
 $\text{FIRST}(T) = \{ (, \text{int} \}$   
 $\text{FIRST}(Y) = \{ *, \epsilon \}$

	int	*	+	(	)	\$
E	$T X$			$T X$		
X			$+ E$		$\epsilon$	$\epsilon$
T	$\text{int } Y$			$( E )$		
Y		$* T$	$\epsilon$		$\epsilon$	$\epsilon$

# Reading and Exercises



## Reading

- Chapter: 2.3.1, 2.3.2, and 2.3.3 (Michael Scott Book)

## Exercises

- Exercises: 2.11 - 2.26 (Michael Scott Book)