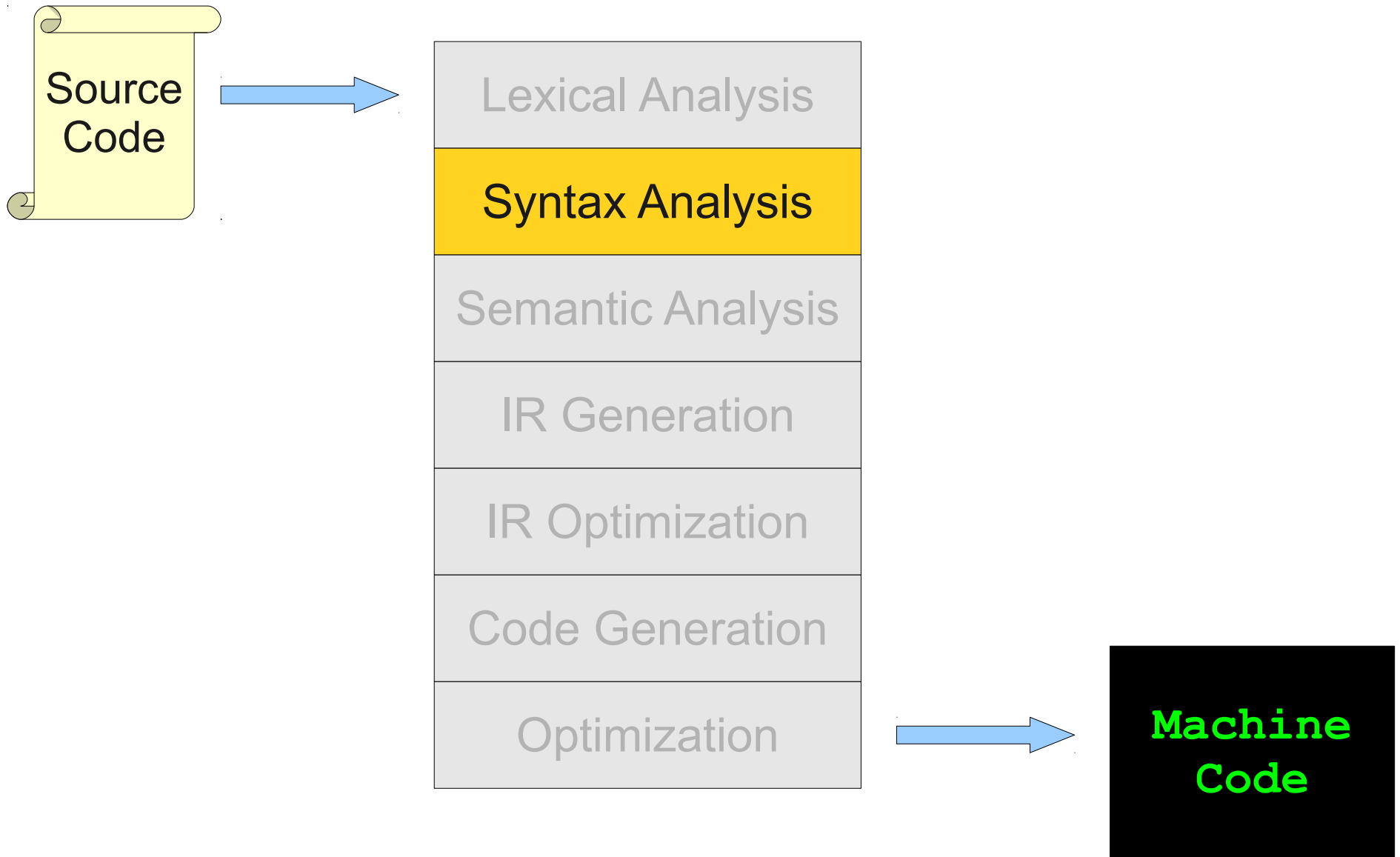


# Top-Down Parsing, Part II

# Announcements

- Programming Project 1 due **Friday, 11:59PM**
- Office hours every day until then.
- Submission instructions will be emailed out tonight.

# Where We Are



# Review of LL(1)

- Left-to-right scan, Leftmost derivation, 1 token lookahead.
- Predict which production to use based on the current nonterminal and the lookahead token.
- Build an **LL(1) parse table** to make these lookups fast and streamlined.

# Review of FIRST Sets

- The set **FIRST(A)** is defined as the set of terminals that could be at the start of a string ultimately derived from A.
  - Formally:  $\text{FIRST}(A) = \{ t \mid A \rightarrow^* tv \}$
- Can compute iteratively:
  - Set  $\text{FIRST}(A) = \{ t \mid A \rightarrow tv \}$
  - Keep computing  $\text{FIRST}(A) = \text{FIRST}(A) \cup \text{FIRST}(B)$  for all productions  $A \rightarrow wBv$ , where  $w$  is a string of nonterminals that can derive  $\epsilon$ .

# Review of FOLLOW Sets

- The set **FOLLOW(A)** is defined as the set of terminals that could potentially follow a string produced from nonterminal A.
- Formally:  $\text{FOLLOW}(A) = \{ t \mid B \rightarrow^* uAtv \}$
- Can also be computed iteratively:
  - Initially, set  $\text{FOLLOW}(A) = \text{FIRST}(w) - \{\epsilon\}$  for all rules  $C \rightarrow vAw$ .
  - Keep computing  $\text{FOLLOW}(A) = \text{FOLLOW}(A) \cup \text{FOLLOW}(B)$  for all rules  $B \rightarrow vAw$  where  $w$  is or can derive  $\epsilon$ .

# LL(1) Table Construction

- Compute  $\text{FIRST}(A)$  and  $\text{FOLLOW}(A)$  for all nonterminals  $A$ .
- For each rule  $A \rightarrow w$  and for each **terminal**  $t$  in  $\text{FIRST}(w)$ , set  $T[A, t] = w$ .
  - Note that  $\epsilon$  is **not** a terminal!
- For each rule  $A \rightarrow w$  with  $\epsilon$  in  $\text{FIRST}(w)$ , set  $T[A, t] = w$  for each  $t$  in  $\text{FOLLOW}(A)$ .

# Example LL(1) Construction

# The Limits of LL(1)

# A Grammar that is Not LL(1)

- Consider the following (left-recursive) grammar:

$$A \rightarrow Ab \mid c$$

- $\text{FIRST}(A) = \{c\}$
- However, we cannot build an LL(1) parse table.
- **Why?**

# A Grammar that is Not LL(1)

- Consider the following (left-recursive) grammar:

$$A \rightarrow Ab \mid c$$

- $\text{FIRST}(A) = \{c\}$
- However, we cannot build an LL(1) parse table.
- **Why?**

	b	c
A		$A \rightarrow Ab$ $A \rightarrow c$

# A Grammar that is Not LL(1)

- Consider the following (left-recursive) grammar:

$$A \rightarrow Ab \mid c$$

- $\text{FIRST}(A) = \{c\}$
- However, we cannot build an LL(1) parse table.
- **Why?**

	b	c
A		$A \rightarrow Ab$ $A \rightarrow c$

- **Cannot uniquely predict production!**
- This is called a **FIRST/FIRST** conflict.

# Eliminating Left Recursion

- In general, left recursion can be converted into **right recursion** by a mechanical transformation.

- Consider the grammar

$$A \rightarrow Av \mid w$$

- This will produce  $w$  followed by some number of  $v$ 's.
- Can rewrite the grammar as

$$A \rightarrow wB$$

$$B \rightarrow \varepsilon \mid vB$$

# Another Non-LL(1) Grammar

- Consider the following grammar:

$S \rightarrow E$

$E \rightarrow T$

$E \rightarrow T + E$

$T \rightarrow \mathbf{int}$

$T \rightarrow (E)$

- $\text{FIRST}(E) = \{ \mathbf{int}, ( \}$
- $\text{FIRST}(T) = \{ \mathbf{int}, ( \}$
- Why is this grammar not LL(1)?

# Another Non-LL(1) Grammar

- Consider the following grammar:

$S \rightarrow E$

$E \rightarrow T$

$E \rightarrow T + E$

$T \rightarrow \text{int}$

$T \rightarrow (E)$

- $\text{FIRST}(E) = \{ \text{int}, ( \}$
- $\text{FIRST}(T) = \{ \text{int}, ( \}$
- Why is this grammar not LL(1)?

How do you  
predict which of  
these to use?

# Left-Factoring

$S \rightarrow E$

$E \rightarrow T$

$E \rightarrow T + E$

$T \rightarrow \text{int}$

$T \rightarrow (E)$

# Left-Factoring

$$S \rightarrow E$$
$$E \rightarrow T\epsilon$$
$$E \rightarrow T + E$$
$$T \rightarrow \mathbf{int}$$
$$T \rightarrow (E)$$

# Left-Factoring

$S \rightarrow E$

$E \rightarrow T\epsilon$

$E \rightarrow T + E$

$T \rightarrow \mathbf{int}$

$T \rightarrow (E)$

$Y \rightarrow + E$

$Y \rightarrow \epsilon$

# Left-Factoring

$S \rightarrow E$

$E \rightarrow TY$

$T \rightarrow \text{int}$

$T \rightarrow (E)$

$Y \rightarrow + E$

$Y \rightarrow \varepsilon$

# Left-Factoring

$S \rightarrow E$

$E \rightarrow TY$

$T \rightarrow \text{int}$

$T \rightarrow (E)$

$Y \rightarrow + E$

$Y \rightarrow \varepsilon$

	int	(	)	+	\$
S					
E					
T					
Y					

# Left-Factoring

$S \rightarrow E$

$E \rightarrow TY$

$T \rightarrow \text{int}$

$T \rightarrow (E)$

$Y \rightarrow + E$

$Y \rightarrow \varepsilon$

FIRST			
S	E	T	Y

	int	(	)	+	\$
S					
E					
T					
Y					

# Left-Factoring

$S \rightarrow E$

$E \rightarrow TY$

$T \rightarrow \text{int}$

$T \rightarrow (E)$

$Y \rightarrow + E$

$Y \rightarrow \epsilon$

FIRST			
S	E	T	Y
		int (	+ $\epsilon$

	int	(	)	+	\$
S					
E					
T					
Y					

# Left-Factoring

$S \rightarrow E$

$E \rightarrow TY$

$T \rightarrow \text{int}$

$T \rightarrow (E)$

$Y \rightarrow + E$

$Y \rightarrow \epsilon$

FIRST			
S	E	T	Y
	int (	int (	+ $\epsilon$

	int	(	)	+	\$
S					
E					
T					
Y					

# Left-Factoring

$S \rightarrow E$

$E \rightarrow TY$

$T \rightarrow \text{int}$

$T \rightarrow (E)$

$Y \rightarrow + E$

$Y \rightarrow \epsilon$

FIRST			
S	E	T	Y
int (	int (	int (	+ $\epsilon$

	int	(	)	+	\$
S					
E					
T					
Y					

# Left-Factoring

$S \rightarrow E$   
 $E \rightarrow TY$   
  
 $T \rightarrow \text{int}$   
 $T \rightarrow (E)$   
 $Y \rightarrow + E$   
 $Y \rightarrow \epsilon$

FIRST			
S	E	T	Y
int (	int (	int (	+ $\epsilon$
FOLLOW			
S	E	T	Y

	int	(	)	+	\$
S					
E					
T					
Y					

# Left-Factoring

$S \rightarrow E$   
 $E \rightarrow TY$   
  
 $T \rightarrow \text{int}$   
 $T \rightarrow (E)$   
 $Y \rightarrow + E$   
 $Y \rightarrow \epsilon$

FIRST			
S	E	T	Y
int (	int (	int (	+ $\epsilon$
FOLLOW			
S	E	T	Y
\$	\$ )		

	int	(	)	+	\$
S					
E					
T					
Y					

# Left-Factoring

$S \rightarrow E$   
 $E \rightarrow TY$   
  
 $T \rightarrow \text{int}$   
 $T \rightarrow (E)$   
 $Y \rightarrow + E$   
 $Y \rightarrow \epsilon$

FIRST			
S	E	T	Y
int (	int (	int (	+ $\epsilon$
FOLLOW			
S	E	T	Y
\$	\$ )	+	

	int	(	)	+	\$
S					
E					
T					
Y					

# Left-Factoring

$S \rightarrow E$   
 $E \rightarrow TY$   
  
 $T \rightarrow \text{int}$   
 $T \rightarrow (E)$   
 $Y \rightarrow + E$   
 $Y \rightarrow \epsilon$

FIRST			
S	E	T	Y
int (	int (	int (	+ $\epsilon$
FOLLOW			
S	E	T	Y
\$	\$ )	+	\$ )

	int	(	)	+	\$
S					
E					
T					
Y					

# Left-Factoring

$S \rightarrow E$   
 $E \rightarrow TY$   
  
 $T \rightarrow \text{int}$   
 $T \rightarrow (E)$   
 $Y \rightarrow + E$   
 $Y \rightarrow \epsilon$

FIRST			
S	E	T	Y
int (	int (	int (	+ $\epsilon$
FOLLOW			
S	E	T	Y
\$	\$ )	+ \$ )	\$ )

	int	(	)	+	\$
S					
E					
T					
Y					

# Left-Factoring

$S \rightarrow E$	1
$E \rightarrow TY$	2
$T \rightarrow \text{int}$	3
$T \rightarrow (E)$	4
$Y \rightarrow + E$	5
$Y \rightarrow \epsilon$	6

FIRST			
S	E	T	Y
int (	int (	int (	+ $\epsilon$
FOLLOW			
S	E	T	Y
\$	\$ )	+ \$ )	\$ )

	int	(	)	+	\$
S					
E					
T					
Y					

# Left-Factoring

$S \rightarrow E$	1
$E \rightarrow TY$	2
$T \rightarrow \text{int}$	3
$T \rightarrow (E)$	4
$Y \rightarrow + E$	5
$Y \rightarrow \epsilon$	6

FIRST			
S	E	T	Y
int (	int (	int (	+ $\epsilon$
FOLLOW			
S	E	T	Y
\$	\$ )	+ \$ )	\$ )

	int	(	)	+	\$
S	1	1			
E					
T					
Y					

# Left-Factoring

$S \rightarrow E$	1
$E \rightarrow TY$	2
$T \rightarrow \text{int}$	3
$T \rightarrow (E)$	4
$Y \rightarrow + E$	5
$Y \rightarrow \epsilon$	6

FIRST			
S	E	T	Y
int	int	int	+
(	(	(	$\epsilon$
FOLLOW			
S	E	T	Y
\$	\$	+	\$
	)	\$	)
		)	

	int	(	)	+	\$
S	1	1			
E	2	2			
T					
Y					

# Left-Factoring

$S \rightarrow E$	1
$E \rightarrow TY$	2
$T \rightarrow \text{int}$	3
$T \rightarrow (E)$	4
$Y \rightarrow + E$	5
$Y \rightarrow \epsilon$	6

FIRST			
S	E	T	Y
int	int	int	+
(	(	(	$\epsilon$
FOLLOW			
S	E	T	Y
\$	\$	+	\$
	)	\$	)
		)	

	int	(	)	+	\$
S	1	1			
E	2	2			
T	3	4			
Y					

# Left-Factoring

$S \rightarrow E$	1
$E \rightarrow TY$	2
$T \rightarrow \text{int}$	3
$T \rightarrow (E)$	4
$Y \rightarrow + E$	5
$Y \rightarrow \epsilon$	6

FIRST			
S	E	T	Y
int (	int (	int (	+ $\epsilon$
FOLLOW			
S	E	T	Y
\$	\$ )	+ \$ )	\$ )

	int	(	)	+	\$
S	1	1			
E	2	2			
T	3	4			
Y				5	

# Left-Factoring

$S \rightarrow E$	1
$E \rightarrow TY$	2
$T \rightarrow \text{int}$	3
$T \rightarrow (E)$	4
$Y \rightarrow + E$	5
$Y \rightarrow \epsilon$	6

FIRST			
S	E	T	Y
int	int	int	+
(	(	(	$\epsilon$
FOLLOW			
S	E	T	Y
\$	\$	+	\$
	)	\$	)
		)	

	int	(	)	+	\$
S	1	1			
E	2	2			
T	3	4			
Y			6	5	6

# The Strengths of LL(1)

# LL(1) is Realistic

- Some real-world programming languages are LL(1)-parsable or parsable with a minor modification on LL(1).
- Examples:
  - LISP
  - Python
  - JavaScript

# LL(1) is Straightforward

- Can be implemented quickly with a table-driven design.
- Can be implemented by **recursive descent**:
  - Define a function for each nonterminal.
  - Have these functions call each other based on the lookahead token.
- See Handout #09 for more details.

# LL(1) is Fast

- Both table-driven LL(1) and recursive-descent-powered LL(1) are fast.
- Can parse in  $O(n f(G))$  time, where  $n$  is the length of the string and  $f(G)$  is some function of the size of the grammar (usually a small constant).

# Summary

- **Top-down parsing** tries to derive the user's program from the start symbol.
- **Leftmost BFS** is one approach to top-down parsing; it is mostly of theoretical interest.
- **Leftmost DFS** is another approach to top-down parsing that is uncommon in practice.
- **LL(1)** parsing scans from left-to-right, using one token of lookahead to find a leftmost derivation.
- **FIRST sets** contain terminals that may be the first symbol of a production.
- **FOLLOW sets** contain terminals that may follow a nonterminal in a production.
- **Left recursion** and **left factoring** cause LL(1) to fail and can be mechanically eliminated.