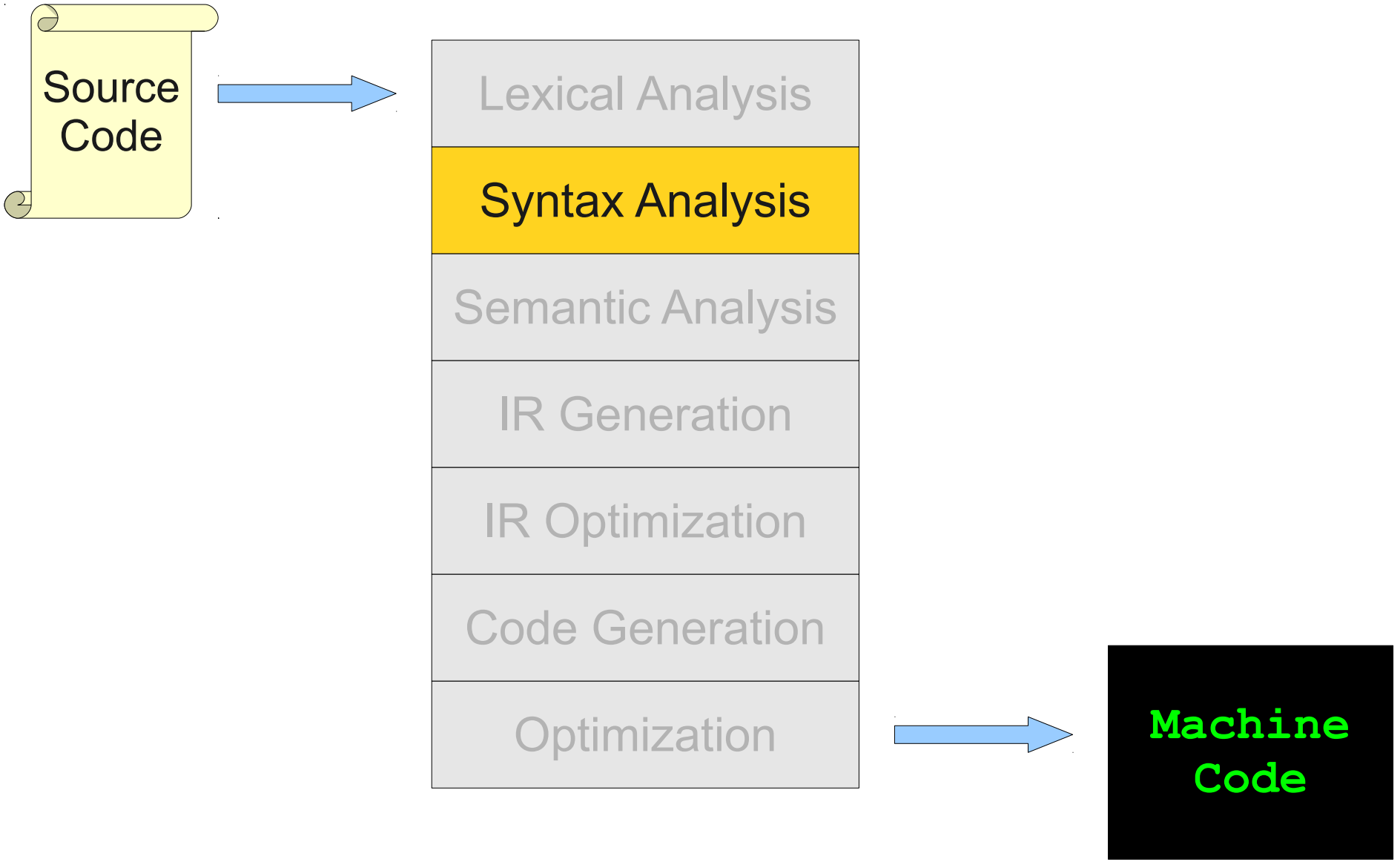
Top-Down Parsing, Part II

Announcements

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

Where We Are


Source Code
Lexical Analysis
Syntax Analysis
Semantic Analysis
IR Generation
IR Optimization
Code Generation
Optimization
Machine Code

Review of LL(1)

- **L**eft-to-right scan, **L**eftmost 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: FIRST(A) = { t | A $\rightarrow^*$ t*v* }
- Can compute iteratively:
  - Set FIRST(A) = { t | A $\rightarrow$ t*v* }
  - Keep computing FIRST(A) = FIRST(A) $\cup$ FIRST(B) for all productions A $\rightarrow$ *w*B*v*, where *w* is a string of nonterminals that can derive ε.

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: FOLLOW(A) = { t | B $\rightarrow^*$ *u*A*t*v* }
- Can also be computed iteratively:
  - Initially, set FOLLOW(A) = FIRST(*w*) – {ε} for all rules C $\rightarrow$ *v*A*w*.
  - Keep computing FOLLOW(A) = FOLLOW(A) ∪ FOLLOW(B) for all rules B $\rightarrow$ *v*A*w* where *w* is or can derive ε.

LL(1) Table Construction

- Compute FIRST(A) and FOLLOW(A) for all nonterminals A.
- For each rule A → *w* and for each **terminal** t in FIRST(*w*), set T[A, t] = w.
  - Note that ε is **not** a terminal!
- For each rule A → *w* with ε in FIRST(*w*), set T[A, t] = *w* for each t in FOLLOW(A).

Example LL(1) Construction

The Limits of LL(1)

A Grammar that is Not LL(1)

- Consider the following (left-recursive) grammar:

  $A \rightarrow Ab \mid c$

- 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:

  $A \rightarrow Ab \mid c$

- 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:

  $A \rightarrow Ab \mid c$

- 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

  $A \to Av \mid w$

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

  $A \to wB$

  $B \to \varepsilon \mid vB$

Another Non-LL(1) Grammar

- Consider the following grammar:

  S → E

  E → T

  E → T **+** E

  T → **`int`**

  T → **`(`**E**`)`**

- FIRST(E) = { **`int`**, **`(`** }
- FIRST(T) = { **`int`**, **`(`** }
- Why is this grammar not LL(1)?

Another Non-LL(1) Grammar

- Consider the following grammar:

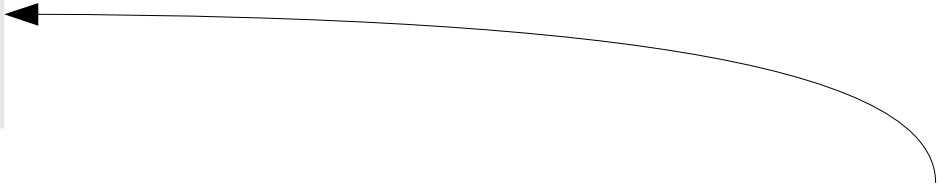
  $S \rightarrow E$

  $E \rightarrow T$

  $E \rightarrow T$ **+** E

  $T \rightarrow$ **int**

  $T \rightarrow$ **(** E **)**



How do you predict which of these to use?

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

Left-Factoring

$S \rightarrow E$

$E \rightarrow T$

$E \rightarrow T + E$

$T \rightarrow$ **`int`**

$T \rightarrow$ `(`E`)`

Left-Factoring

$S \rightarrow E$

$E \rightarrow T\varepsilon$

$E \rightarrow T + E$

$T \rightarrow$ **`int`**

$T \rightarrow$ `(`E`)`

Left-Factoring

$S \rightarrow E$

$E \rightarrow T\varepsilon$

$E \rightarrow T + E$

$T \rightarrow \texttt{int}$

$T \rightarrow (E)$

$Y \rightarrow + E$

$Y \rightarrow \varepsilon$

Left-Factoring

$S \rightarrow E$

$E \rightarrow TY$

$T \rightarrow \texttt{int}$

$T \rightarrow \texttt{(}E\texttt{)}$

$Y \rightarrow \texttt{+}\,E$

$Y \rightarrow \varepsilon$

Left-Factoring

$S \rightarrow E$
$E \rightarrow TY$

$T \rightarrow$ `int`
$T \rightarrow$ `(`E`)`
$Y \rightarrow$ `+` E
$Y \rightarrow \varepsilon$

	`int`	`(`	`)`	`+`	`$`
S					
E					
T					
Y					

Left-Factoring

$S \rightarrow E$
$E \rightarrow TY$

$T \rightarrow$ `int`
$T \rightarrow$ `(`E`)`
$Y \rightarrow$ `+` E
$Y \rightarrow \varepsilon$

FIRST			
S	E	T	Y

	`int`	`(`	`)`	`+`	`$`
S					
E					
T					
Y					

Left-Factoring

$S \rightarrow E$
$E \rightarrow TY$

$T \rightarrow$ `int`
$T \rightarrow$ `(`E`)`
$Y \rightarrow$ `+` E
$Y \rightarrow \varepsilon$

FIRST			
S	E	T	Y
		`int` `(`	`+` ε

	`int`	`(`	`)`	`+`	`$`
S					
E					
T					
Y					

Left-Factoring

$S \rightarrow E$
$E \rightarrow TY$

$T \rightarrow$ `int`
$T \rightarrow$ `(`E`)`
$Y \rightarrow$ `+` E
$Y \rightarrow \varepsilon$

FIRST			
S	E	T	Y
	`int` `(`	`int` `(`	`+` ε

	`int`	`(`	`)`	`+`	`$`
S					
E					
T					
Y					

Left-Factoring

$S \rightarrow E$
$E \rightarrow TY$

$T \rightarrow$ `int`
$T \rightarrow$ `(`E`)`
$Y \rightarrow$ `+` E
$Y \rightarrow \varepsilon$

FIRST			
S	E	T	Y
`int` `(`	`int` `(`	`int` `(`	`+` ε

	`int`	`(`	`)`	`+`	`$`
S					
E					
T					
Y					

Left-Factoring

$S \rightarrow E$
$E \rightarrow TY$

$T \rightarrow \texttt{int}$
$T \rightarrow \texttt{(}E\texttt{)}$
$Y \rightarrow \texttt{+}E$
$Y \rightarrow \varepsilon$

FIRST			
S	E	T	Y
`int` `(`	`int` `(`	`int` `(`	`+` ε

FOLLOW			
S	E	T	Y

	`int`	`(`	`)`	`+`	`$`
S					
E					
T					
Y					

Left-Factoring

$S \rightarrow E$
$E \rightarrow TY$

$T \rightarrow \texttt{int}$
$T \rightarrow (E)$
$Y \rightarrow + E$
$Y \rightarrow \varepsilon$

FIRST			
S	E	T	Y
int (	int (	int (	+ ε

FOLLOW			
S	E	T	Y
$	$)		

	int	(	)	+	$
S					
E					
T					
Y					

Left-Factoring

$S \rightarrow E$
$E \rightarrow TY$

$T \rightarrow$ `int`
$T \rightarrow$ `(`E`)`
$Y \rightarrow$ `+`E
$Y \rightarrow \varepsilon$

FIRST			
S	E	T	Y
`int` `(`	`int` `(`	`int` `(`	`+` ε

FOLLOW			
S	E	T	Y
`$`	`$` `)`	`+`	

	`int`	`(`	`)`	`+`	`$`
S					
E					
T					
Y					

Left-Factoring

$S \rightarrow E$
$E \rightarrow TY$

$T \rightarrow \texttt{int}$
$T \rightarrow (E)$
$Y \rightarrow +E$
$Y \rightarrow \varepsilon$

FIRST			
S	E	T	Y
int (	int (	int (	+ ε

FOLLOW			
S	E	T	Y
$	$)	+	$)

	int	(	)	+	$
S					
E					
T					
Y					

Left-Factoring

$S \rightarrow E$
$E \rightarrow TY$

$T \rightarrow \texttt{int}$
$T \rightarrow (E)$
$Y \rightarrow + E$
$Y \rightarrow \varepsilon$

FIRST			
S	E	T	Y
int (	int (	int (	+ ε

FOLLOW			
S	E	T	Y
$	$)	+ $)	$)

	int	(	)	+	$
S					
E					
T					
Y					

Left-Factoring

S → E	1
E → TY	2
T → `int`	3
T → `(`E`)`	4
Y → `+` E	5
Y → ε	6

FIRST			
S	E	T	Y
`int` `(`	`int` `(`	`int` `(`	`+` ε

FOLLOW			
S	E	T	Y
`$`	`$` `)`	`+` `$` `)`	`$` `)`

	`int`	`(`	`)`	`+`	`$`
S					
E					
T					
Y					

Left-Factoring

S → E	1
E → TY	2
T → `int`	3
T → `(`E`)`	4
Y → `+` E	5
Y → ε	6

FIRST			
S	E	T	Y
`int` `(`	`int` `(`	`int` `(`	`+` ε

FOLLOW			
S	E	T	Y
`$`	`$` `)`	`+` `$` `)`	`$` `)`

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

Left-Factoring

S → E	1
E → TY	2
T → **int**	3
T → **(**E**)**	4
Y → **+** E	5
Y → ε	6

FIRST			
S	E	T	Y
int **(**	**int** **(**	**int** **(**	**+** ε

FOLLOW			
S	E	T	Y
$	**$** **)**	**+** **$** **)**	**$** **)**

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

Left-Factoring

S → E	1
E → TY	2
T → `int`	3
T → (E)	4
Y → + E	5
Y → ε	6

FIRST			
S	E	T	Y
`int` `(`	`int` `(`	`int` `(`	`+` ε

FOLLOW			
S	E	T	Y
`$`	`$` `)`	`+` `$` `)`	`$` `)`

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

Left-Factoring

S → E	1
E → TY	2
T → **int**	3
T → **(**E**)**	4
Y → **+** E	5
Y → ε	6

FIRST			
S	E	T	Y
int **(**	**int** **(**	**int** **(**	**+** ε

FOLLOW			
S	E	T	Y
$	**$** **)**	**+** **$** **)**	**$** **)**

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

Left-Factoring

S → E	1
E → TY	2
T → **int**	3
T → **(**E**)**	4
Y → **+** E	5
Y → ε	6

FIRST			
S	E	T	Y
int **(**	**int** **(**	**int** **(**	**+** ε

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.