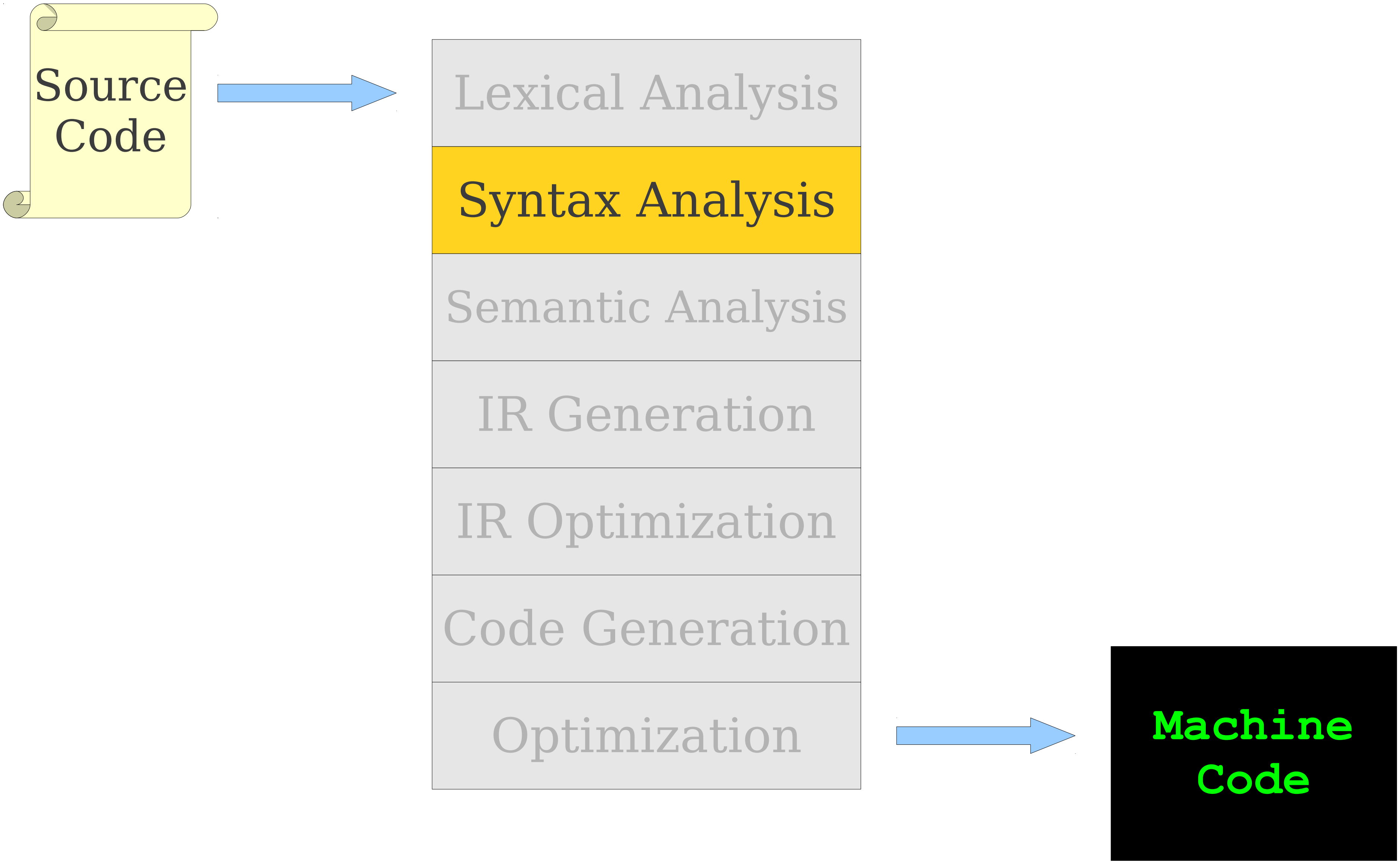
Top-Down Parsing II

Announcements

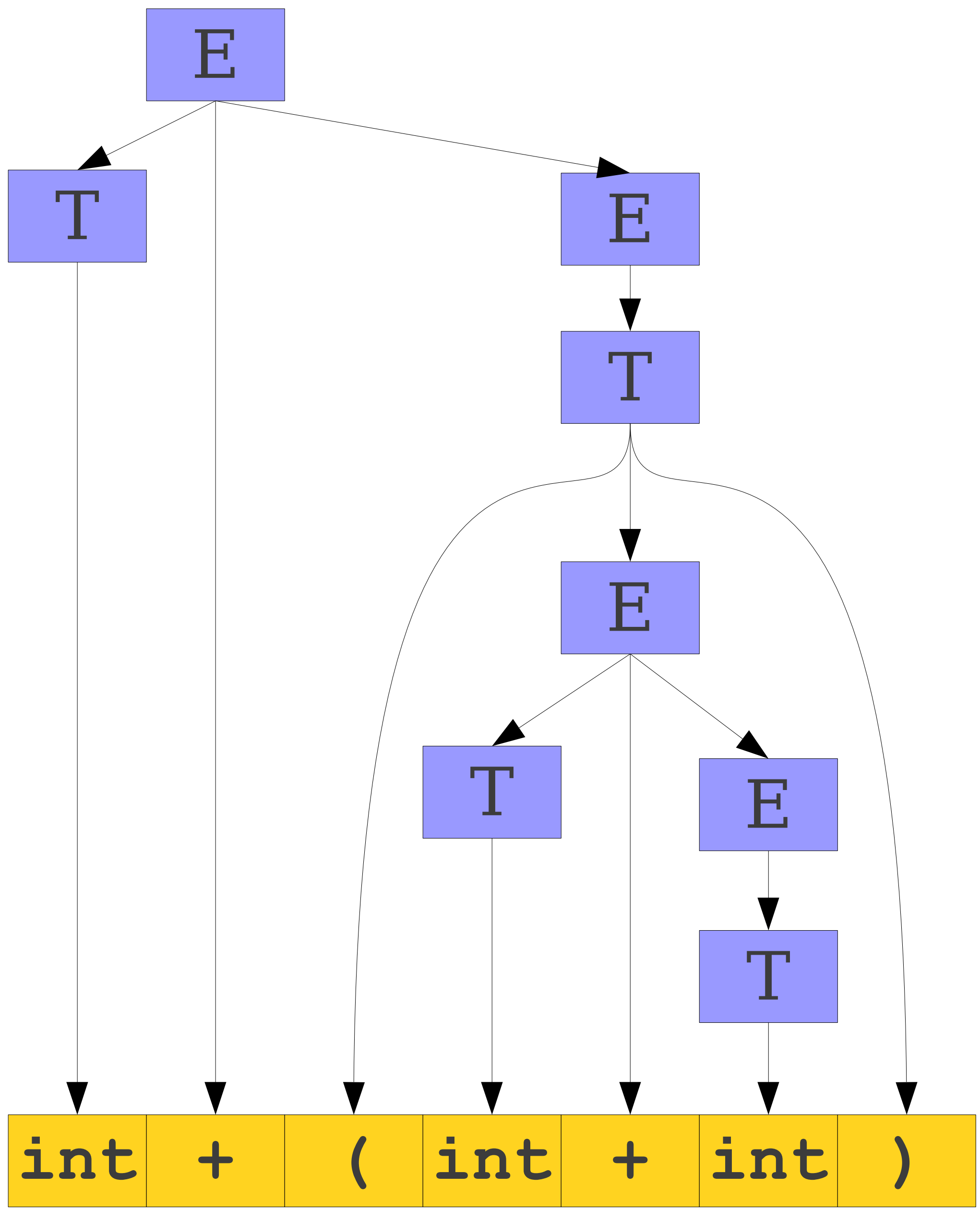
- Written Assignment 1 due this afternoon at 5PM.
  - Can submit electronically by emailing us at **cs143-sum1112-staff@lists.stanford.edu** with [WA1] somewhere in the subject line.
  - Can submit hard copies to the drop-off box in Gates (details in the problem set).
- C++ review session next Monday, time and place TBA.

Where We Are


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

Top-Down Parsing

E → T
E → T + E
T → int
T → (E)



LL(1) Parse Tables

E → `int`
E → `(`**E Op E**`)`
Op → `+`
Op → `*`

	`int`	`(`	`)`	`+`	`*`
E	`int`	(E Op E)			
Op				`+`	`*`

FIRST Sets

- We want to tell if a particular nonterminal **A** derives a string starting with a particular nonterminal **`t`**.
- We can formalize this with **FIRST sets**.

$$\text{FIRST}(\mathbf{A}) = \{\ \mathtt{t} \mid \mathbf{A} \Rightarrow^* \mathtt{t}\boldsymbol{\omega} \text{ for some } \boldsymbol{\omega}\ \}$$

- We also include **ε** in FIRST(**A**) if A can produce the empty string.
- Intuitively, FIRST(**A**) is the set of terminals that can be at the start of a string produced by **A**.
- We can generalize FIRST to strings with FIRST*(***ω***) being the set of all terminals (or **ε**) that can appear at the start of a string derived from ***ω***.

FIRST Computation with ε

- Initially, for all nonterminals **A**, set

  FIRST(**A**) = { **t** | **A** → **t*ω*** for some ***ω*** }
- For all nonterminals **A** where **A** → **ε** is a production, add **ε** to FIRST(**A**).
- Repeat the following until no changes occur:
  - For each production **A** → ***α***, set FIRST(**A**) = FIRST(**A**) ∪ FIRST*(***α***)

LL(1) Tables with ε

Num	→	**Sign Digits**
Sign	→	+ | - | ε
Digits	→	**Digit More**
More	→	**Digits** | ε
Digit	→	0 | 1 | ... | 9

LL(1) Tables with ε

Num → **Sign Digits**
Sign → `+` | `-` | ε
Digits → **Digit More**
More → **Digits** | ε
Digit → `0` | `1` | … | `9`

	+	-	#	$
Num				
Sign				
Digits				
More				
Digit				

LL(1) Tables with ε

Num → **Sign Digits**
Sign → **+** | **-** | **ε**
Digits → **Digit More**
More → **Digits** | **ε**
Digit → **0** | **1** | … | **9**

Num	Sign	Digit	Digits	More
+ − 0 5 1 6 2 7 3 8 4 9	+ − ε	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9 ε

	+	-	#	$
Num				
Sign				
Digits				
More				
Digit				

LL(1) Tables with ε

Num → **Sign Digits**
Sign → + | - | ε
Digits → **Digit More**
More → **Digits** | ε
Digit → 0 | 1 | … | 9

Num	Sign	Digit	Digits	More
+ - 0 5 1 6 2 7 3 8 4 9	+ - ε	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9 ε

	+	-	#	$
Num				
Sign				
Digits				
More				
Digit				

LL(1) Tables with ε

Num → **Sign Digits**
Sign → **+** | **–** | **ε**
Digits → **Digit More**
More → **Digits** | **ε**
Digit → **0** | **1** | … | **9**

Num	Sign	Digit	Digits	More
+ – 0 5 1 6 2 7 3 8 4 9	+ – ε	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9 ε

	+	–	#	$
Num	**Sign Digits**	**Sign Digits**		
Sign				
Digits				
More				
Digit				

LL(1) Tables with ε

Num → **Sign Digits**
Sign → + | - | ε
Digits → **Digit More**
More → **Digits** | ε
Digit → 0 | 1 | … | 9

Num	Sign	Digit	Digits	More
+ - 0 1 2 3 4 5 6 7 8 9	+ - ε	0 1 2 3 4 5 6 7 8 9	0 1 2 3 4 5 6 7 8 9	0 1 2 3 4 5 6 7 8 9 ε

	+	-	#	$
Num	**Sign Digits**	**Sign Digits**		
Sign				
Digits				
More				
Digit				

LL(1) Tables with ε

Num → **Sign Digits**
Sign → + | - | ε
Digits → **Digit More**
More → **Digits** | ε
Digit → 0 | 1 | … | 9

Num	Sign	Digit	Digits	More
+ − 0 5 1 6 2 7 3 8 4 9	+ − ε	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9 ε

	+	-	#	$
Num	**Sign Digits**	**Sign Digits**		
Sign	+	-		
Digits				
More				
Digit				

LL(1) Tables with ε

Num → **Sign Digits**
Sign → + | - | ε
Digits → **Digit More**
More → **Digits** | ε
Digit → 0 | 1 | … | 9

Num	Sign	Digit	Digits	More
+ − 0 5 1 6 2 7 3 8 4 9	+ − ε	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9 ε

	+	-	#	$
Num	Sign Digits	Sign Digits		
Sign	+	-		
Digits				
More				
Digit				

LL(1) Tables with ε

Num → **Sign Digits**
Sign → + | - | ε
Digits → **Digit More**
More → **Digits** | ε
Digit → 0 | 1 | … | 9

Num	Sign	Digit	Digits	More
+ – 0 5 1 6 2 7 3 8 4 9	+ – ε	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9 ε

	+	-	#	$
Num	**Sign Digits**	**Sign Digits**		
Sign	+	-		
Digits			**Digits More**	
More				
Digit				

LL(1) Tables with ε

Num → **Sign Digits**
Sign → + | - | ε
Digits → **Digit More**
More → **Digits** | ε
Digit → 0 | 1 | … | 9

Num	Sign	Digit	Digits	More
+ - 0 5 1 6 2 7 3 8 4 9	+ - ε	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9 ε

	+	-	#	$
Num	**Sign Digits**	**Sign Digits**		
Sign	+	-		
Digits			**Digits More**	
More				
Digit				

LL(1) Tables with ε

Num → **Sign Digits**
Sign → + | - | ε
Digits → **Digit More**
More → **Digits** | ε
Digit → 0 | 1 | … | 9

Num	Sign	Digit	Digits	More
+ - 0 5 1 6 2 7 3 8 4 9	+ - ε	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9 ε

	+	-	#	$
Num	**Sign Digits**	**Sign Digits**		
Sign	+	-		
Digits			**Digits More**	
More			**Digits**	
Digit				

LL(1) Tables with ε

Num → **Sign Digits**
Sign → + | - | ε
Digits → **Digit More**
More → **Digits** | ε
Digit → 0 | 1 | … | 9

Num	Sign	Digit	Digits	More
+ – 0 5 1 6 2 7 3 8 4 9	+ – ε	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9 ε

	+	-	#	$
Num	**Sign Digits**	**Sign Digits**		
Sign	+	-		
Digits			**Digits More**	
More			**Digits**	
Digit				

LL(1) Tables with ε

Num → **Sign Digits**
Sign → + | - | ε
Digits → **Digit More**
More → **Digits** | ε
Digit → 0 | 1 | … | 9

Num	Sign	Digit	Digits	More
+ – 0 5 1 6 2 7 3 8 4 9	+ – ε	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9 ε

	+	–	#	$
Num	**Sign Digits**	**Sign Digits**		
Sign	+	-		
Digits			**Digits More**	
More			**Digits**	
Digit			#	

LL(1) Tables with ε

Num → **Sign Digits**
Sign → + | - | ε
Digits → **Digit More**
More → **Digits** | ε
Digit → 0 | 1 | … | 9

Num	Sign	Digit	Digits	More
+ – 0 5 1 6 2 7 3 8 4 9	+ – ε	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9 ε

	+	–	#	$
Num	**Sign Digits**	**Sign Digits**		
Sign	+	-		
Digits			**Digits More**	
More			**Digits**	
Digit			#	

LL(1) Tables with ε

Num → **Sign Digits**
Sign → + | - | ε
Digits → **Digit More**
More → **Digits** | ε
Digit → 0 | 1 | … | 9

Num	Sign	Digit	Digits	More
+ - 0 5 1 6 2 7 3 8 4 9	+ - ε	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9 ε

	+	-	#	$
Num	**Sign Digits**	**Sign Digits**		
Sign	+	-		
Digits			**Digits More**	
More			**Digits**	
Digit			#	

LL(1) Tables with ε

Num → **Sign Digits**
Sign → + | - | ε
Digits → **Digit More**
More → **Digits** | ε
Digit → 0 | 1 | … | 9

Num	Sign	Digit	Digits	More
+ - 0 5 1 6 2 7 3 8 4 9	+ - ε	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9 ε

	+	-	#	$
Num	**Sign Digits**	**Sign Digits**	**Sign Digits**	
Sign	+	-		
Digits			**Digits More**	
More			**Digits**	
Digit			#	

LL(1) Tables with ε

Num → **Sign Digits**
Sign → + | - | ε
Digits → **Digit More**
More → **Digits** | ε
Digit → 0 | 1 | … | 9

Num	Sign	Digit	Digits	More
+ - 0 5 1 6 2 7 3 8 4 9	+ - ε	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9 ε

	+	-	#	$
Num	**Sign Digits**	**Sign Digits**	**Sign Digits**	
Sign	+	-		
Digits			**Digits More**	
More			**Digits**	
Digit			#	

LL(1) Tables with ε

Num → **Sign Digits**
Sign → + | - | ε
Digits → **Digit More**
More → **Digits** | ε
Digit → 0 | 1 | … | 9

Num	Sign	Digit	Digits	More
+ – 0 5 1 6 2 7 3 8 4 9	+ – ε	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9 ε

	+	-	#	$
Num	**Sign Digits**	**Sign Digits**	**Sign Digits**	
Sign	+	-	ε	
Digits			**Digits More**	
More			**Digits**	
Digit			#	

LL(1) Tables with ε

Num → **Sign Digits**
Sign → + | - | ε
Digits → **Digit More**
More → **Digits** | ε
Digit → 0 | 1 | … | 9

Num	Sign	Digit	Digits	More
+ – 0 5 1 6 2 7 3 8 4 9	+ – ε	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9 ε

	+	–	#	$
Num	**Sign Digits**	**Sign Digits**	**Sign Digits**	
Sign	+	-	ε	
Digits			**Digits More**	
More			**Digits**	
Digit			#	

LL(1) Tables with ε

Num → **Sign Digits**
Sign → + | - | ε
Digits → **Digit More**
More → **Digits** | ε
Digit → 0 | 1 | … | 9

Num	Sign	Digit	Digits	More
+ - 0 5 1 6 2 7 3 8 4 9	+ - ε	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9 ε

	+	-	#	$
Num	**Sign Digits**	**Sign Digits**	**Sign Digits**	
Sign	+	-	ε	
Digits			**Digits More**	
More			**Digits**	
Digit			#	

LL(1) Tables with ε

Num → **Sign Digits**
Sign → + | - | ε
Digits → **Digit More**
More → **Digits** | ε
Digit → 0 | 1 | ... | 9

Num	Sign	Digit	Digits	More
+ - 0 5 1 6 2 7 3 8 4 9	+ - ε	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9 ε

	+	-	#	$
Num	**Sign Digits**	**Sign Digits**	**Sign Digits**	
Sign	+	-	ε	
Digits			**Digits More**	
More			**Digits**	ε
Digit			#	

LL(1) Tables with ε

Num → **Sign Digits**
Sign → + | - | ε
Digits → **Digit More**
More → **Digits** | ε
Digit → 0 | 1 | … | 9

Num	Sign	Digit	Digits	More
+ – 0 5 1 6 2 7 3 8 4 9	+ – ε	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9 ε

	+	-	#	$
Num	**Sign Digits**	**Sign Digits**	**Sign Digits**	
Sign	+	-	ε	
Digits			**Digits More**	
More			**Digits**	ε
Digit			#	

LL(1) Tables with ε

Num → **Sign Digits**
Sign → + | - | ε
Digits → **Digit More**
More → **Digits** | ε
Digit → 0 | 1 | … | 9

Num	Sign	Digit	Digits	More
+ - 0 5 1 6 2 7 3 8 4 9	+ - ε	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9 ε

	+	-	#	$
Num	**Sign Digits**	**Sign Digits**	**Sign Digits**	
Sign	+	-	ε	
Digits			**Digits More**	
More			**Digits**	ε
Digit			#	

FOLLOW Sets

- With ε-productions in the grammar, we may have to "look past" the current nonterminal to what can come after it.
- The **FOLLOW set** represents the set of terminals that might come after a given nonterminal.
- Formally:

$$\text{FOLLOW}(\mathbf{A}) = \{\ \mathtt{t} \mid \mathbf{S} \Rightarrow^* \boldsymbol{\alpha}\mathbf{A}\mathtt{t}\boldsymbol{\omega} \text{ for some } \boldsymbol{\alpha}, \boldsymbol{\omega}\ \}$$

  where **S** is the start symbol of the grammar.
- Informally, every nonterminal that can ever come after **A** in a derivation.

Computation of FOLLOW Sets

- Initially, for each nonterminal **A**, set

  FOLLOW(**A**) = { **t** | **B** → ***αAtω*** is a production }
- Add **$** to FOLLOW(**S**), where **S** is the start symbol.
- Repeat the following until no changes occur:
  - If **B** → ***αAω*** is a production, set FOLLOW(**A**) = FOLLOW(**A**) ∪ FIRST*(***ω***) - { **ε** }.
  - If **B** → ***αAω*** is a production and **ε** ∈ FIRST*(***ω***), set FOLLOW(**A**) = FOLLOW(**A**) ∪ FOLLOW(**B**).

The Final LL(1) Table Algorithm

- Compute FIRST(**A**) and FOLLOW(**A**) for all nonterminals **A**.
- For each rule **A** → ***ω***, for each terminal **`t`** ∈ FIRST*(***ω***), set T[**A**, **`t`**] = ***ω***.
  - Note that **ε** is not a terminal.
- For each rule **A** → ***ω***, if **ε** ∈ FIRST*(***ω***), set T[**A**, **`t`**] = ***ω*** for each **`t`** ∈ FOLLOW(**A**).

An Egregious Abuse of Notation

- Compute FIRST(**A**) and FOLLOW(**A**) for all nonterminals **A**.
- For each rule **A** → $\boldsymbol{\omega}$, for each terminal `t` ∈ FIRST*($\boldsymbol{\omega}$FOLLOW(**A**)), set T[**A**, `t`] = $\boldsymbol{\omega}$.

Example LL(1) Construction

The Limits of LL(1)

A Grammar that is Not LL(1)

- Consider the following (left-recursive) grammar:

  A → **A**`b` | `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 → **A**`b` | `c`

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

	b	c
A		**A** → **A**`b` **A** → `c`

A Grammar that is Not LL(1)

- Consider the following (left-recursive) grammar:

  A → Ab | c

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

	b	c
A		A → Ab A → 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

$$\mathbf{A} \rightarrow \mathbf{A}\boldsymbol{\omega} \mid \boldsymbol{\alpha}$$

- This will produce $\boldsymbol{\alpha}$ followed by some number of $\boldsymbol{\omega}$'s.
- Can rewrite the grammar as

$$\mathbf{A} \rightarrow \boldsymbol{\alpha}\mathbf{B}$$

$$\mathbf{B} \rightarrow \boldsymbol{\varepsilon} \mid \boldsymbol{\omega}\mathbf{B}$$

Another Non-LL(1) Grammar

- Consider the following grammar:

  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:

E → **T**

E → **T** `+` **E**

T → `int`

T → `(`**E**`)`

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

How do you predict which of these to use?

Left-Factoring

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

Left-Factoring

$$\mathbf{E} \rightarrow \mathbf{T}\varepsilon$$

$$\mathbf{E} \rightarrow \mathbf{T} + \mathbf{E}$$

$$\mathbf{T} \rightarrow \texttt{int}$$

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

Left-Factoring

$$E \rightarrow TY$$

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

Left-Factoring

$E \rightarrow TY$

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

Left-Factoring

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

Left-Factoring

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

Left-Factoring

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

FIRST		
E	T	Y

FOLLOW		
E	T	Y

Left-Factoring

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

FIRST		
E	T	Y
	`int` `(`	

FOLLOW		
E	T	Y

Left-Factoring

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

FIRST		
E	T	Y
	`int` `(`	`+` ε

FOLLOW		
E	T	Y

Left-Factoring

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

FIRST		
E	T	Y
int (	int (	+ ε

FOLLOW		
E	T	Y

Left-Factoring

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

FIRST		
E	T	Y
int (	int (	+ ε

FOLLOW		
E	T	Y
$		

Left-Factoring

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

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

FOLLOW		
E	T	Y
`$` `)`		

Left-Factoring

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

FIRST		
E	T	Y
int (	int (	+ ε

FOLLOW		
E	T	Y
$)	+	

Left-Factoring

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

FIRST		
E	T	Y
`int` `(`	`int` `(`	`+` ε
FOLLOW		
E	T	Y
`$` `)`	`+`	`$` `)`

Left-Factoring

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

FIRST		
E	T	Y
int (	int (	+ ε
FOLLOW		
E	T	Y
$)	+ $)	$)

Left-Factoring

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

FIRST		
E	T	Y
int (	int (	+ ε

FOLLOW		
E	T	Y
$)	+ $)	$)

	int	(	)	+	$
E					
T					
Y					

Left-Factoring

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

FIRST		
E	T	Y
int (	int (	+ ε

FOLLOW		
E	T	Y
$)	+ $)	$)

	int	(	)	+	$
E	1	1			
T					
Y					

Left-Factoring

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

FIRST		
E	T	Y
int (	int (	+ ε

FOLLOW		
E	T	Y
$)	+ $)	$)

	int	(	)	+	$
E	1	1			
T	2	3			
Y					

Left-Factoring

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

FIRST		
E	T	Y
int (	int (	+ ε

FOLLOW		
E	T	Y
$)	+ $)	$)

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

Left-Factoring

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

FIRST		
E	T	Y
int (	int (	+ ε

FOLLOW		
E	T	Y
$)	+ $)	$)

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

A Formal Characterization of LL(1)

- A grammar *G* is LL(1) iff for any productions **A** → ω_1 and **A** → ω_2, the sets

  $$\text{FIRST}(\omega_1 \ \text{FOLLOW}(\mathbf{A}))$$

  and

  $$\text{FIRST}(\omega_2 \ \text{FOLLOW}(\mathbf{A}))$$

  are disjoint.
- This condition is equivalent to saying that there are no conflicts in the table.

The Strengths of LL(1)

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 $O(n\ |G|)$ time, where n is the length of the string and $|G|$ is the size of the grammar.

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 factorability** cause LL(1) to fail and can be mechanically eliminated in some cases.