Compiler Design

Lecture 8: Syntax Analysis IV (Parsing)

Sahar Selim

Agenda

- Continue with LL(1) Parsing
  - First Sets
  - Follow Sets
  - Left factoring

Can we find an algorithm for constructing LL(1) parse tables?

LL(1) Parsing

(1) E → int
(2) E → (E Op E)
(3) Op → +
(4) Op → *

	int	(	)	+	*
E	1	2			
Op				3	4

E$	(int + (int * int))$
(E Op E)$	(int + (int * int))$
E Op E)$	int + (int * int))$
int Op E)$	int + (int * int))$
Op E)$	+ (int * int))$
+ E)$	+ (int * int))$
E)$	(int * int))$
(E Op E))$	(int * int))$
E Op E))$	int * int))$
int Op E))$	int * int))$
Op E))$	* int))$
* E))$	* int))$
E))$	int))$
int))$	int))$
))$	))$
)$	)$

In what follows, assume that our grammar does not contain any ε-productions.

(We'll relax this restriction later)

FIRST Sets

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**.
- FIRST(**A**) = { **t** | **A** $\Rightarrow^*$ **t**ω for some ω }
- Intuitively, FIRST(**A**) is the set of terminals that can be at the start of a string produced by **A**.
- If we can compute FIRST sets for all nonterminals in a grammar, we can efficiently construct the LL(1) parsing table. Details soon.

Computing FIRST Sets

- Initially, for all nonterminals A, set

$$\text{FIRST}(A) = \{\ t \mid A \rightarrow t\omega \text{ for some } \omega\ \}$$

- Then, repeat the following until no changes occur:
  - For each nonterminal A, for each production $A \rightarrow B\omega$, set

$$\text{FIRST}(A) = \text{FIRST}(A) \cup \text{FIRST}(B)$$

- This is known as a **fixed-point iteration** or a **transitive closure algorithm**.

Iterative FIRST Computations

```
STMT → if EXPR then STMT
     | while EXPR do STMT
     | EXPR ;

EXPR → TERM -> id
     | zero? TERM
     | not EXPR
     | ++ id
     | -- id

TERM → id
     | constant
```

Iterative FIRST Computations

```
STMT → if EXPR then STMT
     | while EXPR do STMT
     | EXPR ;

EXPR → TERM -> id
     | zero? TERM
     | not EXPR
     | ++ id
     | -- id

TERM → id
     | constant
```

STMT	EXPR	TERM

Iterative FIRST Computations

```
STMT → if EXPR then STMT
     | while EXPR do STMT
     | EXPR ;

EXPR → TERM -> id
     | zero? TERM
     | not EXPR
     | ++ id
     | -- id

TERM → id
     | constant
```

STMT	EXPR	TERM
if while		

Iterative FIRST Computations

```
STMT → if EXPR then STMT
     | while EXPR do STMT
     | EXPR ;

EXPR → TERM -> id
     | zero? TERM
     | not EXPR
     | ++ id
     | -- id

TERM → id
     | constant
```

STMT	EXPR	TERM
if while	zero? not ++ --	

Iterative FIRST Computations

```
STMT → if EXPR then STMT
     | while EXPR do STMT
     | EXPR ;

EXPR → TERM -> id
     | zero? TERM
     | not EXPR
     | ++ id
     | -- id

TERM → id
     | constant
```

STMT	EXPR	TERM
if while	zero? not ++ --	id constant

Iterative FIRST Computations

```
STMT → if EXPR then STMT
     | while EXPR do STMT
     | EXPR ;

EXPR → TERM -> id
     | zero? TERM
     | not EXPR
     | ++ id
     | -- id

TERM → id
     | constant
```

STMT	EXPR	TERM
if while	zero? not ++ --	id constant

Iterative FIRST Computations

```
STMT → if EXPR then STMT
     | while EXPR do STMT
     | EXPR ;

EXPR → TERM -> id
     | zero? TERM
     | not EXPR
     | ++ id
     | -- id

TERM → id
     | constant
```

STMT	EXPR	TERM
if while zero? not ++ --	zero? not ++ --	id constant

Iterative FIRST Computations

```
STMT → if EXPR then STMT
     | while EXPR do STMT
     | EXPR ;

EXPR → TERM -> id
     | zero? TERM
     | not EXPR
     | ++ id
     | -- id

TERM → id
     | constant
```

first(EXPR)

STMT	EXPR	TERM
if while zero? not ++ --	zero? not ++ --	id constant

Iterative FIRST Computations

```
STMT → if EXPR then STMT
     | while EXPR do STMT
     | EXPR ;

EXPR → TERM -> id
     | zero? TERM
     | not EXPR
     | ++ id
     | -- id

TERM → id
     | constant
```

STMT	EXPR	TERM
if	zero?	id
while	not	constant
zero?	++	
not	--	
++		
--		

Iterative FIRST Computations

```
STMT → if EXPR then STMT
     | while EXPR do STMT
     | EXPR ;

EXPR → TERM -> id
     | zero? TERM
     | not EXPR
     | ++ id
     | -- id

TERM → id
     | constant
```

STMT	EXPR	TERM
if while zero? not ++ --	zero? not ++ --	id constant

Iterative FIRST Computations

```
STMT  →  if EXPR then STMT
      |  while EXPR do STMT
      |  EXPR ;

EXPR  →  TERM -> id
      |  zero? TERM
      |  not EXPR
      |  ++ id
      |  -- id

TERM  →  id
      |  constant
```

STMT	EXPR	TERM
if	zero?	id
while	not	constant
zero?	++	
not	--	
++	id	
--	constant	

first(TERM)

Iterative FIRST Computations

```
STMT  →  if EXPR then STMT
      |  while EXPR do STMT
      |  EXPR ;

EXPR  →  TERM -> id
      |  zero? TERM
      |  not EXPR
      |  ++ id
      |  -- id

TERM  →  id
      |  constant
```

STMT	EXPR	TERM
if	zero?	id
while	not	constant
zero?	++	
not	--	
++	id	
--	constant	

Iterative FIRST Computations

```
STMT → if EXPR then STMT
     | while EXPR do STMT
     | EXPR ;

EXPR → TERM -> id
     | zero? TERM
     | not EXPR
     | ++ id
     | -- id

TERM → id
     | constant
```

STMT	EXPR	TERM
if	zero?	id
while	not	constant
zero?	++	
not	--	
++	id	
--	constant	

Iterative FIRST Computations

```
STMT → if EXPR then STMT
     | while EXPR do STMT
     | EXPR ;

EXPR → TERM -> id
     | zero? TERM
     | not EXPR
     | ++ id
     | -- id

TERM → id
     | constant
```

STMT	EXPR	TERM
if	zero?	id
while	not	constant
zero?	++	
not	--	
++	id	
--	constant	
id		
constant		

Iterative FIRST Computations

```
STMT → if EXPR then STMT
     | while EXPR do STMT
     | EXPR ;

EXPR → TERM -> id
     | zero? TERM
     | not EXPR
     | ++ id
     | -- id

TERM → id
     | constant
```

STMT	EXPR	TERM
if	zero?	id
while	not	constant
zero?	++	
not	--	
++	id	
--	constant	
id		
constant		

From FIRST Sets to LL(1) Tables

STMT	→	if EXPR then STMT	(1)
	|	while EXPR do STMT	(2)
	|	EXPR ;	(3)
EXPR	→	TERM -> id	(4)
	|	zero? TERM	(5)
	|	not EXPR	(6)
	|	++ id	(7)
	|	-- id	(8)
TERM	→	id	(9)
	|	constant	(10)

From FIRST Sets to LL(1) Tables

STMT	→	if EXPR then STMT	(1)
	|	while EXPR do STMT	(2)
	|	EXPR ;	(3)
EXPR	→	TERM -> id	(4)
	|	zero? TERM	(5)
	|	not EXPR	(6)
	|	++ id	(7)
	|	-- id	(8)
TERM	→	id	(9)
	|	constant	(10)

	if	then	while	do	zero?	not	++	--	→	id	const	;
STMT												
EXPR												
TERM												

From FIRST Sets to LL(1) Tables

STMT	→	if EXPR then STMT	(1)
	|	while EXPR do STMT	(2)
	|	EXPR ;	(3)
EXPR	→	TERM -> id	(4)
	|	zero? TERM	(5)
	|	not EXPR	(6)
	|	++ id	(7)
	|	-- id	(8)
TERM	→	id	(9)
	|	constant	(10)

STMT	EXPR	TERM
if while zero? not ++ -- id constant	zero? not ++ -- id constant	id constant

	if	then	while	do	zero?	not	++	--	→	id	const	;
STMT												
EXPR												
TERM												

From FIRST Sets to LL(1) Tables

STMT	→	if EXPR then STMT	(1)
	|	while EXPR do STMT	(2)
	|	EXPR ;	(3)
EXPR	→	TERM -> id	(4)
	|	zero? TERM	(5)
	|	not EXPR	(6)
	|	++ id	(7)
	|	-- id	(8)
TERM	→	id	(9)
	|	constant	(10)

STMT	EXPR	TERM
if while zero? not ++ -- id constant	zero? not ++ -- id constant	id constant

	if	then	while	do	zero?	not	++	--	→	id	const	;
STMT	1											
EXPR												
TERM												

From FIRST Sets to LL(1) Tables

STMT	→	if EXPR then STMT	(1)
	|	while EXPR do STMT	(2)
	|	EXPR ;	(3)
EXPR	→	TERM -> id	(4)
	|	zero? TERM	(5)
	|	not EXPR	(6)
	|	++ id	(7)
	|	-- id	(8)
TERM	→	id	(9)
	|	constant	(10)

STMT	EXPR	TERM
if while zero? not ++ -- id constant	zero? not ++ -- id constant	id constant

	if	then	while	do	zero?	not	++	--	→	id	const	;
STMT	1											
EXPR												
TERM												

From FIRST Sets to LL(1) Tables

STMT	→	if EXPR then STMT	(1)
	|	while EXPR do STMT	(2)
	|	EXPR ;	(3)
EXPR	→	TERM -> id	(4)
	|	zero? TERM	(5)
	|	not EXPR	(6)
	|	++ id	(7)
	|	-- id	(8)
TERM	→	id	(9)
	|	constant	(10)

STMT	EXPR	TERM
if while zero? not ++ -- id constant	zero? not ++ -- id constant	id constant

	if	then	while	do	zero?	not	++	--	→	id	const	;
STMT	1		2									
EXPR												
TERM												

From FIRST Sets to LL(1) Tables

STMT	→	if EXPR then STMT	(1)
	|	while EXPR do STMT	(2)
	|	EXPR ;	(3)
EXPR	→	TERM -> id	(4)
	|	zero? TERM	(5)
	|	not EXPR	(6)
	|	++ id	(7)
	|	-- id	(8)
TERM	→	id	(9)
	|	constant	(10)

STMT	EXPR	TERM
if while zero? not ++ -- id constant	zero? not ++ -- id constant	id constant

	if	then	while	do	zero?	not	++	--	→	id	const	;
STMT	1		2									
EXPR												
TERM												

From FIRST Sets to LL(1) Tables

STMT	→	if EXPR then STMT	(1)
	|	while EXPR do STMT	(2)
	|	EXPR ;	(3)
EXPR	→	TERM -> id	(4)
	|	zero? TERM	(5)
	|	not EXPR	(6)
	|	++ id	(7)
	|	-- id	(8)
TERM	→	id	(9)
	|	constant	(10)

STMT	EXPR	TERM
if while zero? not ++ -- id constant	zero? not ++ -- id constant	id constant

	if	then	while	do	zero?	not	++	--	→	id	const	;
STMT	1		2									
EXPR												
TERM												

From FIRST Sets to LL(1) Tables

STMT	→	if EXPR then STMT	(1)
	|	while EXPR do STMT	(2)
	|	EXPR ;	(3)
EXPR	→	TERM -> id	(4)
	|	zero? TERM	(5)
	|	not EXPR	(6)
	|	++ id	(7)
	|	-- id	(8)
TERM	→	id	(9)
	|	constant	(10)

STMT	EXPR	TERM
if while zero? not ++ -- id constant	zero? not ++ -- id constant	id constant

	if	then	while	do	zero?	not	++	--	→	id	const	;
STMT	1		2		3	3	3	3		3	3	
EXPR												
TERM												

From FIRST Sets to LL(1) Tables

STMT	→	if EXPR then STMT	(1)
	|	while EXPR do STMT	(2)
	|	EXPR ;	(3)
EXPR	→	TERM -> id	(4)
	|	zero? TERM	(5)
	|	not EXPR	(6)
	|	++ id	(7)
	|	-- id	(8)
TERM	→	id	(9)
	|	constant	(10)

STMT	EXPR	TERM
if while zero? not ++ -- id constant	zero? not ++ -- id constant	id constant

	if	then	while	do	zero?	not	++	--	→	id	const	;
STMT	1		2		3	3	3	3		3	3	
EXPR												
TERM												

From FIRST Sets to LL(1) Tables

STMT	→	if EXPR then STMT	(1)
	|	while EXPR do STMT	(2)
	|	EXPR ;	(3)
EXPR	→	TERM -> id	(4)
	|	zero? TERM	(5)
	|	not EXPR	(6)
	|	++ id	(7)
	|	-- id	(8)
TERM	→	id	(9)
	|	constant	(10)

STMT	EXPR	TERM
if while zero? not ++ -- id constant	zero? not ++ -- id constant	id constant

	if	then	while	do	zero?	not	++	--	→	id	const	;
STMT	1		2		3	3	3	3		3	3	
EXPR												
TERM												

From FIRST Sets to LL(1) Tables

STMT	→	if EXPR then STMT	(1)
	|	while EXPR do STMT	(2)
	|	EXPR ;	(3)
EXPR	→	TERM -> id	(4)
	|	zero? TERM	(5)
	|	not EXPR	(6)
	|	++ id	(7)
	|	-- id	(8)
TERM	→	id	(9)
	|	constant	(10)

STMT	EXPR	TERM
if while zero? not ++ -- id constant	zero? not ++ -- id constant	id constant

	if	then	while	do	zero?	not	++	--	→	id	const	;
STMT	1		2		3	3	3	3		3	3	
EXPR										4	4	
TERM												

From FIRST Sets to LL(1) Tables

STMT	→	if EXPR then STMT	(1)
	|	while EXPR do STMT	(2)
	|	EXPR ;	(3)
EXPR	→	TERM -> id	(4)
	|	zero? TERM	(5)
	|	not EXPR	(6)
	|	++ id	(7)
	|	-- id	(8)
TERM	→	id	(9)
	|	constant	(10)

STMT	EXPR	TERM
if while zero? not ++ -- id constant	zero? not ++ -- id constant	id constant

	if	then	while	do	zero?	not	++	--	→	id	const	;
STMT	1		2		3	3	3	3		3	3	
EXPR										4	4	
TERM												

From FIRST Sets to LL(1) Tables

STMT	→	if EXPR then STMT	(1)
	|	while EXPR do STMT	(2)
	|	EXPR ;	(3)
EXPR	→	TERM -> id	(4)
	|	zero? TERM	(5)
	|	not EXPR	(6)
	|	++ id	(7)
	|	-- id	(8)
TERM	→	id	(9)
	|	constant	(10)

STMT	EXPR	TERM
if while zero? not ++ -- id constant	zero? not ++ -- id constant	id constant

	if	then	while	do	zero?	not	++	--	→	id	const	;
STMT	1		2		3	3	3	3		3	3	
EXPR					5					4	4	
TERM												

From FIRST Sets to LL(1) Tables

STMT	→	if EXPR then STMT	(1)
	|	while EXPR do STMT	(2)
	|	EXPR ;	(3)
EXPR	→	TERM -> id	(4)
	|	zero? TERM	(5)
	|	not EXPR	(6)
	|	++ id	(7)
	|	-- id	(8)
TERM	→	id	(9)
	|	constant	(10)

STMT	EXPR	TERM
if while zero? not ++ -- id constant	zero? not ++ -- id constant	id constant

	if	then	while	do	zero?	not	++	--	→	id	const	;
STMT	1		2		3	3	3	3		3	3	
EXPR					5					4	4	
TERM												

From FIRST Sets to LL(1) Tables

STMT	→	if EXPR then STMT	(1)
	|	while EXPR do STMT	(2)
	|	EXPR ;	(3)
EXPR	→	TERM -> id	(4)
	|	zero? TERM	(5)
	|	not EXPR	(6)
	|	++ id	(7)
	|	-- id	(8)
TERM	→	id	(9)
	|	constant	(10)

STMT	EXPR	TERM
if while zero? not ++ -- id constant	zero? not ++ -- id constant	id constant

	if	then	while	do	zero?	not	++	--	→	id	const	;
STMT	1		2		3	3	3	3		3	3	
EXPR					5	6				4	4	
TERM												

From FIRST Sets to LL(1) Tables

STMT	→	if EXPR then STMT	(1)
	|	while EXPR do STMT	(2)
	|	EXPR ;	(3)
EXPR	→	TERM -> id	(4)
	|	zero? TERM	(5)
	|	not EXPR	(6)
	|	++ id	(7)
	|	-- id	(8)
TERM	→	id	(9)
	|	constant	(10)

STMT	EXPR	TERM
if while zero? not ++ -- id constant	zero? not ++ -- id constant	id constant

	if	then	while	do	zero?	not	++	--	→	id	const	;
STMT	1		2		3	3	3	3		3	3	
EXPR					5	6				4	4	
TERM												

From FIRST Sets to LL(1) Tables

STMT	→	if EXPR then STMT	(1)
	|	while EXPR do STMT	(2)
	|	EXPR ;	(3)
EXPR	→	TERM -> id	(4)
	|	zero? TERM	(5)
	|	not EXPR	(6)
	|	++ id	(7)
	|	-- id	(8)
TERM	→	id	(9)
	|	constant	(10)

STMT	EXPR	TERM
if while zero? not ++ -- id constant	zero? not ++ -- id constant	id constant

	if	then	while	do	zero?	not	++	--	→	id	const	;
STMT	1		2		3	3	3	3		3	3	
EXPR					5	6	7			4	4	
TERM												

From FIRST Sets to LL(1) Tables

STMT	→	if EXPR then STMT	(1)
	|	while EXPR do STMT	(2)
	|	EXPR ;	(3)
EXPR	→	TERM -> id	(4)
	|	zero? TERM	(5)
	|	not EXPR	(6)
	|	++ id	(7)
	|	-- id	(8)
TERM	→	id	(9)
	|	constant	(10)

STMT	EXPR	TERM
if while zero? not ++ -- id constant	zero? not ++ -- id constant	id constant

	if	then	while	do	zero?	not	++	--	→	id	const	;
STMT	1		2		3	3	3	3		3	3	
EXPR					5	6	7			4	4	
TERM												

From FIRST Sets to LL(1) Tables

STMT	→	if EXPR then STMT	(1)
	|	while EXPR do STMT	(2)
	|	EXPR ;	(3)
EXPR	→	TERM -> id	(4)
	|	zero? TERM	(5)
	|	not EXPR	(6)
	|	++ id	(7)
	|	-- id	(8)
TERM	→	id	(9)
	|	constant	(10)

STMT	EXPR	TERM
if while zero? not ++ -- id constant	zero? not ++ -- id constant	id constant

	if	then	while	do	zero?	not	++	--	→	id	const	;
STMT	1		2		3	3	3	3		3	3	
EXPR					5	6	7	8		4	4	
TERM												

From FIRST Sets to LL(1) Tables

STMT	→	if EXPR then STMT	(1)
	|	while EXPR do STMT	(2)
	|	EXPR ;	(3)
EXPR	→	TERM -> id	(4)
	|	zero? TERM	(5)
	|	not EXPR	(6)
	|	++ id	(7)
	|	-- id	(8)
TERM	→	id	(9)
	|	constant	(10)

STMT	EXPR	TERM
if while zero? not ++ -- id constant	zero? not ++ -- id constant	id constant

	if	then	while	do	zero?	not	++	--	→	id	const	;
STMT	1		2		3	3	3	3		3	3	
EXPR					5	6	7	8		4	4	
TERM												

From FIRST Sets to LL(1) Tables

STMT	→	if EXPR then STMT	(1)
	|	while EXPR do STMT	(2)
	|	EXPR ;	(3)
EXPR	→	TERM -> id	(4)
	|	zero? TERM	(5)
	|	not EXPR	(6)
	|	++ id	(7)
	|	-- id	(8)
TERM	→	id	(9)
	|	constant	(10)

STMT	EXPR	TERM
if while zero? not ++ -- id constant	zero? not ++ -- id constant	id constant

	if	then	while	do	zero?	not	++	--	→	id	const	;
STMT	1		2		3	3	3	3		3	3	
EXPR					5	6	7	8		4	4	
TERM										9		

From FIRST Sets to LL(1) Tables

STMT	→	if EXPR then STMT	(1)
	|	while EXPR do STMT	(2)
	|	EXPR ;	(3)
EXPR	→	TERM -> id	(4)
	|	zero? TERM	(5)
	|	not EXPR	(6)
	|	++ id	(7)
	|	-- id	(8)
TERM	→	id	(9)
	|	constant	(10)

STMT	EXPR	TERM
if while zero? not ++ -- id constant	zero? not ++ -- id constant	id constant

	if	then	while	do	zero?	not	++	--	→	id	const	;
STMT	1		2		3	3	3	3		3	3	
EXPR					5	6	7	8		4	4	
TERM										9		

From FIRST Sets to LL(1) Tables

STMT	→	if EXPR then STMT	(1)
	|	while EXPR do STMT	(2)
	|	EXPR ;	(3)
EXPR	→	TERM -> id	(4)
	|	zero? TERM	(5)
	|	not EXPR	(6)
	|	++ id	(7)
	|	-- id	(8)
TERM	→	id	(9)
	|	constant	(10)

STMT	EXPR	TERM
if while zero? not ++ -- id constant	zero? not ++ -- id constant	id constant

	if	then	while	do	zero?	not	++	--	→	id	const	;
STMT	1		2		3	3	3	3		3	3	
EXPR					5	6	7	8		4	4	
TERM										9	10	

From FIRST Sets to LL(1) Tables

STMT	→	if EXPR then STMT	(1)
	|	while EXPR do STMT	(2)
	|	EXPR ;	(3)
EXPR	→	TERM -> id	(4)
	|	zero? TERM	(5)
	|	not EXPR	(6)
	|	++ id	(7)
	|	-- id	(8)
TERM	→	id	(9)
	|	constant	(10)

STMT	EXPR	TERM
if while zero? not ++ -- id constant	zero? not ++ -- id constant	id constant

	if	then	while	do	zero?	not	++	--	→	id	const	;
STMT	1		2		3	3	3	3		3	3	
EXPR					5	6	7	8		4	4	
TERM										9	10	

ε-Free LL(1) Parse Tables

- The following algorithm constructs an LL(1) parse table for a grammar with no ε-productions.
- Compute the FIRST sets for all nonterminals in the grammar.
- For each production $A \rightarrow t\omega$, set $T[A, t] = t\omega$
- For each production $A \rightarrow B\omega$, set $T[A, t] = B\omega$ for each $t \in FIRST(B)$.

LL(1) with ε-Productions

- Computation of FIRST is different.
  - What if the first nonterminal in a production can produce ε?
- Building the table is different.
  - What action do you take if the correct production produces the empty string?

FIRST Sets with ε

Num → **Sign Digits**
Sign → + | - | ε
Digits → **Digit More**
More → **Digits** | ε
Digit → 0 | 1 | 2 | ... | 9

Num	Sign	Digit	Digits	More

FIRST Sets with ε

Num → **Sign Digits**
Sign → + | - | ε
Digits → **Digit More**
More → **Digits** | ε
Digit → 0 | 1 | 2 | ... | 9

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

FIRST Sets with ε

Num → **Sign Digits**
Sign → + | - | ε
Digits → **Digit More**
More → **Digits** | ε
Digit → 0 | 1 | 2 | ... | 9

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

FIRST Sets with ε

Num → **Sign Digits**
Sign → + | - | ε
Digits → **Digit More**
More → **Digits** | ε
Digit → 0 | 1 | 2 | ... | 9

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

FIRST Sets with ε

Num → **Sign Digits**
Sign → + | - | ε
Digits → **Digit More**
More → **Digits** | ε
Digit → 0 | 1 | 2 | ... | 9

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

FIRST Sets with ε

Num → **Sign Digits**
Sign → + | - | ε
Digits → **Digit More**
More → **Digits** | ε
Digit → 0 | 1 | 2 | ... | 9

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

FIRST Sets with ε

Num → Sign Digits
Sign → + | - | ε
Digits → Digit More
More → Digits | ε
Digit → 0 | 1 | 2 | ... | 9

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

FIRST Sets with ε

Num → **Sign Digits**
Sign → + | - | ε
Digits → **Digit More**
More → **Digits** | ε
Digit → 0 | 1 | 2 | ... | 9

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

FIRST Sets with ε

Num → Sign Digits
Sign → + | - | ε
Digits → Digit More
More → Digits | ε
Digit → 0 | 1 | 2 | ... | 9

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

FIRST Sets with ε

Num → Sign Digits
Sign → + | - | ε
Digits → Digit More
More → Digits | ε
Digit → 0 | 1 | 2 | ... | 9

Num	Sign	Digit	Digits	More
+ -	+ -	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

FIRST Sets with ε

Num → **Sign Digits**
Sign → + | - | ε
Digits → **Digit More**
More → **Digits** | ε
Digit → 0 | 1 | 2 | ... | 9

Num	Sign	Digit	Digits	More
+ -	+ -	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

FIRST Sets with ε

Num → **Sign Digits**
Sign → + | - | ε
Digits → **Digit More**
More → **Digits** | ε
Digit → 0 | 1 | 2 | ... | 9

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

FIRST Sets with ε

Num → **Sign Digits**
Sign → + | - | ε
Digits → Digit More
More → Digits | ε
Digit → 0 | 1 | 2 | ... | 9

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

FIRST Sets with ε

Num → **Sign Digits**
Sign → + | - | ε
Digits → Digit More
More → Digits | ε
Digit → 0 | 1 | 2 | ... | 9

If Sign is ε

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

FIRST Sets with ε

Num → **Sign Digits**
Sign → + | - | ε
Digits → **Digit More**
More → **Digits** | ε
Digit → 0 | 1 | 2 | ... | 9

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

FIRST and ε

- When computing FIRST sets in a grammar with ε-productions, we often have to "look through" nonterminals.
- Rationale: Might have a derivation like this:

$$A \Rightarrow Bt \Rightarrow t$$

- So $t \in \text{FIRST}(A)$.

FIRST Computation with ε

- Initially, for all nonterminals A, set

$$\text{FIRST}(A) = \{\ t \mid A \rightarrow t\omega \text{ for some } \omega\ \}$$

- For all nonterminals A where $A \rightarrow \varepsilon$ is a production, add ε to FIRST(A).
- Repeat the following until no changes occur:
  - For each production $A \rightarrow \alpha$, where α is a string of nonterminals whose FIRST sets contain ε, set FIRST(A) = FIRST(A) ∪ { ε }.
  - For each production $A \rightarrow \alpha t\omega$, where α is a string of nonterminals whose FIRST sets contain ε, set

$$\text{FIRST}(A) = \text{FIRST}(A) \cup \{\ t\ \}$$

  - For each production $A \rightarrow \alpha B\omega$, where α is string of nonterminals whose FIRST sets contain ε, set

$$\text{FIRST}(A) = \text{FIRST}(A) \cup (\text{FIRST}(B) - \{\ \varepsilon\ \}).$$

A Notational Diversion

- Once we have computed the correct FIRST sets for each nonterminal, we can generalize our definition of FIRST sets to strings.
- Define FIRST*(ω) as follows:
  - FIRST*(ε) = { ε }
  - FIRST*(tω) = { t }
  - If ε ∉ FIRST(A):
    - FIRST*(Aω) = FIRST(A)
  - If ε ∈ FIRST(A):
    - FIRST*(Aω) = (FIRST(A) - { ε }) ∪ FIRST*(ω)

FIRST Computation with ε

- Initially, for all nonterminals A, set

  $$\text{FIRST}(A) = \{\, t \mid A \rightarrow t\omega \text{ for some } \omega \,\}$$

- For all nonterminals A where $A \rightarrow \varepsilon$ is a production,
  - add ε to FIRST(A).
- Repeat the following until no changes occur:
  - For each production $A \rightarrow \alpha$, set

    $$\text{FIRST}(A) = \text{FIRST}(A) \cup \text{FIRST}^*(\alpha)$$

LL(1) Tables with ε

Msg → Hi End

Hi → hello | heya | yo

End → world! | ε

LL(1) Tables with ε

Msg → Hi End

Hi → hello | heya | yo

End → world! | ε

	hello	heya	yo	world!
Msg				
Hi				
End				

LL(1) Tables with ε

Msg → Hi End

Hi → hello | heya | yo

End → world! | ε

Msg	Hi	End

	hello	heya	yo	world!
Msg				
Hi				
End				

LL(1) Tables with ε

Msg → Hi End

Hi → hello | heya | yo

End → world! | ε

Msg	Hi	End
	hello heya yo	

	hello	heya	yo	world!
Msg				
Hi				
End				

LL(1) Tables with ε

Msg → Hi End

Hi → hello | heya | yo

End → world! | ε

Msg	Hi	End
	hello heya yo	world ε

	hello	heya	yo	world!
Msg				
Hi				
End				

LL(1) Tables with ε

Msg → Hi End

Hi → hello | heya | yo

End → world! | ε

Msg	Hi	End
	hello heya yo	world ε

	hello	heya	yo	world!
Msg				
Hi				
End				

LL(1) Tables with ε

Msg → Hi End

Hi → hello | heya | yo

End → world! | ε

Msg	Hi	End
hello heya yo	hello heya yo	world ε

	hello	heya	yo	world!
Msg				
Hi				
End				

LL(1) Tables with ε

Msg → Hi End

Hi → hello | heya | yo

End → world! | ε

Msg	Hi	End
hello heya yo	hello heya yo	world ε

	hello	heya	yo	world!
Msg				
Hi				
End				

LL(1) Tables with ε

Msg → Hi End

Hi → hello | heya | yo

End → world! | ε

Msg	Hi	End
hello heya yo	hello heya yo	world ε

	hello	heya	yo	world!
Msg				
Hi				
End				

LL(1) Tables with ε

Msg → Hi End

Hi → hello | heya | yo

End → world! | ε

Msg	Hi	End
hello heya yo	hello heya yo	world ε

	hello	heya	yo	world!
Msg	**Hi End**	**Hi End**	**Hi End**	
Hi				
End				

LL(1) Tables with ε

Msg → Hi End

Hi → hello | heya | yo

End → world! | ε

Msg	Hi	End
hello heya yo	hello heya yo	world ε

	hello	heya	yo	world!
Msg	**Hi End**	**Hi End**	**Hi End**	
Hi				
End				

LL(1) Tables with ε

Msg → Hi End

Hi → hello | heya | yo

End → world! | ε

Msg	Hi	End
hello heya yo	hello heya yo	world ε

	hello	heya	yo	world!
Msg	**Hi End**	**Hi End**	**Hi End**	
Hi	hello	heya	yo	
End				

LL(1) Tables with ε

Msg → Hi End

Hi → hello | heya | yo

End → world! | ε

Msg	Hi	End
hello heya yo	hello heya yo	world ε

	hello	heya	yo	world!
Msg	**Hi End**	**Hi End**	**Hi End**	
Hi	hello	heya	yo	
End				

LL(1) Tables with ε

Msg → Hi End

Hi → hello | heya | yo

End → world! | ε

Msg	Hi	End
hello heya yo	hello heya yo	world ε

	hello	heya	yo	world!
Msg	**Hi End**	**Hi End**	**Hi End**	
Hi	hello	heya	yo	
End				world!

LL(1) Tables with ε

Msg → Hi End

Hi → hello | heya | yo

End → world! | ε

Msg	Hi	End
hello heya yo	hello heya yo	world ε

	hello	heya	yo	world!
Msg	**Hi End**	**Hi End**	**Hi End**	
Hi	hello	heya	yo	
End				world!

LL(1) Tables with ε

	hello	heya	yo	world!
Msg	**Hi End**	**Hi End**	**Hi End**	
Hi	hello	heya	yo	
End				world!

LL(1) Tables with ε

Msg $	hello $

	hello	heya	yo	world!
Msg	**Hi End**	**Hi End**	**Hi End**	
Hi	hello	heya	yo	
End				world!

LL(1) Tables with ε

Msg $	hello $

	hello	heya	yo	world!
Msg	**Hi End**	**Hi End**	**Hi End**	
Hi	hello	heya	yo	
End				world!

LL(1) Tables with ε

Msg $	hello $
Hi End $	hello $

	hello	heya	yo	world!
Msg	**Hi End**	**Hi End**	**Hi End**	
Hi	hello	heya	yo	
End				world!

LL(1) Tables with ε

Msg $	hello $
Hi End $	hello $

	hello	heya	yo	world!
Msg	**Hi End**	**Hi End**	**Hi End**	
Hi	hello	heya	yo	
End				world!

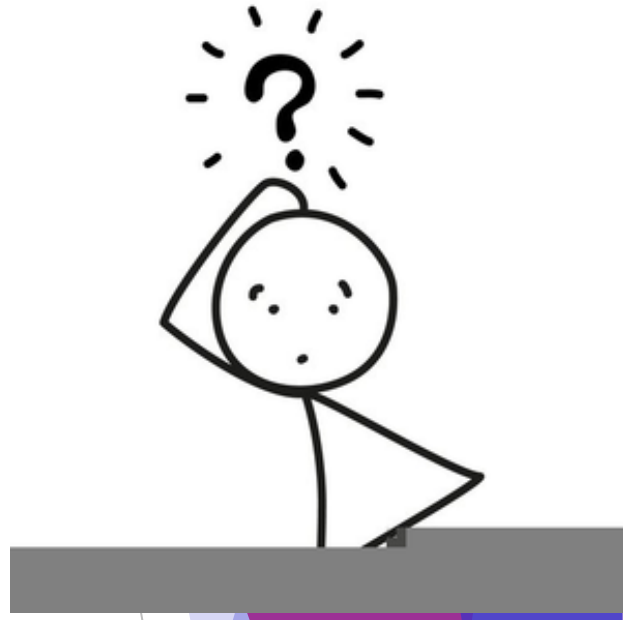
LL(1) Tables with ε

Msg $	hello $
Hi End $	hello $
hello End $	hello $

	hello	heya	yo	world!
Msg	**Hi End**	**Hi End**	**Hi End**	
Hi	hello	heya	yo	
End				world!

LL(1) Tables with ε

Msg $	hello $
Hi End $	hello $
hello End $	hello $
End $	$



	hello	heya	yo	world!
Msg	**Hi End**	**Hi End**	**Hi End**	
Hi	hello	heya	yo	
End				world!

ε is Complicated

- When constructing LL(1) tables with ε-productions, we need to have an extra column for **$**.

  Msg → Hi End

  Hi → hello | heya | yo

  End → world! | ε

	hello	heya	yo	world!
Msg	**Hi End**	**Hi End**	**Hi End**	
Hi	hello	heya	yo	
End				world!

LL(1) Tables with ε

LL(1) Tables with ε

Msg $	hello $
Hi End $	hello $
hello End $	hello $
End $	$

LL(1) Tables with ε

Msg → Hi End
Hi → hello | heya | yo
End → world! | ε

Msg $	hello $
Hi End $	hello $
hello End $	hello $
End $	$

	hello	heya	yo	world!	$
Msg	Hi End	Hi End	Hi End		
Hi	hello	heya	yo		
End				world!	ε

LL(1) Tables with ε

Msg → Hi End
Hi → hello | heya | yo
End → world! | ε

Msg $	hello $
Hi End $	hello $
hello End $	hello $
End $	$

	hello	heya	yo	world!	$
Msg	Hi End	Hi End	Hi End		
Hi	hello	heya	yo		
End				world!	ε

LL(1) Tables with ε

Msg → Hi End
Hi → hello | heya | yo
End → world! | ε

Msg $	hello $
Hi End $	hello $
hello End $	hello $
End $	$
$	$

	hello	heya	yo	world!	$
Msg	Hi End	Hi End	Hi End		
Hi	hello	heya	yo		
End				world!	ε


Q
A

It Gets Trickier

Num → **Sign Digits**

Sign → **+** | **-** | **ε**

Digits → **Digit More**

More → **Digits** | **ε**

Digit → **0** | **1** | … | **9**

It Gets Trickier

Num → **Sign Digits**

Sign → **+** | **-** | **ε**

Digits → **Digit More**

More → **Digits** | **ε**

Digit → **0** | **1** | … | **9**

	+	-	#	$
Num				
Sign				
Digits				
More				
Digit				

It Gets Trickier

Num → Sign Digits

Sign → + | - | ε

Digits → Digit More

More → Digits | ε

Digit → 0 | 1 | … | 9

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

	+	-	#	$
Num				
Sign				
Digits				
More				
Digit				

It Gets Trickier

Num → Sign Digits

Sign → + | - | ε

Digits → Digit More

More → Digits | ε

Digit → 0 | 1 | … | 9

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

	+	-	#	$
Num				
Sign				
Digits				
More				
Digit				

It Gets Trickier

Num → Sign Digits

Sign → + | - | ε

Digits → Digit More

More → Digits | ε

Digit → 0 | 1 | … | 9

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

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

It Gets Trickier

Num → Sign Digits

Sign → + | - | ε

Digits → Digit More

More → Digits | ε

Digit → 0 | 1 | … | 9

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

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

It Gets Trickier

Num → Sign Digits

Sign → + | - | ε

Digits → Digit More

More → Digits | ε

Digit → 0 | 1 | … | 9

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

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

It Gets Trickier

Num → Sign Digits

Sign → + | - | ε

Digits → Digit More

More → Digits | ε

Digit → 0 | 1 | … | 9

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

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

It Gets Trickier

Num → Sign Digits

Sign → + | - | ε

Digits → Digit More

More → Digits | ε

Digit → 0 | 1 | … | 9

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

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

It Gets Trickier

Num → Sign Digits

Sign → + | - | ε

Digits → Digit More

More → Digits | ε

Digit → 0 | 1 | … | 9

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

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

It Gets Trickier

Num → Sign Digits

Sign → + | - | ε

Digits → Digit More

More → Digits | ε

Digit → 0 | 1 | … | 9

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

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

It Gets Trickier

Num → Sign Digits

Sign → + | - | ε

Digits → Digit More

More → Digits | ε

Digit → 0 | 1 | … | 9

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

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

It Gets Trickier

Num → Sign Digits

Sign → + | - | ε

Digits → Digit More

More → Digits | ε

Digit → 0 | 1 | … | 9

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

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

It Gets Trickier

Num → Sign Digits

Sign → + | - | ε

Digits → Digit More

More → Digits | ε

Digit → 0 | 1 | … | 9

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

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

It Gets Trickier

Sign contains ε therefore
Add first(Digits) to first(Num)

Num → Sign Digits

Sign → + | - | ε

Digits → Digit More

More → Digits | ε

Digit → 0 | 1 | … | 9

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

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

It Gets Trickier

Num → Sign Digits

Sign → + | - | ε

Digits → Digit More

More → Digits | ε

Digit → 0 | 1 | … | 9

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

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

It Gets Trickier

Num → Sign Digits

Sign → + | - | ε

Digits → Digit More

More → Digits | ε

Digit → 0 | 1 | … | 9

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

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

It Gets Trickier

Num → Sign Digits

Sign → + | - | ε

Digits → Digit More

More → Digits | ε

Digit → 0 | 1 | … | 9

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

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

It Gets Trickier

Num → Sign Digits

Sign → + | - | ε

Digits → Digit More

More → Digits | ε

Digit → 0 | 1 | … | 9

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

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

It Gets Trickier

Num → Sign Digits

Sign → + | - | ε

Digits → Digit More

More → Digits | ε

Digit → 0 | 1 | … | 9

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

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

It Gets Trickier

Num → Sign Digits

Sign → + | - | ε

Digits → Digit More

More → Digits | ε

Digit → 0 | 1 | … | 9

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

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

It Gets Trickier

Num → Sign Digits

Sign → + | - | ε

Digits → Digit More

More → Digits | ε

Digit → 0 | 1 | … | 9

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

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

It Gets Trickier

Num → Sign Digits

Sign → + | - | ε

Digits → Digit More

More → Digits | ε

Digit → 0 | 1 | … | 9

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

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


FOLLOW Sets

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:

- FOLLOW(A) = { t | S ⇒* αAtω for some α, ω } where S is the start symbol of the grammar.
- Informally, every terminal 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 $A \rightarrow \omega$, for each terminal $t \in$ FIRST*(ω), set T[A, t] = ω.
- Note that ε *is not a terminal*.
- For each rule $A \rightarrow \omega$, if $\varepsilon \in$ FIRST*(ω), set T[A, t] = ω for each $t \in$ FOLLOW(A).

Notes

- ε may *appear in the first function* of a non-terminal.
- ε will *never appear in the follow* function of a non-terminal.
- We calculate the follow function of a non-terminal by looking where it is present on the RHS of a production rule.
- Before calculating the first and follow functions, eliminate Left Recursion from the grammar, if present.

Left Factoring

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 → Ab A → 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 → 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

$$A \rightarrow A\omega \mid \alpha$$

- This will produce α followed by some number of ω's.
- Can rewrite the grammar as

$$A \rightarrow \alpha B$$

$$B \rightarrow \varepsilon \mid \omega B$$

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

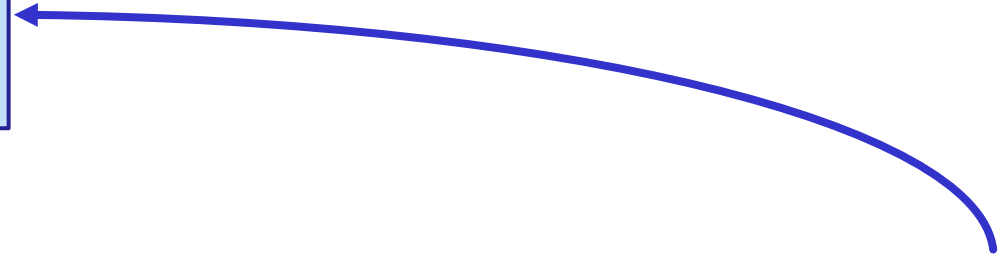
- Consider the following grammar:

$E \rightarrow T$

$E \rightarrow T + E$

$T \rightarrow int$

$T \rightarrow (E)$



How do you predict which of these to use?

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

Left-Factoring

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

Left-Factoring

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

Left-Factoring

$E \rightarrow TY$

$E \rightarrow T + E$

$T \rightarrow int$

$T \rightarrow (E)$

Left-Factoring

$E \rightarrow TY$

$T \rightarrow int$

$T \rightarrow (E)$

$Y \rightarrow +E$

$Y \rightarrow \varepsilon$

$E \rightarrow T + E$

Left-Factoring

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

Left-Factoring

$E \rightarrow TY$	1
$T \rightarrow int$	2
$T \rightarrow (E)$	3
$Y \rightarrow + E$	4
$Y \rightarrow \varepsilon$	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
$)	+ $)	$)

If ε ∈ FIRST(Y)
FOLLOW(T) = FOLLOW(T) ∪ FOLLOW(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**				
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

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.

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.
- **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.

Summary of Rules of First Sets

- **Essential conditions to check first are:**
  1. The grammar is free from left recursion.
  2. The grammar should not be ambiguous.
  3. The grammar should be left factored in so that the grammar is deterministic grammar.

Summary of Rules of First Sets

- If there is a Production X → ε, add ε to First(X) = { ε }
- If there is a Production X → $Y_1Y_2..Y_k$ then add first($Y_1Y_2..Y_k$) to first(X).

  First($Y_1Y_2..Y_k$) is either:
  - if First(Y_1) doesn't contain ε, then First($Y_1Y_2..Y_k$) = First(Y_1)
  - If First(Y_1) does contain ε then First ($Y_1Y_2..Y_k$) is everything in First(Y_1) <except for ε > as well as everything in First($Y_2..Y_k$)
  - If First(Y_1) First(Y_2)..First(Y_k) all contain ε then add ε to First($Y_1Y_2..Y_k$) as well.

Summary of Rules of Follow Sets

- For the start symbol S, place $ in Follow(S).
- For any production rule $A \rightarrow \alpha B$,
  - Follow(B) = Follow(A)
- For any production rule $A \rightarrow \alpha B \beta$,
  - If $\varepsilon \notin$ First(β), then Follow(B) = First(β)
  - If $\varepsilon \in$ First(β), then Follow(B) = { First(β) – ε } $\cup$ Follow(A)


Q
A

Review Questions

Problem 1

- Find the first and follow of the following productions:

$S \rightarrow (A) \mid \varepsilon$

$A \rightarrow TE$

$E \rightarrow \&TE \mid \varepsilon$

$T \rightarrow (A) \mid a \mid b \mid c$

Solution

$S \rightarrow (A) \mid \varepsilon$

$A \rightarrow TE$

$E \rightarrow \&TE \mid \varepsilon$

$T \rightarrow (A) \mid a \mid b \mid c$

Non-Terminal	First	Follow
S	{ (, ε }	{ $ }
A	{ (, a, b, c }	{) }
E	{ &, ε }	Follow(A) = {) }
T	{ (, a, b, c }	First(E) ∪ Follow(E) = { &,) }

Problem 2

Find the first and follow of the following productions:

$E \rightarrow TE'$

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

$T \rightarrow FT'$

$T' \rightarrow *FT' \mid \varepsilon$

$F \rightarrow (E) \mid id$

Solution

$E \rightarrow TE'$
$E' \rightarrow +TE' \mid \varepsilon$
$T \rightarrow FT'$
$T' \rightarrow *FT' \mid \varepsilon$
$F \rightarrow (E) \mid id$

Solution

E → TE'
E' → +TE' | ε
T → FT'
T' → *FT' | ε
F → (E) | id

First(E) = first(T) = { (, id }
First(E') = { +, ε }
First(T) = first(F) = { (, id }
First(T') = { *, ε }
First(F) = { (, id }

Follow(E) = {$} ∪ {) } = { $,) }
Follow(E') = follow(E) = { $,) }
Follow(T) = first(E') ∪ follow(E') = { +, $,) }
Follow(T') = follow(T) = { +, $,) }
Follow(F) = first(T') ∪ follow(T') = { *, +, $,) }

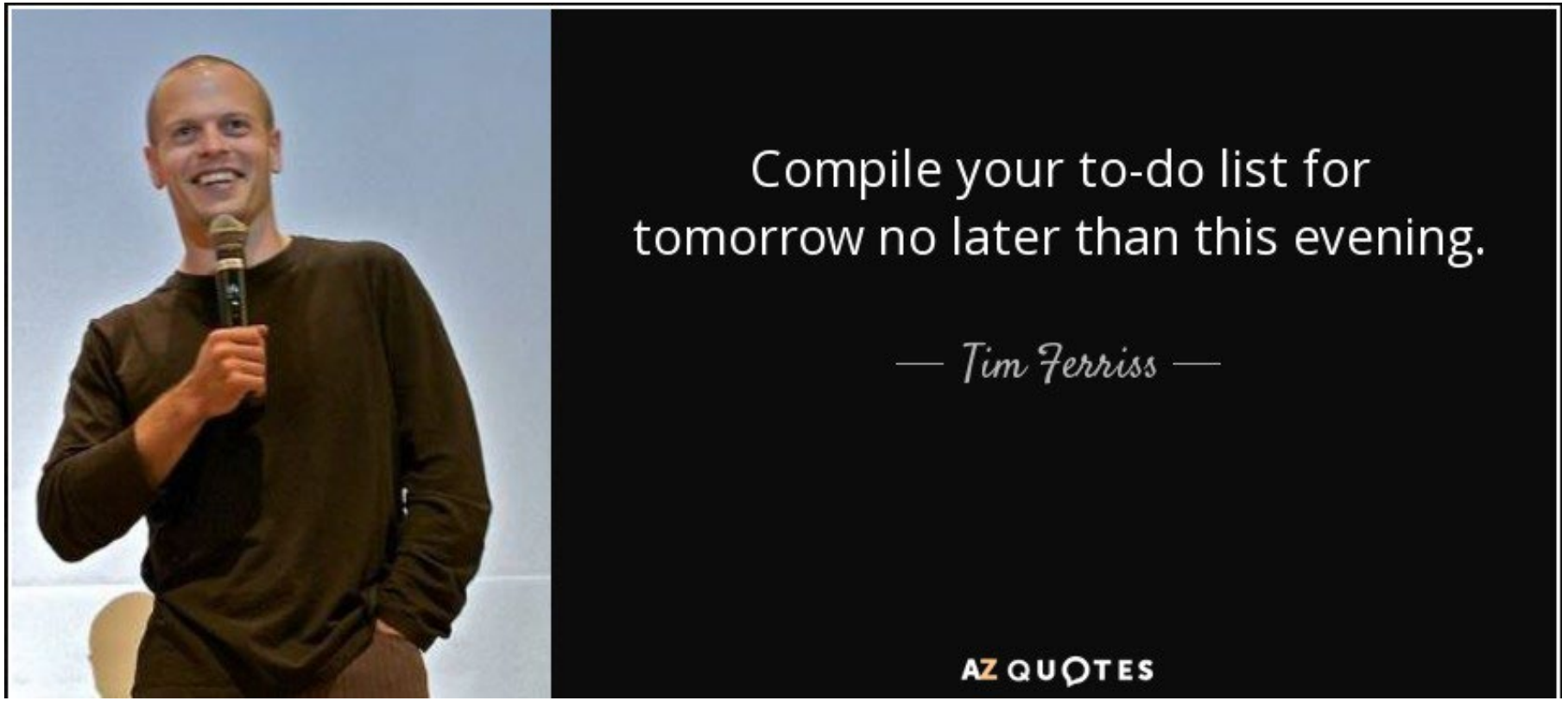
Useful links

- Compiler Design Lecture 7 -- Construction of LL(1) parsing table https://www.youtube.com/watch?v=R1ZlWEZWMKk

- Compiler Design Lecture 6 -- Examples on how to find first and follow in LL(1) https://www.youtube.com/watch?v=_uSlP91jmTM&t=21s

References of this lecture

- Presentation slides of the book: COMPILER CONSTRUCTION, Principles and Practice, by Kenneth C. Louden
- Credits for Dr. Sally Saad, Prof. Mostafa Aref, Dr. Islam Hegazy, and Dr. Abd ElAziz for help in content preparation and aggregation (FCIS-ASU)



See you next lecture