Introduction to Compiler Construction

Parsing: Top-down LL(1) Algorithm

Outline

LL(1) Parsing

LL(1) Parsing

The first L in the name indicates a left-to-right scan of the input

LL(1) Parsing

The first L in the name indicates a left-to-right scan of the input

The second L indicates that the algorithm produces a left-most derivation

LL(1) Parsing

The first L in the name indicates a left-to-right scan of the input

The second L indicates that the algorithm produces a left-most derivation

The 1 indicates a lookahead of a single token

LL(1) Parsing

The first L in the name indicates a left-to-right scan of the input

The second L indicates that the algorithm produces a left-most derivation

The 1 indicates a lookahead of a single token

At the start, the start symbol S is pushed onto a stack

LL(1) Parsing

The first L in the name indicates a left-to-right scan of the input

The second L indicates that the algorithm produces a left-most derivation

The 1 indicates a lookahead of a single token

At the start, the start symbol S is pushed onto a stack

The parser continues by parsing each symbol as it is removed from the top of the stack

- If the symbol is a terminal, it scans a token from the input; if they do not match, an error is raised
- If the symbol is a non-terminal, the input token is used to decide which rule to apply to replace that non-terminal

LL(1) Parsing

LL(1) Parsing

LL(1) parsing technique is table-driven, with a unique parse table produced for each grammar

LL(1) Parsing

LL(1) parsing technique is table-driven, with a unique parse table produced for each grammar

The parse table has a row for each non-terminal and a column for each terminal, including a special terminator # to mark the end of the sentence

LL(1) Parsing

LL(1) parsing technique is table-driven, with a unique parse table produced for each grammar

The parse table has a row for each non-terminal and a column for each terminal, including a special terminator # to mark the end of the sentence

The parser consults this table, given the non-terminal on top of the stack and the next input token to determine which rule to use in replacing the non-terminal

LL(1) Parsing

LL(1) parsing technique is table-driven, with a unique parse table produced for each grammar

The parse table has a row for each non-terminal and a column for each terminal, including a special terminator `#` to mark the end of the sentence

The parser consults this table, given the non-terminal on top of the stack and the next input token to determine which rule to use in replacing the non-terminal

No table entry may contain more than one rule

LL(1) Parsing · Example (Parse Table for an Arithmetic Expression Grammar)

LL(1) Parsing · Example (Parse Table for an Arithmetic Expression Grammar)

1. $E \ ::= T \ E'$
2. $E' ::= + \ T \ E'$
3. $E' ::= \epsilon$
4. $T \ ::= F \ T'$
5. $T' ::= * \ F \ T'$
6. $T' ::= \epsilon$
7. $F \ ::= (E)$
8. $F \ ::= \texttt{id}$

LL(1) Parsing · Example (Parse Table for an Arithmetic Expression Grammar)

1. $E ::= T\ E'$
2. $E' ::= \texttt{+}\ T\ E'$
3. $E' ::= \epsilon$
4. $T ::= F\ T'$
5. $T' ::= \texttt{*}\ F\ T'$
6. $T' ::= \epsilon$
7. $F ::= \texttt{(}E\texttt{)}$
8. $F ::= \texttt{id}$

LL(1) parse table for the grammar

	+	*	(	)	id	#
E			1		1	
E'	2			3		3
T			4		4	
T'	6	5		6		6
F			7		8	

LL(1) Parsing

LL(1) Parsing

Input: parse table *table*, productions *rules*, and a sentence *w*
Output: a left-most derivation for *w*

```
stk ← Stack(#, S)
sym ← first symbol in w#
while true do
    top ← stk.pop()
    if top = sym = # then
        Halt successfully
    else if top is a terminal then
        if top = sym then
            Advance sym to be the next symbol in w#
        else
            Halt with an error: sym found where top was expected
        end if
    else if top is a non-terminal Y then
        index ← table[Y, sym]
        if index ≠ err then
            rule ← rules[index]
            If Y ::= X_1 X_2 ... X_n, then stk.push(X_n, ..., X_2, X_1)
        else
            Halt with an error: no rule to follow
        end if
    end if
end while
```

LL(1) Parsing · Example (Parsing `id+id*id`)

LL(1) Parsing · Example (Parsing `id+id*id`)

1. $E ::= T\ E'$
2. $E' ::= \texttt{+}\ \mathrm{T}\ E'$
3. $E' ::= \epsilon$
4. $T ::= F\ T'$
5. $T' ::= \texttt{*}\ F\ T'$
6. $T' ::= \epsilon$
7. $F ::= \texttt{(}E\texttt{)}$
8. $F ::= \texttt{id}$

	+	*	(	)	id	#
E			1		1	
E'	2			3		3
T			4		4	
T'	6	5		6		6
F			7		8	

LL(1) Parsing · Example (Parsing `id+id*id`)

1. $E \ ::= \ T \ E'$
2. $E' \ ::= \ \texttt{+} \ \mathrm{T} \ E'$
3. $E' \ ::= \ \epsilon$
4. $T \ ::= \ F \ T'$
5. $T' \ ::= \ \texttt{*} \ F \ T'$
6. $T' \ ::= \ \epsilon$
7. $F \ ::= \ \texttt{(} E \texttt{)}$
8. $F \ ::= \ \texttt{id}$

	+	*	(	)	id	#
E			1		1	
E'	2			3		3
T			4		4	
T'	6	5		6		6
F			7		8	

Stack	Input	Output

LL(1) Parsing · Example (Parsing `id+id*id`)

1. E ::= $T\ E'$
2. E' ::= + T E'
3. E' ::= ϵ
4. T ::= $F\ T'$
5. T' ::= * $F\ T'$
6. T' ::= ϵ
7. F ::= (E)
8. F ::= id

	+	*	(	)	id	#
E			1		1	
E'	2			3		3
T			4		4	
T'	6	5		6		6
F			7		8	

Stack	Input	Output
#E	id+id*id#	1

LL(1) Parsing · Example (Parsing `id+id*id`)

1. $E ::= T\ E'$
2. $E' ::= +\ T\ E'$
3. $E' ::= \epsilon$
4. $T ::= F\ T'$
5. $T' ::= *\ F\ T'$
6. $T' ::= \epsilon$
7. $F ::= (E)$
8. $F ::= \mathtt{id}$

	+	*	(	)	id	#
E			1		1	
E'	2			3		3
T			4		4	
T'	6	5		6		6
F			7		8	

Stack	Input	Output
$\#E$	id+id*id#	1
$\#E'\,T$	id+id*id#	4

LL(1) Parsing · Example (Parsing `id+id*id`)

1. $E ::= T\ E'$
2. $E' ::= +\ T\ E'$
3. $E' ::= \epsilon$
4. $T ::= F\ T'$
5. $T' ::= *\ F\ T'$
6. $T' ::= \epsilon$
7. $F ::= (E)$
8. $F ::= \mathtt{id}$

	+	*	(	)	id	#
E			1		1	
E'	2			3		3
T			4		4	
T'	6	5		6		6
F			7		8	

Stack	Input	Output
$\#E$	id+id*id#	1
$\#E'T$	id+id*id#	4
$\#E'T'F$	id+id*id#	8

LL(1) Parsing · Example (Parsing `id+id*id`)

1. $E ::= T\ E'$
2. $E' ::= \texttt{+}\ \mathrm{T}\ E'$
3. $E' ::= \epsilon$
4. $T ::= F\ T'$
5. $T' ::= \texttt{*}\ F\ T'$
6. $T' ::= \epsilon$
7. $F ::= \texttt{(}E\texttt{)}$
8. $F ::= \texttt{id}$

	+	*	(	)	id	#
E			1		1	
E'	2			3		3
T			4		4	
T'	6	5		6		6
F			7		8	

Stack	Input	Output
$\#E$	`id+id*id#`	1
$\#E'\,T$	`id+id*id#`	4
$\#E'\,T'\,F$	`id+id*id#`	8
$\#E'\,T'\,\texttt{id}$	`id+id*id#`	

LL(1) Parsing · Example (Parsing `id+id*id`)

1. $E ::= T\ E'$
2. $E' ::= \texttt{+}\ \mathrm{T}\ E'$
3. $E' ::= \epsilon$
4. $T ::= F\ T'$
5. $T' ::= \texttt{*}\ F\ T'$
6. $T' ::= \epsilon$
7. $F ::= \texttt{(}E\texttt{)}$
8. $F ::= \texttt{id}$

	+	*	(	)	id	#
E			1		1	
E'	2			3		3
T			4		4	
T'	6	5		6		6
F			7		8	

Stack	Input	Output
$\#E$	id+id*id#	1
$\#E'\ T$	id+id*id#	4
$\#E'\ T'\ F$	id+id*id#	8
$\#E'\ T'\ \texttt{id}$	id+id*id#	
$\#E'\ T'$	+id*id#	6

LL(1) Parsing · Example (Parsing `id+id*id`)

1. $E ::= T\ E'$
2. $E' ::= \texttt{+}\ \mathrm{T}\ E'$
3. $E' ::= \epsilon$
4. $T ::= F\ T'$
5. $T' ::= \texttt{*}\ F\ T'$
6. $T' ::= \epsilon$
7. $F ::= \texttt{(}E\texttt{)}$
8. $F ::= \texttt{id}$

	+	*	(	)	id	#
E			1		1	
E'	2			3		3
T			4		4	
T'	6	5		6		6
F			7		8	

Stack	Input	Output
$\#E$	id+id*id#	1
$\#E'\ T$	id+id*id#	4
$\#E'\ T'\ F$	id+id*id#	8
$\#E'\ T'\ \texttt{id}$	id+id*id#	
$\#E'\ T'$	+id*id#	6
$\#E'$	+id*id#	2

LL(1) Parsing · Example (Parsing `id+id*id`)

1. $E ::= T\ E'$
2. $E' ::= +\ T\ E'$
3. $E' ::= \epsilon$
4. $T ::= F\ T'$
5. $T' ::= *\ F\ T'$
6. $T' ::= \epsilon$
7. $F ::= (E)$
8. $F ::= \mathtt{id}$

	+	*	(	)	id	#
E			1		1	
E'	2			3		3
T			4		4	
T'	6	5		6		6
F			7		8	

Stack	Input	Output
$\#E$	id+id*id#	1
$\#E'\ T$	id+id*id#	4
$\#E'\ T'\ F$	id+id*id#	8
$\#E'\ T'\ \mathtt{id}$	id+id*id#	
$\#E'\ T'$	+id*id#	6
$\#E'$	+id*id#	2
$\#E'\ T +$	+id*id#	

LL(1) Parsing · Example (Parsing id+id*id)

1. $E ::= T\ E'$
2. $E' ::= +\ T\ E'$
3. $E' ::= \epsilon$
4. $T ::= F\ T'$
5. $T' ::= *\ F\ T'$
6. $T' ::= \epsilon$
7. $F ::= (E)$
8. $F ::= \texttt{id}$

	+	*	(	)	id	#
E			1		1	
E'	2			3		3
T			4		4	
T'	6	5		6		6
F			7		8	

Stack	Input	Output
$\#E$	id+id*id#	1
$\#E'\ T$	id+id*id#	4
$\#E'\ T'\ F$	id+id*id#	8
$\#E'\ T'\ \texttt{id}$	id+id*id#	
$\#E'\ T'$	+id*id#	6
$\#E'$	+id*id#	2
$\#E'\ T+$	+id*id#	
$\#E'\ T$	id*id#	4

LL(1) Parsing · Example (Parsing `id+id*id`)

1. $E ::= T\ E'$
2. $E' ::=$ + T E'
3. $E' ::= \epsilon$
4. $T ::= F\ T'$
5. $T' ::= * \ F\ T'$
6. $T' ::= \epsilon$
7. $F ::=$ (E)
8. $F ::=$ id

	+	*	(	)	id	#
E			1		1	
E'	2			3		3
T			4		4	
T'	6	5		6		6
F			7		8	

Stack	Input	Output
#E	id+id*id#	1
#$E'\ T$	id+id*id#	4
#$E'\ T'\ F$	id+id*id#	8
#$E'\ T'$ id	id+id*id#	
#$E'\ T'$	+id*id#	6
#E'	+id*id#	2
#$E'\ T$+	+id*id#	
#$E'\ T$	id*id#	4
#$E'\ T'\ F$	id*id#	8

LL(1) Parsing · Example (Parsing `id+id*id`)

1. $E ::= T\ E'$
2. $E' ::= + \ T\ E'$
3. $E' ::= \epsilon$
4. $T ::= F\ T'$
5. $T' ::= *\ F\ T'$
6. $T' ::= \epsilon$
7. $F ::= (E)$
8. $F ::= \mathtt{id}$

	+	*	(	)	id	#
E			1		1	
E'	2			3		3
T			4		4	
T'	6	5		6		6
F			7		8	

Stack	Input	Output
$\#E$	id+id*id#	1
$\#E'\ T$	id+id*id#	4
$\#E'\ T'\ F$	id+id*id#	8
$\#E'\ T'\ \mathtt{id}$	id+id*id#	
$\#E'\ T'$	+id*id#	6
$\#E'$	+id*id#	2
$\#E'\ T+$	+id*id#	
$\#E'\ T$	id*id#	4
$\#E'\ T'\ F$	id*id#	8
$\#E'\ T'\ \mathtt{id}$	id*id#	

LL(1) Parsing · Example (Parsing `id+id*id`)

1. $E ::= T\ E'$
2. $E' ::= {+}\ \mathrm{T}\ E'$
3. $E' ::= \epsilon$
4. $T ::= F\ T'$
5. $T' ::= {*}\ F\ T'$
6. $T' ::= \epsilon$
7. $F ::= (E)$
8. $F ::= \mathtt{id}$

	+	*	(	)	id	#
E			1		1	
E'	2			3		3
T			4		4	
T'	6	5		6		6
F			7		8	

Stack	Input	Output
$\#E$	id+id*id#	1
$\#E'\,T$	id+id*id#	4
$\#E'\,T'\,F$	id+id*id#	8
$\#E'\,T'\,\mathtt{id}$	id+id*id#	
$\#E'\,T'$	+id*id#	6
$\#E'$	+id*id#	2
$\#E'\,T{+}$	+id*id#	
$\#E'\,T$	id*id#	4
$\#E'\,T'\,F$	id*id#	8
$\#E'\,T'\,\mathtt{id}$	id*id#	
$\#E'\,T'$	*id#	5

LL(1) Parsing · Example (Parsing `id+id*id`)

1. $E ::= T\ E'$
2. $E' ::= +\ T\ E'$
3. $E' ::= \epsilon$
4. $T ::= F\ T'$
5. $T' ::= *\ F\ T'$
6. $T' ::= \epsilon$
7. $F ::= (E)$
8. $F ::= \mathtt{id}$

	+	*	(	)	id	#
E			1		1	
E'	2			3		3
T			4		4	
T'	6	5		6		6
F			7		8	

Stack	Input	Output
$\#E$	id+id*id#	1
$\#E'T$	id+id*id#	4
$\#E'T'F$	id+id*id#	8
$\#E'T'\mathtt{id}$	id+id*id#	
$\#E'T'$	+id*id#	6
$\#E'$	+id*id#	2
$\#E'T+$	+id*id#	
$\#E'T$	id*id#	4
$\#E'T'F$	id*id#	8
$\#E'T'\mathtt{id}$	id*id#	
$\#E'T'$	*id#	5
$\#E'T'F*$	*id#	

LL(1) Parsing · Example (Parsing `id+id*id`)

1. $E ::= T\ E'$
2. $E' ::= +\ T\ E'$
3. $E' ::= \epsilon$
4. $T ::= F\ T'$
5. $T' ::= *\ F\ T'$
6. $T' ::= \epsilon$
7. $F ::= (E)$
8. $F ::= \mathtt{id}$

	+	*	(	)	id	#
E			1		1	
E'	2			3		3
T			4		4	
T'	6	5		6		6
F			7		8	

Stack	Input	Output
$\#E$	id+id*id#	1
$\#E'\ T$	id+id*id#	4
$\#E'\ T'\ F$	id+id*id#	8
$\#E'\ T'\ \mathtt{id}$	id+id*id#	
$\#E'\ T'$	+id*id#	6
$\#E'$	+id*id#	2
$\#E'\ T+$	+id*id#	
$\#E'\ T$	id*id#	4
$\#E'\ T'\ F$	id*id#	8
$\#E'\ T'\ \mathtt{id}$	id*id#	
$\#E'\ T'$	*id#	5
$\#E'\ T'\ F*$	*id#	
$\#E'\ T'\ F$	id#	8

LL(1) Parsing · Example (Parsing `id+id*id`)

1. $E ::= T\ E'$
2. $E' ::= +\ T\ E'$
3. $E' ::= \epsilon$
4. $T ::= F\ T'$
5. $T' ::= *\ F\ T'$
6. $T' ::= \epsilon$
7. $F ::= (E)$
8. $F ::= \texttt{id}$

	+	*	(	)	id	#
E			1		1	
E'	2			3		3
T			4		4	
T'	6	5		6		6
F			7		8	

Stack	Input	Output
$\#E$	id+id*id#	1
$\#E'\ T$	id+id*id#	4
$\#E'\ T'\ F$	id+id*id#	8
$\#E'\ T'\ \texttt{id}$	id+id*id#	
$\#E'\ T'$	+id*id#	6
$\#E'$	+id*id#	2
$\#E'\ T+$	+id*id#	
$\#E'\ T$	id*id#	4
$\#E'\ T'\ F$	id*id#	8
$\#E'\ T'\ \texttt{id}$	id*id#	
$\#E'\ T'$	*id#	5
$\#E'\ T'\ F*$	*id#	
$\#E'\ T'\ F$	id#	8
$\#E'\ T'\ \texttt{id}$	id#	

LL(1) Parsing · Example (Parsing `id+id*id`)

1. $E ::= T\ E'$
2. $E' ::= +\ T\ E'$
3. $E' ::= \epsilon$
4. $T ::= F\ T'$
5. $T' ::= *\ F\ T'$
6. $T' ::= \epsilon$
7. $F ::= (E)$
8. $F ::= id$

	+	*	(	)	id	#
E			1		1	
E'	2			3		3
T			4		4	
T'	6	5		6		6
F			7		8	

Stack	Input	Output
$\#E$	id+id*id#	1
$\#E'\ T$	id+id*id#	4
$\#E'\ T'\ F$	id+id*id#	8
$\#E'\ T'$ id	id+id*id#	
$\#E'\ T'$	+id*id#	6
$\#E'$	+id*id#	2
$\#E'\ T\ +$	+id*id#	
$\#E'\ T$	id*id#	4
$\#E'\ T'\ F$	id*id#	8
$\#E'\ T'$ id	id*id#	
$\#E'\ T'$	*id#	5
$\#E'\ T'\ F\ *$	*id#	
$\#E'\ T'\ F$	id#	8
$\#E'\ T'$ id	id#	
$\#E'\ T'$	#	6

LL(1) Parsing · Example (Parsing `id+id*id`)

1. $E ::= T\ E'$
2. $E' ::= \texttt{+}\ \mathrm{T}\ E'$
3. $E' ::= \epsilon$
4. $T ::= F\ T'$
5. $T' ::= \texttt{*}\ F\ T'$
6. $T' ::= \epsilon$
7. $F ::= \texttt{(}E\texttt{)}$
8. $F ::= \texttt{id}$

	+	*	(	)	id	#
E			1		1	
E'	2			3		3
T			4		4	
T'	6	5		6		6
F			7		8	

Stack	Input	Output
$\#E$	id+id*id#	1
$\#E'\,T$	id+id*id#	4
$\#E'\,T'\,F$	id+id*id#	8
$\#E'\,T'\,\texttt{id}$	id+id*id#	
$\#E'\,T'$	+id*id#	6
$\#E'$	+id*id#	2
$\#E'\,T\,\texttt{+}$	+id*id#	
$\#E'\,T$	id*id#	4
$\#E'\,T'\,F$	id*id#	8
$\#E'\,T'\,\texttt{id}$	id*id#	
$\#E'\,T'$	*id#	5
$\#E'\,T'\,F\,\texttt{*}$	*id#	
$\#E'\,T'\,F$	id#	8
$\#E'\,T'\,\texttt{id}$	id#	
$\#E'\,T'$	#	6
$\#E'$	#	3

LL(1) Parsing · Example (Parsing `id+id*id`)

1. E ::= T E'
2. E' ::= + T E'
3. E' ::= ϵ
4. T ::= F T'
5. T' ::= * F T'
6. T' ::= ϵ
7. F ::= (E)
8. F ::= id

	+	*	(	)	id	#
E			1		1	
E'	2			3		3
T			4		4	
T'	6	5		6		6
F			7		8	

Stack	Input	Output
#E	id+id*id#	1
#E' T	id+id*id#	4
#E' T' F	id+id*id#	8
#E' T' id	id+id*id#	
#E' T'	+id*id#	6
#E'	+id*id#	2
#E' T +	+id*id#	
#E' T	id*id#	4
#E' T' F	id*id#	8
#E' T' id	id*id#	
#E' T'	*id#	5
#E' T' F *	*id#	
#E' T' F	id#	8
#E' T' id	id#	
#E' T'	#	6
#E'	#	3
#	#	✓

LL(1) Parsing

LL(1) Parsing

Assuming both α and β are (possibly empty) strings of terminals and non-terminals, table$[Y, a] = i$, where i is the number of the rule $Y ::= X_1 X_2 \ldots X_n$, if either

1. $X_1 X_2 \ldots X_n \overset{*}{\Rightarrow} a\alpha$, or
2. $X_1 X_2 \ldots X_n \overset{*}{\Rightarrow} \epsilon$, and there is a derivation $S_\# \overset{*}{\Rightarrow} \alpha Y a \beta$

LL(1) Parsing

Assuming both α and β are (possibly empty) strings of terminals and non-terminals, table$[Y, a] = i$, where i is the number of the rule $Y ::= X_1 X_2 \ldots X_n$, if either

1. $X_1 X_2 \ldots X_n \stackrel{*}{\Rightarrow} a\alpha$, or
2. $X_1 X_2 \ldots X_n \stackrel{*}{\Rightarrow} \epsilon$, and there is a derivation $S_\# \stackrel{*}{\Rightarrow} \alpha Y a \beta$

For this we need two helper functions, *first* and *follow*

First Set

First Set

first($X_1 X_2 \ldots X_n$) is the set of all terminals that can start strings derivable from $X_1 X_2 \ldots X_n$

First Set

first($X_1 X_2 \dots X_n$) is the set of all terminals that can start strings derivable from $X_1 X_2 \dots X_n$

Formally, first($X_1 X_2 \dots X_n$) $= \{a | X_1 X_2 \dots X_n \stackrel{*}{\Rightarrow} a\alpha, a \in T\}$

First Set

first$(X_1X_2 \ldots X_n)$ is the set of all terminals that can start strings derivable from $X_1X_2 \ldots X_n$

Formally, first$(X_1X_2 \ldots X_n) = \{a | X_1X_2 \ldots X_n \overset{*}{\Rightarrow} a\alpha, a \in T\}$

If $X_1X_2 \ldots X_n \overset{*}{\Rightarrow} \epsilon$, then we say that first$(X_1X_2 \ldots X_n)$ includes ϵ

First Set · First Set of a Single Symbol

Input: a context-free grammar $G = (N, T, S, P)$

Output: first(X) for all symbols $X \in T \cup N$

for $X \in T$ **do**
    first(X) $\leftarrow \{X\}$
end for
for $X \in N$ **do**
    first(X) $\leftarrow \{\}$
end for
if $X ::= \epsilon \in P$ **then**
    Add ϵ to first(X)
end if
repeat
    for $Y ::= X_1 X_2 \ldots X_n \in P$ **do**
        Add first($X_1 X_2 \ldots X_n$) to first(Y)
    end for
until no new symbols are added to any set

First Set · First Set of a Sequence of Symbols

Input: a context-free grammar $G = (N, T, S, P)$ and a sequence of symbols $X_1 X_2 \dots X_n$
Output: $\text{first}(X_1 X_2 \dots X_n)$

$F \leftarrow \text{first}(X_1)$
$i \leftarrow 2$
while $\epsilon \in F$ and $i \leq n$ **do**
    $F \leftarrow F - \epsilon$
    Add $\text{first}(X_i)$ to F
    $i \leftarrow i + 1$
end while
return F

First Set · Example (First Sets for the Arithmetic Expression Grammar)

First Set · Example (First Sets for the Arithmetic Expression Grammar)

1. $E ::= T\ E'$
2. $E' ::= \texttt{+}\ \mathrm{T}\ E'$
3. $E' ::= \epsilon$
4. $T ::= F\ T'$
5. $T' ::= \texttt{*}\ F\ T'$
6. $T' ::= \epsilon$
7. $F ::= \texttt{(}E\texttt{)}$
8. $F ::= \texttt{id}$

First Set · Example (First Sets for the Arithmetic Expression Grammar)

1. $E ::= T\ E'$
2. $E' ::= \texttt{+}\ \mathrm{T}\ E'$
3. $E' ::= \epsilon$
4. $T ::= F\ T'$
5. $T' ::= \texttt{*}\ F\ T'$
6. $T' ::= \epsilon$
7. $F ::= \texttt{(}E\texttt{)}$
8. $F ::= \texttt{id}$

$$\begin{aligned} \text{first}(E) &= \\ \text{first}(E') &= \\ \text{first}(T) &= \\ \text{first}(T') &= \\ \text{first}(F) &= \end{aligned}$$

First Set · Example (First Sets for the Arithmetic Expression Grammar)

1. $E ::= T\ E'$
2. $E' ::= + \mathrm{T}\ E'$
3. $E' ::= \epsilon$
4. $T ::= F\ T'$
5. $T' ::= *\ F\ T'$
6. $T' ::= \epsilon$
7. $F ::= (E)$
8. $F ::= \mathtt{id}$

$$\begin{aligned}
\text{first}(E) &= \{\} \\
\text{first}(E') &= \{\} \\
\text{first}(T) &= \{\} \\
\text{first}(T') &= \{\} \\
\text{first}(F) &= \{\}
\end{aligned}$$

First Set · Example (First Sets for the Arithmetic Expression Grammar)

1. $E ::= T\ E'$
2. $E' ::= \texttt{+}\ \mathrm{T}\ E'$
3. $E' ::= \epsilon$
4. $T ::= F\ T'$
5. $T' ::= \texttt{*}\ F\ T'$
6. $T' ::= \epsilon$
7. $F ::= \texttt{(}E\texttt{)}$
8. $F ::= \texttt{id}$

$$\mathrm{first}(E) = \{\}$$
$$\mathrm{first}(E') = \{\epsilon\}$$
$$\mathrm{first}(T) = \{\}$$
$$\mathrm{first}(T') = \{\epsilon\}$$
$$\mathrm{first}(F) = \{\}$$

First Set · Example (First Sets for the Arithmetic Expression Grammar)

1. $E ::= T\ E'$
2. $E' ::= \texttt{+}\ \mathrm{T}\ E'$
3. $E' ::= \epsilon$
4. $T ::= F\ T'$
5. $T' ::= \texttt{*}\ F\ T'$
6. $T' ::= \epsilon$
7. $F ::= \texttt{(}E\texttt{)}$
8. $F ::= \texttt{id}$

$$
\begin{aligned}
\mathrm{first}(E) &= \{\} \\
\mathrm{first}(E') &= \{\epsilon, \texttt{+}\} \\
\mathrm{first}(T) &= \{\} \\
\mathrm{first}(T') &= \{\epsilon, \texttt{*}\} \\
\mathrm{first}(F) &= \{\texttt{(}, \texttt{id}\}
\end{aligned}
$$

First Set · Example (First Sets for the Arithmetic Expression Grammar)

1. $E ::= T\ E'$
2. $E' ::= \texttt{+}\ \mathrm{T}\ E'$
3. $E' ::= \epsilon$
4. $T ::= F\ T'$
5. $T' ::= \texttt{*}\ F\ T'$
6. $T' ::= \epsilon$
7. $F ::= \texttt{(}E\texttt{)}$
8. $F ::= \texttt{id}$

$$
\begin{aligned}
\mathrm{first}(E) &= \{\} \\
\mathrm{first}(E') &= \{\epsilon, \texttt{+}\} \\
\mathrm{first}(T) &= \{\texttt{(}, \texttt{id}\} \\
\mathrm{first}(T') &= \{\epsilon, \texttt{*}\} \\
\mathrm{first}(F) &= \{\texttt{(}, \texttt{id}\}
\end{aligned}
$$

First Set · Example (First Sets for the Arithmetic Expression Grammar)

1. $E \ ::= T \ E'$
2. $E' ::= \texttt{+} \ \mathrm{T} \ E'$
3. $E' ::= \epsilon$
4. $T \ ::= F \ T'$
5. $T' ::= \texttt{*} \ F \ T'$
6. $T' ::= \epsilon$
7. $F \ ::= \texttt{(} E \texttt{)}$
8. $F \ ::= \texttt{id}$

$$\begin{aligned}
\mathrm{first}(E) &= \{\texttt{(}, \texttt{id}\} \\
\mathrm{first}(E') &= \{\epsilon, \texttt{+}\} \\
\mathrm{first}(T) &= \{\texttt{(}, \texttt{id}\} \\
\mathrm{first}(T') &= \{\epsilon, \texttt{*}\} \\
\mathrm{first}(F) &= \{\texttt{(}, \texttt{id}\}
\end{aligned}$$

Follow Set

Follow Set

follow(X) is the set of all terminals that can start strings derivable from what can follow X in a derivation

Follow Set

follow(X) is the set of all terminals that can start strings derivable from what can follow X in a derivation

Formally, follow(X) $= \{a | S \overset{*}{\Rightarrow} wX\alpha \text{ and } \alpha \overset{*}{\Rightarrow} a \ldots\}$

Follow Set

follow(X) is the set of all terminals that can start strings derivable from what can follow X in a derivation

Formally, follow(X) $= \{a | S \stackrel{*}{\Rightarrow} wX\alpha \text{ and } \alpha \stackrel{*}{\Rightarrow} a\ldots\}$

Alternate definition

1. follow(S) contains #, ie, the terminator follows the start symbol
2. If there is a rule $Y ::= \alpha X \beta$ in P, follow(X) contains first(β) $- \{\epsilon\}$
3. If there is a rule $Y ::= \alpha X \beta$ in P and either $\beta = \epsilon$ or first(β) contains ϵ, follow(X) contains follow(Y)

Follow Set

Follow Set

Input: a context-free grammar $G = (N, T, S, P)$
Output: follow(X) for all symbols $X \in N$

follow(S) $\leftarrow \{\#\}$
for $X \in N$ **do**
  follow(X) $\leftarrow \{\}$
end for
repeat
  for $Y ::= X_1 X_2 \ldots X_n \in P$ **do**
    for $X_i \in X_1 X_2 \ldots X_n$ **do**
      Add first($X_{i+1} X_{i+2} \ldots X_n$) $- \{\epsilon\}$ to follow(X_i)
      If X_i is the last symbol or $\epsilon \in$ first($X_{i+1} \ldots X_n$), add follow(Y) to follow(X_i)
    end for
  end for
until no new symbols are added to any set

Follow Set · Example (Follow Sets for the Arithmetic Expression Grammar)

Follow Set · Example (Follow Sets for the Arithmetic Expression Grammar)

1. $E ::= T\ E'$
2. $E' ::= \texttt{+}\ \mathrm{T}\ E'$
3. $E' ::= \epsilon$
4. $T ::= F\ T'$
5. $T' ::= \texttt{*}\ F\ T'$
6. $T' ::= \epsilon$
7. $F ::= \texttt{(}E\texttt{)}$
8. $F ::= \texttt{id}$

$$
\begin{aligned}
\mathrm{first}(E) &= \{\texttt{(},\ \texttt{id}\} \\
\mathrm{first}(E') &= \{\texttt{+},\ \epsilon\} \\
\mathrm{first}(T) &= \{\texttt{(},\ \texttt{id}\} \\
\mathrm{first}(T') &= \{\texttt{*},\ \epsilon\} \\
\mathrm{first}(F) &= \{\texttt{(},\ \texttt{id}\}
\end{aligned}
$$

Follow Set · Example (Follow Sets for the Arithmetic Expression Grammar)

1. $E ::= T \; E'$
2. $E' ::= + \; \mathrm{T} \; E'$
3. $E' ::= \epsilon$
4. $T ::= F \; T'$
5. $T' ::= * \; F \; T'$
6. $T' ::= \epsilon$
7. $F ::= (E)$
8. $F ::= \mathtt{id}$

$$
\begin{aligned}
\mathrm{first}(E) &= \{(, \mathtt{id}\} \\
\mathrm{first}(E') &= \{+, \epsilon\} \\
\mathrm{first}(T) &= \{(, \mathtt{id}\} \\
\mathrm{first}(T') &= \{*, \epsilon\} \\
\mathrm{first}(F) &= \{(, \mathtt{id}\}
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{follow}(E) &= \\
\mathrm{follow}(E') &= \\
\mathrm{follow}(T) &= \\
\mathrm{follow}(T') &= \\
\mathrm{follow}(F) &=
\end{aligned}
$$

Follow Set · Example (Follow Sets for the Arithmetic Expression Grammar)

1. $E ::= T\ E'$
2. $E' ::= +\ \mathrm{T}\ E'$
3. $E' ::= \epsilon$
4. $T ::= F\ T'$
5. $T' ::= *\ F\ T'$
6. $T' ::= \epsilon$
7. $F ::= (E)$
8. $F ::= \mathtt{id}$

$$\begin{aligned}
\text{first}(E) &= \{(, \mathtt{id}\} \\
\text{first}(E') &= \{+, \epsilon\} \\
\text{first}(T) &= \{(, \mathtt{id}\} \\
\text{first}(T') &= \{*, \epsilon\} \\
\text{first}(F) &= \{(, \mathtt{id}\}
\end{aligned}$$

$$\begin{aligned}
\text{follow}(E) &= \{\#\} \\
\text{follow}(E') &= \{\} \\
\text{follow}(T) &= \{\} \\
\text{follow}(T') &= \{\} \\
\text{follow}(F) &= \{\}
\end{aligned}$$

Follow Set · Example (Follow Sets for the Arithmetic Expression Grammar)

1. $E ::= T\ E'$
2. $E' ::= \texttt{+}\ \mathrm{T}\ E'$
3. $E' ::= \epsilon$
4. $T ::= F\ T'$
5. $T' ::= \texttt{*}\ F\ T'$
6. $T' ::= \epsilon$
7. $F ::= \texttt{(}E\texttt{)}$
8. $F ::= \texttt{id}$

$$\begin{aligned}
\mathrm{first}(E) &= \{\texttt{(}, \texttt{id}\} \\
\mathrm{first}(E') &= \{\texttt{+}, \epsilon\} \\
\mathrm{first}(T) &= \{\texttt{(}, \texttt{id}\} \\
\mathrm{first}(T') &= \{\texttt{*}, \epsilon\} \\
\mathrm{first}(F) &= \{\texttt{(}, \texttt{id}\}
\end{aligned}$$

$$\begin{aligned}
\mathrm{follow}(E) &= \{\texttt{)}, \texttt{\#}\} \\
\mathrm{follow}(E') &= \{\texttt{\#}\} \\
\mathrm{follow}(T) &= \{\texttt{+}, \texttt{\#}\} \\
\mathrm{follow}(T') &= \{\texttt{+}, \texttt{\#}\} \\
\mathrm{follow}(F) &= \{\texttt{*}, \texttt{+}, \texttt{\#}\}
\end{aligned}$$

Follow Set · Example (Follow Sets for the Arithmetic Expression Grammar)

1. $E \; ::= T \; E'$
2. $E' ::= + \; \mathrm{T} \; E'$
3. $E' ::= \epsilon$
4. $T \; ::= F \; T'$
5. $T' ::= * \; F \; T'$
6. $T' ::= \epsilon$
7. $F \; ::= (E)$
8. $F \; ::= \texttt{id}$

$$\begin{aligned}
\mathrm{first}(E) &= \{(, \texttt{id}\} \\
\mathrm{first}(E') &= \{+, \epsilon\} \\
\mathrm{first}(T) &= \{(, \texttt{id}\} \\
\mathrm{first}(T') &= \{*, \epsilon\} \\
\mathrm{first}(F) &= \{(, \texttt{id}\}
\end{aligned}$$

$$\begin{aligned}
\mathrm{follow}(E) &= \{), \#\} \\
\mathrm{follow}(E') &= \{), \#\} \\
\mathrm{follow}(T) &= \{+,), \#\} \\
\mathrm{follow}(T') &= \{+,), \#\} \\
\mathrm{follow}(F) &= \{*, +,), \#\}
\end{aligned}$$

LL(1) Parse Table

LL(1) Parse Table

Input: a context-free grammar $G = (N, T, S, P)$

Output: LL(1) parse table for G

```
for Y ∈ N do
    for Y ::= X1 X2 ... Xn ∈ P with index i do
        for a ∈ first(X1 X2 ... Xn) − {ϵ} do
            table[Y, a] ← i
            if ϵ ∈ first(X1 X2 ... Xn) then
                for a ∈ follow(Y) do
                    table[Y, a] ← i
                end for
            end if
        end for
    end for
end for
```

LL(1) Parse Table · Example (Parser Table for the Arithmetic Expression Grammar)

LL(1) Parse Table · Example (Parser Table for the Arithmetic Expression Grammar)

1. $E ::= T\ E'$
2. $E' ::= \texttt{+}\ \mathrm{T}\ E'$
3. $E' ::= \epsilon$
4. $T ::= F\ T'$
5. $T' ::= \texttt{*}\ F\ T'$
6. $T' ::= \epsilon$
7. $F ::= \texttt{(}E\texttt{)}$
8. $F ::= \texttt{id}$

$$\begin{aligned}
\mathrm{first}(E) &= \{\texttt{(},\ \texttt{id}\} & \mathrm{follow}(E) &= \{\texttt{)},\ \texttt{\#}\} \\
\mathrm{first}(E') &= \{\epsilon,\ \texttt{+}\} & \mathrm{follow}(E') &= \{\texttt{)},\ \texttt{\#}\} \\
\mathrm{first}(T) &= \{\texttt{(},\ \texttt{id}\} & \mathrm{follow}(T) &= \{\texttt{+},\ \texttt{)},\ \texttt{\#}\} \\
\mathrm{first}(T') &= \{\epsilon,\ \texttt{*}\} & \mathrm{follow}(T') &= \{\texttt{+},\ \texttt{)},\ \texttt{\#}\} \\
\mathrm{first}(F) &= \{\texttt{(},\ \texttt{id}\} & \mathrm{follow}(F) &= \{\texttt{*},\ \texttt{+},\ \texttt{)},\ \texttt{\#}\}
\end{aligned}$$

LL(1) Parse Table · Example (Parser Table for the Arithmetic Expression Grammar)

1. $E \ ::= T \ E'$
2. $E' ::= + \ \mathrm{T} \ E'$
3. $E' ::= \epsilon$
4. $T \ ::= F \ T'$
5. $T' ::= * \ F \ T'$
6. $T' ::= \epsilon$
7. $F \ ::= (E)$
8. $F \ ::= \mathtt{id}$

$$\begin{aligned}
\text{first}(E) &= \{(, \mathtt{id}\} & \text{follow}(E) &= \{), \#\} \\
\text{first}(E') &= \{\epsilon, +\} & \text{follow}(E') &= \{), \#\} \\
\text{first}(T) &= \{(, \mathtt{id}\} & \text{follow}(T) &= \{+,), \#\} \\
\text{first}(T') &= \{\epsilon, *\} & \text{follow}(T') &= \{+,), \#\} \\
\text{first}(F) &= \{(, \mathtt{id}\} & \text{follow}(F) &= \{*, +,), \#\}
\end{aligned}$$

	+	*	(	)	id	#
E						
E'						
T						
T'						
F						

LL(1) Parse Table · Example (Parser Table for the Arithmetic Expression Grammar)

1. $E \ ::= T \ E'$
2. $E' ::= + \ \mathrm{T} \ E'$
3. $E' ::= \epsilon$
4. $T \ ::= F \ T'$
5. $T' ::= * \ F \ T'$
6. $T' ::= \epsilon$
7. $F \ ::= (E)$
8. $F \ ::= \mathtt{id}$

$$\begin{aligned}
\text{first}(E) &= \{(, \mathtt{id}\} & \text{follow}(E) &= \{), \#\} \\
\text{first}(E') &= \{\epsilon, +\} & \text{follow}(E') &= \{), \#\} \\
\text{first}(T) &= \{(, \mathtt{id}\} & \text{follow}(T) &= \{+,), \#\} \\
\text{first}(T') &= \{\epsilon, *\} & \text{follow}(T') &= \{+,), \#\} \\
\text{first}(F) &= \{(, \mathtt{id}\} & \text{follow}(F) &= \{*, +,), \#\}
\end{aligned}$$

	+	*	(	)	id	#
E			1		1	
E'						
T						
T'						
F						

LL(1) Parse Table · Example (Parser Table for the Arithmetic Expression Grammar)

1. $E \ ::= \ T \ E'$
2. $E' \ ::= \ \texttt{+} \ \mathrm{T} \ E'$
3. $E' \ ::= \ \epsilon$
4. $T \ ::= \ F \ T'$
5. $T' \ ::= \ \texttt{*} \ F \ T'$
6. $T' \ ::= \ \epsilon$
7. $F \ ::= \ \texttt{(}E\texttt{)}$
8. $F \ ::= \ \texttt{id}$

$$\begin{aligned}
\mathrm{first}(E) &= \{\texttt{(}, \texttt{id}\} & \mathrm{follow}(E) &= \{\texttt{)}, \texttt{\#}\} \\
\mathrm{first}(E') &= \{\epsilon, \texttt{+}\} & \mathrm{follow}(E') &= \{\texttt{)}, \texttt{\#}\} \\
\mathrm{first}(T) &= \{\texttt{(}, \texttt{id}\} & \mathrm{follow}(T) &= \{\texttt{+}, \texttt{)}, \texttt{\#}\} \\
\mathrm{first}(T') &= \{\epsilon, \texttt{*}\} & \mathrm{follow}(T') &= \{\texttt{+}, \texttt{)}, \texttt{\#}\} \\
\mathrm{first}(F) &= \{\texttt{(}, \texttt{id}\} & \mathrm{follow}(F) &= \{\texttt{*}, \texttt{+}, \texttt{)}, \texttt{\#}\}
\end{aligned}$$

	+	*	(	)	id	#
E			1		1	
E'	2					
T						
T'						
F						

LL(1) Parse Table · Example (Parser Table for the Arithmetic Expression Grammar)

1. $E ::= T\ E'$
2. $E' ::= +\ \mathrm{T}\ E'$
3. $E' ::= \epsilon$
4. $T ::= F\ T'$
5. $T' ::= *\ F\ T'$
6. $T' ::= \epsilon$
7. $F ::= (E)$
8. $F ::= \mathtt{id}$

$$\begin{aligned} \mathrm{first}(E) &= \{(, \mathtt{id}\} \\ \mathrm{first}(E') &= \{\epsilon, +\} \\ \mathrm{first}(T) &= \{(, \mathtt{id}\} \\ \mathrm{first}(T') &= \{\epsilon, *\} \\ \mathrm{first}(F) &= \{(, \mathtt{id}\} \end{aligned}$$

$$\begin{aligned} \mathrm{follow}(E) &= \{), \#\} \\ \mathrm{follow}(E') &= \{), \#\} \\ \mathrm{follow}(T) &= \{+,), \#\} \\ \mathrm{follow}(T') &= \{+,), \#\} \\ \mathrm{follow}(F) &= \{*, +,), \#\} \end{aligned}$$

	+	*	(	)	id	#
E			1		1	
E'	2			3		3
T						
T'						
F						

LL(1) Parse Table · Example (Parser Table for the Arithmetic Expression Grammar)

1. $E ::= T\ E'$
2. $E' ::= \texttt{+}\ T\ E'$
3. $E' ::= \epsilon$
4. $T ::= F\ T'$
5. $T' ::= \texttt{*}\ F\ T'$
6. $T' ::= \epsilon$
7. $F ::= \texttt{(}E\texttt{)}$
8. $F ::= \texttt{id}$

$$\begin{aligned}
\text{first}(E) &= \{\texttt{(}, \texttt{id}\} & \text{follow}(E) &= \{\texttt{)}, \texttt{\#}\} \\
\text{first}(E') &= \{\epsilon, \texttt{+}\} & \text{follow}(E') &= \{\texttt{)}, \texttt{\#}\} \\
\text{first}(T) &= \{\texttt{(}, \texttt{id}\} & \text{follow}(T) &= \{\texttt{+}, \texttt{)}, \texttt{\#}\} \\
\text{first}(T') &= \{\epsilon, \texttt{*}\} & \text{follow}(T') &= \{\texttt{+}, \texttt{)}, \texttt{\#}\} \\
\text{first}(F) &= \{\texttt{(}, \texttt{id}\} & \text{follow}(F) &= \{\texttt{*}, \texttt{+}, \texttt{)}, \texttt{\#}\}
\end{aligned}$$

	+	*	(	)	id	#
E			1		1	
E'	2			3		3
T			4		4	
T'						
F						

LL(1) Parse Table · Example (Parser Table for the Arithmetic Expression Grammar)

1. $E ::= T\ E'$
2. $E' ::= +\ \mathrm{T}\ E'$
3. $E' ::= \epsilon$
4. $T ::= F\ T'$
5. $T' ::= *\ F\ T'$
6. $T' ::= \epsilon$
7. $F ::= (E)$
8. $F ::= \mathtt{id}$

$$\begin{aligned}
\text{first}(E) &= \{(,\ \mathtt{id}\} & \text{follow}(E) &= \{),\ \#\} \\
\text{first}(E') &= \{\epsilon,\ +\} & \text{follow}(E') &= \{),\ \#\} \\
\text{first}(T) &= \{(,\ \mathtt{id}\} & \text{follow}(T) &= \{+,\ ),\ \#\} \\
\text{first}(T') &= \{\epsilon,\ *\} & \text{follow}(T') &= \{+,\ ),\ \#\} \\
\text{first}(F) &= \{(,\ \mathtt{id}\} & \text{follow}(F) &= \{*,\ +,\ ),\ \#\}
\end{aligned}$$

	+	*	(	)	id	#
E			1		1	
E'	2			3		3
T			4		4	
T'		5				
F						

LL(1) Parse Table · Example (Parser Table for the Arithmetic Expression Grammar)

1. $E \ ::= T \ E'$
2. $E' ::= \texttt{+} \ \mathrm{T} \ E'$
3. $E' ::= \epsilon$
4. $T \ ::= F \ T'$
5. $T' ::= \texttt{*} \ F \ T'$
6. $T' ::= \epsilon$
7. $F \ ::= \texttt{(} E \texttt{)}$
8. $F \ ::= \texttt{id}$

$$\mathrm{first}(E) = \{\texttt{(}, \texttt{id}\} \qquad \mathrm{follow}(E) = \{\texttt{)}, \texttt{\#}\}$$
$$\mathrm{first}(E') = \{\epsilon, \texttt{+}\} \qquad \mathrm{follow}(E') = \{\texttt{)}, \texttt{\#}\}$$
$$\mathrm{first}(T) = \{\texttt{(}, \texttt{id}\} \qquad \mathrm{follow}(T) = \{\texttt{+}, \texttt{)}, \texttt{\#}\}$$
$$\mathrm{first}(T') = \{\epsilon, \texttt{*}\} \qquad \mathrm{follow}(T') = \{\texttt{+}, \texttt{)}, \texttt{\#}\}$$
$$\mathrm{first}(F) = \{\texttt{(}, \texttt{id}\} \qquad \mathrm{follow}(F) = \{\texttt{*}, \texttt{+}, \texttt{)}, \texttt{\#}\}$$

	+	*	(	)	id	#
E			1		1	
E'	2			3		3
T			4		4	
T'	6	5		6		6
F						

LL(1) Parse Table · Example (Parser Table for the Arithmetic Expression Grammar)

1. $E \ ::= \ T \ E'$
2. $E' \ ::= \ \texttt{+} \ \mathrm{T} \ E'$
3. $E' \ ::= \ \epsilon$
4. $T \ ::= \ F \ T'$
5. $T' \ ::= \ \texttt{*} \ F \ T'$
6. $T' \ ::= \ \epsilon$
7. $F \ ::= \ \texttt{(}E\texttt{)}$
8. $F \ ::= \ \texttt{id}$

$$\begin{aligned} \mathrm{first}(E) &= \{\texttt{(}, \texttt{id}\} & \mathrm{follow}(E) &= \{\texttt{)}, \texttt{\#}\} \\ \mathrm{first}(E') &= \{\epsilon, \texttt{+}\} & \mathrm{follow}(E') &= \{\texttt{)}, \texttt{\#}\} \\ \mathrm{first}(T) &= \{\texttt{(}, \texttt{id}\} & \mathrm{follow}(T) &= \{\texttt{+}, \texttt{)}, \texttt{\#}\} \\ \mathrm{first}(T') &= \{\epsilon, \texttt{*}\} & \mathrm{follow}(T') &= \{\texttt{+}, \texttt{)}, \texttt{\#}\} \\ \mathrm{first}(F) &= \{\texttt{(}, \texttt{id}\} & \mathrm{follow}(F) &= \{\texttt{*}, \texttt{+}, \texttt{)}, \texttt{\#}\} \end{aligned}$$

	+	*	(	)	id	#
E			1		1	
E'	2			3		3
T			4		4	
T'	6	5		6		6
F			7		8	

LL(1) Parse Table

LL(1) Parse Table

We say a grammar is LL(1) if the parse table has no conflicts, ie, no entries with more than one rule

LL(1) Parse Table

We say a grammar is LL(1) if the parse table has no conflicts, ie, no entries with more than one rule

If a grammar is LL(1), then it is unambiguous

LL(1) Parse Table

We say a grammar is LL(1) if the parse table has no conflicts, ie, no entries with more than one rule

If a grammar is LL(1), then it is unambiguous

It is possible for a grammar not to be LL(1) but LL(k) for some $k > 1$, which would mean a parse table having columns for each combination of k symbols

LL(1) Parse Table

We say a grammar is LL(1) if the parse table has no conflicts, ie, no entries with more than one rule

If a grammar is LL(1), then it is unambiguous

It is possible for a grammar not to be LL(1) but LL(k) for some $k > 1$, which would mean a parse table having columns for each combination of k symbols

Not all context-free grammars are LL(1), but for many that are not, we can define equivalent grammars that are LL(1)

Removing Left Recursion

Removing Left Recursion

One type of grammar that is not LL(1) is a grammar having a rule with direct left recursion

$$\text{Y} ::= \text{Y}\ \alpha$$
$$\text{Y} ::= \beta$$

Removing Left Recursion

One type of grammar that is not LL(1) is a grammar having a rule with direct left recursion

$$\mathrm{Y} ::= \mathrm{Y}\ \alpha$$
$$\mathrm{Y} ::= \beta$$

Equivalent grammar without the direct left recursion

$$\mathrm{Y} ::= \beta\ Y'$$
$$Y' ::= \alpha\ Y'$$
$$Y' ::= \epsilon$$

Removing Left Recursion · Example

$$E ::= E \texttt{ + } T$$
$$E ::= T$$
$$T ::= T \texttt{ * } \mathrm{F}$$
$$T ::= F$$
$$F ::= \texttt{(}E\texttt{)}$$
$$F ::= \texttt{id}$$

$$E ::= E \texttt{+} T$$
$$E ::= T$$
$$T ::= T \texttt{*} \mathrm{F}$$
$$T ::= F$$
$$F ::= \texttt{(}E\texttt{)}$$
$$F ::= \texttt{id}$$

Equivalent LL(1) grammar

$$E ::= T\ E'$$
$$E' ::= \texttt{+}\ T\ E'$$
$$E' ::= \epsilon$$
$$T ::= F\ T'$$
$$T' ::= \texttt{*}\ F\ T'$$
$$T' ::= \epsilon$$
$$F ::= \texttt{(}E\texttt{)}$$
$$F ::= \texttt{id}$$

Removing Left Recursion

Removing Left Recursion

Input: a context-free grammar $G = (N, T, S, P)$
Output: G with left recursion eliminated

Arbitrarily enumerate the non-terminals of G
for $i := 1$ to n **do**
  for $j := 1$ to $i - 1$ **do**
    Replace pairs of rules of the form $X_i ::= X_j \alpha$ and $X_j ::= \beta_1 | \beta_2 | \ldots | \beta_k$ by the rules $X_i ::= \beta_1 \alpha | \beta_2 \alpha | \ldots | \beta_k \alpha$
    Eliminate any direct left recursion
  end for
end for