

Discussion4 Answer

1. **LL Parsing Ambiguities.** An LL(k) grammar is a CFG used by a parser that scans input left-to-right (“L”), leftmost derivation (“L”), and uses k tokens of lookahead to predict the correct production. We’ve previously seen that a grammar is ambiguous if it has a parse tree that is not unique. A more formal definition of LL conflicts uses FIRST and FOLLOW sets.

- **FIRST(A):** the set of all terminals that could occur first in an expansion of the terminal or nonterminal A (include ϵ if A can expand to ϵ)
- **FOLLOW(A):** the set of all terminals that could follow an occurrence of the terminal or nonterminal A in a (partial) derivation

There are two main types of LL(1) conflicts:

- **FIRST/FIRST:** The FIRST sets of two different productions for same non-terminal intersect.
- **FIRST/FOLLOW:** The FIRST set of a grammar rule contains an epsilon and the intersection with its FOLLOW set is not empty.

Is the following grammar LL(1)? Justify your answer using FIRST and FOLLOW sets.

- $S \rightarrow Xd \quad X \rightarrow C \mid Ba \quad C \rightarrow \epsilon \quad B \rightarrow d$

Answer:

This is an instance of a FIRST/FOLLOW conflict. FIRST(X) contains the empty string and the intersection of FIRST(X) and FOLLOW(X) is not empty:

$$\text{FIRST}(S) = \text{FIRST}(Xd) = \{ 'd' \} \quad \text{FIRST}(X) = \text{FIRST}(C) \cup \text{FIRST}(Ba) \quad \text{FIRST}(C) = \{ \epsilon \}$$

$$\text{FIRST}(Ba) = \{ 'd' \} \quad \text{FIRST}(B) = \{ 'd' \}$$

$$\text{FOLLOW}(S) = \{ \} \quad \text{FOLLOW}(X) = \{ 'd' \} \quad \text{FOLLOW}(C) = \{ 'd' \} \quad \text{FOLLOW}(B) = \{ 'a' \}$$

2. **Resolving Conflicts.**

Consider the following grammar for numerical expressions with division, addition, and unary minus:

- $E \rightarrow Num \mid E/E \mid E + E \mid - E$

- (a) Rewrite the grammar so that it is LL(1), so that $'/'$ has higher precedence than $'+'$, and so that $'-'$ has highest precedence. $'+'$ and $'/'$ should be parsed in a right-associative way.

Answer:

$$\begin{array}{lll} \text{expr} : \text{expr1 rest} & \text{expr1} : \text{expr2 rest1} & \text{expr2} : \text{'-'} \text{expr2} \\ & & \mid \text{NUM} \\ \text{rest} : \epsilon & \text{rest1} : \epsilon & \\ & \mid \text{'+' expr} & \mid \text{'/' expr1} \end{array}$$

- (b) Compute the FIRST and FOLLOW sets for your re-written LL(1) grammar.

Answer:

$$\text{FIRST}(\text{expr1 rest}) = \text{FIRST}(\text{expr2 rest1}) = \{ \text{'-'}, \text{NUM} \}$$

$$\begin{array}{ll} \text{FIRST}(\text{'+' expr}) = \{ \text{'+'} \} & \text{FIRST}(\text{'/' expr1}) = \{ \text{'/'} \} \\ \text{FIRST}(\text{'-' expr2}) = \{ \text{'-'}, \epsilon \} & \text{FIRST}(\text{NUM}) = \{ \text{NUM} \} \\ \text{FIRST}(\epsilon) = \{ \epsilon \} & \end{array}$$

$$\begin{array}{l} \text{FOLLOW}(\text{expr2}) = \{ \text{'/'}, \text{'+'}, \epsilon \} \\ \text{FOLLOW}(\text{expr1}) = \text{FOLLOW}(\text{rest1}) = \{ \text{'+'}, \epsilon \} \\ \text{FOLLOW}(\text{expr}) = \text{FOLLOW}(\text{rest}) = \{ \epsilon \} \end{array}$$

- (c) Draw the LL(1) parsing table for the grammar. You may need the following rules:

- For each production $X \rightarrow A_1 \dots A_n$:
 - For each $1 \leq i \leq n$, and for each b in **First**(A_i): Set $T[X, b] = X \rightarrow A_1 \dots A_n$. Stop when ϵ is not in **First**(A_i).
 - If $A_1 \dots A_n \rightarrow^* \epsilon$, then for each b in **Follow**(X): Set $T[X, b] = \epsilon$.

Answer:

	$-$	NUM	$/$	$+$	ϵ
expr	$\text{expr} \rightarrow \text{expr1 rest}$	$\text{expr} \rightarrow \text{expr1 rest}$			
rest				$\text{rest} \rightarrow +\text{expr}$	ϵ
expr1	$\text{expr1} \rightarrow \text{expr2 rest1}$	$\text{expr1} \rightarrow \text{expr2 rest1}$			
rest1			$\text{rest1} \rightarrow / \text{expr1}$	ϵ	ϵ
expr2	$\text{expr2} \rightarrow - \text{expr2}$	$\text{expr2} \rightarrow \text{NUM}$			

Here is an example trace, using the above table for parsing the string “6+4/4 -”.

```

expr = 6 + 4/4 -
expr1 rest = 6 + 4/4 -
expr2 rest1 rest = 6 + 4/4 -
NUM rest1 rest = 6 + 4/4 -
rest1 rest = +4/4 -
rest = +4/4 -
+ expr = +4/4 -
expr = 4/4 -
expr1 rest = 4/4 -
expr2 rest1 rest = 4/4 -
NUM rest1 rest = 4/4 -
rest1 rest = /4 -
/ expr1 rest = /4 -
expr1 rest = 4 -
expr2 rest1 rest = 4 -
NUM rest1 rest = 4 -
rest1 rest = -
rest = -
EMPTY = -

```

3. Earley's Algorithm.

- Consider the following grammar:

$$\begin{aligned}
 P &\rightarrow E - \\
 E &\rightarrow E + E \\
 E &\rightarrow E * E \\
 E &\rightarrow ID
 \end{aligned}$$

Use Earley's algorithm to parse $ID + ID * ID$ by filling out the chart below (you may not need all of the rows).

		ID	+	ID	*	ID
	0	1	2	3	4	5
a						
b						
c						
d						
e						
f						
g						
h						
i						
j						
k						
l						
m						

Answer: A solution has been posted in the **extra** directory.

- Draw the accepting parse tree(s) and identify whether there are ambiguities in the grammar.

Answer: A solution has been posted in the **extra** directory.