

CSE 401 - LL Parsing Worksheet Sample Solutions - Week 4

1. Compute nullable and the FIRST and FOLLOW sets for each non-terminal in the following grammar:

$A ::= x C B y$

$B ::= z \mid \epsilon$

$C ::= y \mid B x$

Solution:

Non-Terminal	FIRST	FOLLOW	nullable
A	x		no
B	z	x, y	yes
C	x, y, z	y, z	no

Although for this problem it was not expected to show your work at this level of detail, the steps taken to compute these sets are reproduced below to illustrate the algorithm.

Note that for some applications of the FIRST/FOLLOW sets, a special “pseudo” production is used for practical reasons to represent a single instance of the start symbol followed by the special terminal “\$” signifying the end of the input. If a grammar’s FIRST/FOLLOW sets are considered in the context of this special “pseudo” production, such as when constructing an LR(0) state machine, then the special terminal “\$” would be part of the FOLLOW sets for any non-terminal that can appear at the end of the input. Since this problem only asks for the FIRST/FOLLOW sets of the given grammar and there is no LR(0) context requiring the “pseudo” production, there is no need to include “\$”.

Step	Current Production	Updates
0.	--	Initially, for each non-terminal the FIRST and FOLLOW sets are empty and nullable is “no”. For each terminal, the FIRST set simply contains that terminal.
1.	$A ::= x C B y$	We start by examining the first production, $A ::= x C B y$. Since x is the first symbol, we add FIRST[x] to FIRST[A], meaning $\text{FIRST}[A] = \{ x \}$. We also add FIRST[y]

to FOLLOW[B], so FOLLOW[B] = { y }. However, we do not perform any additional updates yet -- although C is directly followed by B, there is no change to FOLLOW[C] because FIRST[B] is currently empty. Likewise, although looking ahead reveals that some non-terminals will be nullable, the algorithm will not use that information until after the relevant production has been visited.

2.	$B ::= z$	We add FIRST[z] to FIRST[B], so FIRST[B] = { z }.
3.	$B ::= \epsilon$	Since everything in the right side of the production is nullable, we change nullable[B] to "yes".
4.	$C ::= y$	We add FIRST[y] to FIRST[C], so FIRST[C] = { y }.
5.	$C ::= B x$	Since B is the first thing in the right side of the production, we add FIRST[B] to FIRST[C]. However, since B is nullable, at this point we have enough information to also add FIRST[x] to FIRST[C], meaning FIRST[C] ends up being { x, y, z }. In addition, we add FIRST[x] to FOLLOW[B], so FOLLOW[B] becomes { x, y }.
6.	$A ::= x C B y$	When we examine this production for the second time, we are able to make more updates as a result of the new information in the table. Since C is directly followed by B, we copy FIRST[B] into FOLLOW[C]. But because B is nullable, we also copy FIRST[y] into FOLLOW[C], so FOLLOW[C] becomes { y, z }.
7.	$B ::= z$	No change.
8.	$B ::= \epsilon$	No change.
9.	$C ::= y$	No change.
10.	$C ::= B x$	No change.
11.	$A ::= x C B y$	No change. Since we have now visited every production without changing anything, the algorithm can terminate.

2. For each of the following grammars, identify whether or not the grammar satisfies the LL(1) condition. If the grammar is not LL(1), explain the problem. *Hint:* Although you are not required to follow the formal algorithm, you may find it helpful to examine the grammar in terms of the FIRST, FOLLOW, and nullable sets.

- a) $X ::= a Y \mid Z$
 $Y ::= a \mid c$
 $Z ::= b Y$

Solution:

This grammar is LL(1).

This grammar satisfies the LL(1) condition because for every non-terminal, all productions have disjoint FIRST sets (and no non-terminals are nullable):

Production	FIRST Set for Production
$X ::= a Y$	$\text{FIRST}[X] = \{ a \}$
$X ::= Z$	$\text{FIRST}[X] = \{ b \}$
$Y ::= a$	$\text{FIRST}[Y] = \{ a \}$
$Y ::= c$	$\text{FIRST}[Y] = \{ c \}$
$Z ::= b Y$	$\text{FIRST}[Z] = \{ b \}$

b) $P ::= d R$
 $R ::= o \mid S$
 $S ::= g \mid o g$

Solution:

This grammar is not LL(1).

This grammar does not satisfy the LL(1) condition because there are two productions for the non-terminal R that both contain o in their FIRST sets.

Production	FIRST Set for Production
$P ::= d R$	$\text{FIRST}[P] = \{ d \}$
$R ::= o$	$\text{FIRST}[R] = \{ o \}$
$R ::= S$	$\text{FIRST}[R] = \{ o, g \}$
$S ::= g$	$\text{FIRST}[Y] = \{ g \}$
$S ::= o g$	$\text{FIRST}[Z] = \{ o \}$

c) $J ::= a K L$
 $K ::= c \mid \epsilon$
 $L ::= c$

Solution:

This grammar is not LL(1).

Although the productions for every non-terminal have disjoint FIRST sets, this grammar still doesn't satisfy the LL(1) condition. Since K is nullable, we must consider the FOLLOW set as well when checking for conflicts -- and because c is in the FIRST set for one production of K and the FOLLOW set for another, nullable production, this grammar isn't LL(1).

Production	FIRST Set for Production	FOLLOW Set for Production
$J ::= a K L$	$FIRST[J] = \{ a \}$	$FOLLOW[J] = \emptyset$
$K ::= c$	$FIRST[K] = \{ c \}$	$FOLLOW[K] = \{ c \}$
$K ::= \epsilon$	$FIRST[K] = \emptyset$	$FOLLOW[K] = \{ c \}$
$L ::= c$	$FIRST[L] = \{ c \}$	$FOLLOW[L] = \emptyset$

d) $J ::= a K L$
 $K ::= c \mid \epsilon$
 $L ::= b$

Solution:

This grammar is LL(1).

By altering the previous grammar to change a single terminal from c to b, the grammar becomes LL(1). Although the fact that the production $K ::= \epsilon$ is nullable means we still consider the FOLLOW set, the FIRST set for $K ::= c$ is disjoint from the FOLLOW set for $K ::= \epsilon$, and thus there is no violation of the LL(1) property.

Production	FIRST Set for Production	FOLLOW Set for Production
$J ::= a K L$	$FIRST[J] = \{ a \}$	$FOLLOW[J] = \emptyset$
$K ::= c$	$FIRST[K] = \{ c \}$	$FOLLOW[K] = \{ b \}$
$K ::= \epsilon$	$FIRST[K] = \emptyset$	$FOLLOW[K] = \{ b \}$
$L ::= b$	$FIRST[L] = \{ b \}$	$FOLLOW[L] = \emptyset$