Complier Design Assignment - Unit 2

Submitted By:
Habibur Rahman
17BCS071

Submitted To:
Mr. Sarfaraz Masood

B-Tech VIth Semester (Computer Science)

Department of Computer Engineering,
Faculty of Engineering and Technology,
Jamia Millia Islamia ,
New Delhi

Session: 2019-20

Q1. With suitable reasons, justify whether the following grammar is LL(1), SLR(1), CLR(1) or LALR(1)?
S → F | H
F → p | c
H → d | c
Soln.)
there are two possible leftmost Derivation for string “c”.

First Leftmost Derivation

S → F

F → c

Second Leftmost Derivation

S → H

H → c

Here we have two left most derivations possible for the given grammar

Then its is a Ambiguous grammar.

A parser works on the basis of given grammar. It takes the grammar as it is. Parser does not work on the basis of the yield of the grammar. Also, while constructing the LL(1) parser table, that entry for terminal 'c' will contain multiple entries. So, LL(1) parser cannot be constructed for the given grammar.

For Ambiguous grammar none of them are possible So , we can’t find LL(1) , SLR(1) , CLR(1) , or LALR(1) .

Q2. Construct FIRST and FOLLOW sets for the following grammar.
S → Abb | C
A → aA | b
C → ab | cde
Indicate whether or not the grammar is LL(1), and explain why or why not.

If we create table of follow and first then we will find first
S → Abb | C
A → aA | b
C → ab | cde
First(S) → { First(A) , First(C) }
= { a , b , c }
First(A) → { a , b }
First(C) → { a , c }

For finding follow of any variable, we have some rules,

1. The follow of the start symbol has $ in the set
2. For a production of the type,
   A→αBβ
   First of (B) is follow (B)
3. For a production,
   (A → αB) or (A → αBβ and B → ∈)
   Follow(A) is in follow(B)

Follow (S) = { $ }
Follow(A) = { b }
Follow(C) = { Follow of (S)
{ $ }

For LL(1) grammar, grammar should not have 2 or more derivations means grammar should not be ambiguous.
Checking for left recursion,
A direct left recursion is of type A → Aα
We have grammar S → Abb | C
A → aA | b
C → ab | cde
Here we can see that there is no grammar like A → Aα so here is not present.
Now, we will check for indirect left recursion if starting from any symbol of the grammar it is possible to derive a string whose head so that symbol then left recursion is present.
S → Ab
A → aA
Derives ‘a’
S → C
C → ab
Derives ‘a’

Both can derive the symbol ‘a’ by using both of the production. So according to above rule left recursion is present.
Grammar is not LL(1)

Q3. Consider the following CFG : S → aSa | bS | c .
Now, using relevant reasons, comment on each of the following options as being True or False
(A) LL(1) but not LR(1) (B) LR(1) but not LL(1)
(C) Both LL(1) and LR(1) (D) Neither LL(1) nor LR(1)
Soln.)

The LL(1) predictive parsing table is as follows:
$S' \rightarrow S$
$S \rightarrow aSa \mid bS \mid c$

	a	b	c	$
S'	-	-	-	$S' \rightarrow S$
R	$S \rightarrow aSa$	$S \rightarrow bS$	$S \rightarrow c$	-

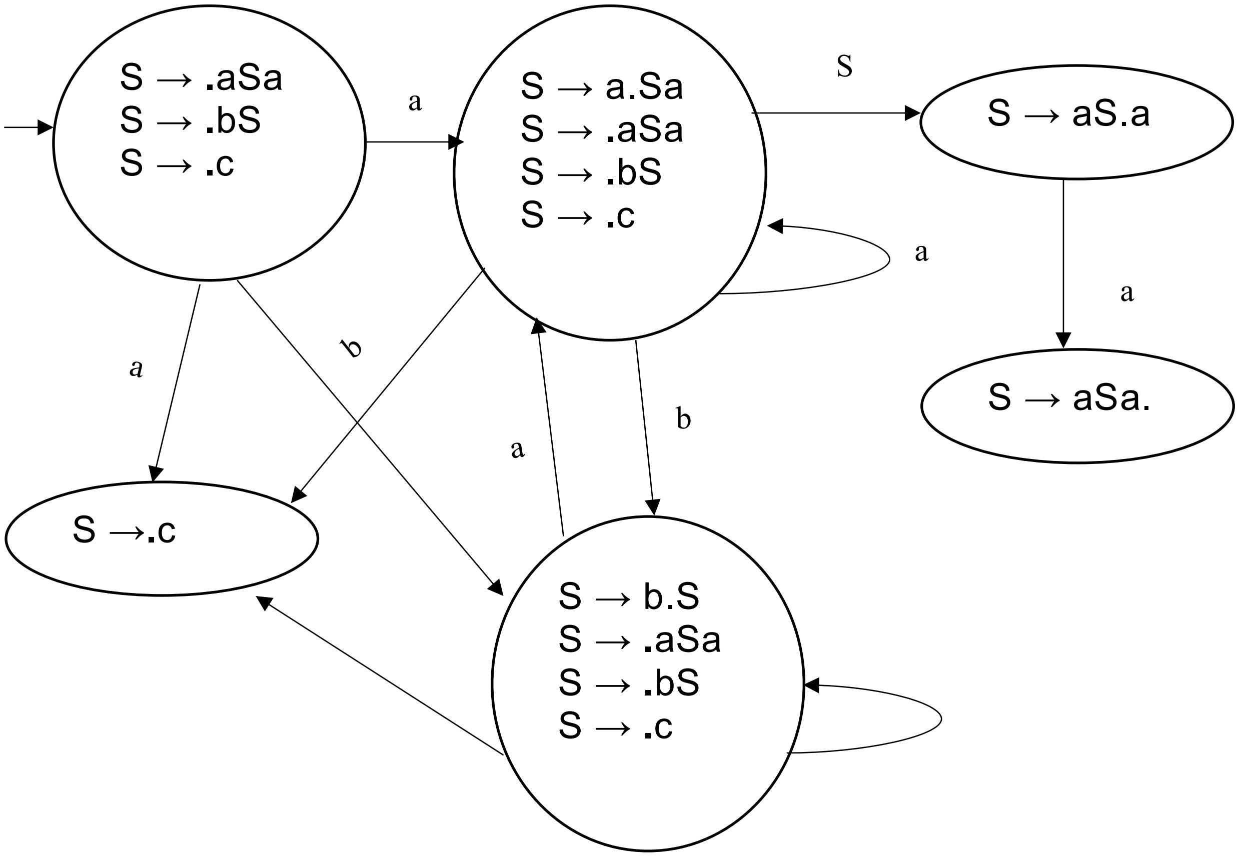
All are mutually disjoint i.e no common terminal
As there is no conflict **the grammer is LL(1) .**

As the grammar is LL(1) so it will also be LR(1) as LR parsers are more powerful then LL(1) parsers. and all LL(1) grammar are also LR(1)
So option C is correct.

(A) LL(1) but not LR(1)	**False**
(B) LR(1) but not LL(1)	**False**
(C) Both LL(1) and LR(1)	**True**
(D) Neither LL(1) nor LR(1)	**False**

The LR(1) transition diagram : -

$S \rightarrow aSa \mid bS \mid c$


S → .aSa
S → .bS
S → .c
a
S → a.Sa
S → .aSa
S → .bS
S → .c
S
S → aS.a
a
a
S → aSa.
a
b
a
b
S →.c
S → b.S
S → .aSa
S → .bS
S → .c