

1. a. (5) Explain why the following grammar is not LL(k) for any k:

$$\begin{aligned} S &\rightarrow A \mid B \\ A &\rightarrow aaA \mid aa \\ B &\rightarrow aaB \mid a \end{aligned}$$

Both nonterminals A and B begin with the string 'aa' (and the symbol a), hence we cannot distinguish between them on a leftmost parse. On a rightmost parse we must reduce a final "a" into B, looking ahead one symbol into the parse. Thus the grammar is LR(1).

b. (5) Explain why the following grammar is LL(1) but not SLR(1):

$$\begin{aligned} S &\rightarrow AaAb \mid BbBa \\ A &\rightarrow \epsilon \\ B &\rightarrow \epsilon \end{aligned}$$

To show that it is not SLR(1) but LL(1), consider the LR(0) items in the initial set for the augmented grammar:

$$\begin{aligned} S' &\rightarrow S \\ S &\rightarrow AaAb \mid BbBa \\ A &\rightarrow \epsilon \\ B &\rightarrow \epsilon \\ I_0 : S' &\rightarrow \cdot S \\ S &\rightarrow \cdot AaAb \\ S &\rightarrow \cdot BbBa \\ A &\rightarrow \cdot \\ B &\rightarrow \cdot \end{aligned}$$

And consider that FOLLOW(A) = FOLLOW(B) = {a,b}.

In the first state we have a reduce/reduce conflict, since the FOLLOW of A and B are same on the input terminal 'a' of the token we don't know whether to retrace the rule A → ε or B → ε. Hence in the LR parsing table we have two conflicting entries for action[0,a]. Same holds for action[0,b].

Now consider the LL(1) parser for the above grammar.

FIRST(S) = {a,b}

FIRST(A) = FIRST(B) = {ε}

FOLLOW(A) = FOLLOW(B) = {a,b}

FOLLOW(S) = {\$}

The parse table is

|   | a        | b        | \$ |
|---|----------|----------|----|
| S | S → AaAb | S → BbBa |    |
| A | A → ε    | A → ε    |    |
| B | B → ε    | B → ε    |    |

No conflicts, so the grammar is LL(1).

- c. (5) Which of LL(1), SLR(1), and LR(1) can parse strings in the following grammar, and why?

$$\begin{aligned} E &\square A \mid B \\ A &\square a \mid c \\ B &\square b \mid c \end{aligned}$$

The grammar is ambiguous; there are two possible derivations of the string "c". None of LL(1), SLR(1), LR(1) is able to parse any ambiguous grammar. While it is possible to work through complete LL(1), SLR(1), and LR(1) constructions to show exactly where they fail, that is not necessary. As soon as you show that a grammar is ambiguous you immediately know that none of LL(1), SLR(1), or LR(1) can possibly work.

2. (10) Consider the following two BNF grammars. In both cases:

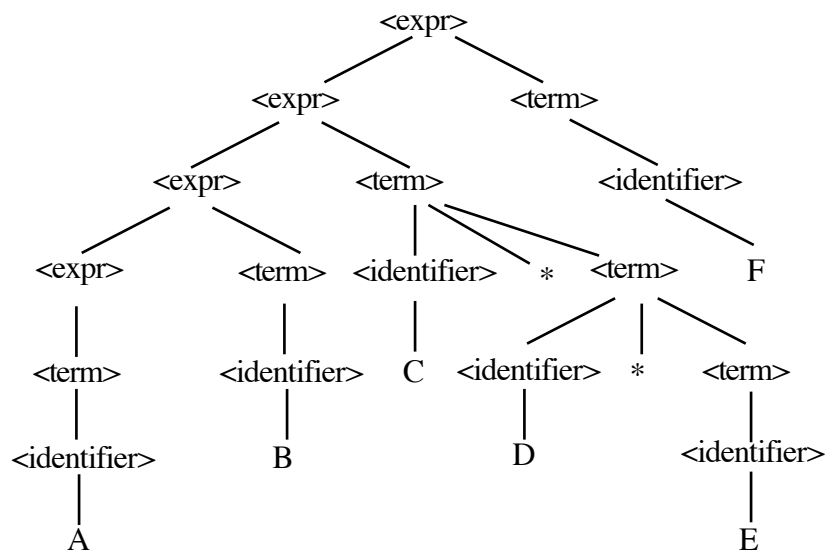
- the terminals are { A, B, C, D, E, F, \* },
- the non-terminals are { <expr>, <term>, <identifier> }, and
- the start symbol is <expr>.

For each grammar draw a parse tree for the expression

**A B C \* D \* E F**

If the grammar is ambiguous, state that it is ambiguous and give the reason for the ambiguity.

Grammar A:

$$\begin{aligned} \langle \text{expr} \rangle &::= \langle \text{term} \rangle \mid \langle \text{expr} \rangle \langle \text{term} \rangle \\ \langle \text{term} \rangle &::= \langle \text{identifier} \rangle \mid \langle \text{identifier} \rangle * \langle \text{term} \rangle \\ \langle \text{identifier} \rangle &::= A \mid B \mid C \mid D \mid E \mid F \end{aligned}$$




3. (10) Write the actions of an LR parse for the following string, for the grammar and parse table shown below:

**aa1bbbb**

Grammar:

- (1) S  $\rightarrow$  A
- (2) S  $\rightarrow$  B
- (3) A  $\rightarrow$  a A b
- (4) A  $\rightarrow$  0
- (5) B  $\rightarrow$  a B b b
- (6) B  $\rightarrow$  1

| State | ACTION |     |    |    |     | GOTO |   |   |
|-------|--------|-----|----|----|-----|------|---|---|
|       | a      | b   | 0  | 1  | \$  | S    | A | B |
| 0     | S1     |     | S2 | S3 |     | 11   | 4 | 5 |
| 1     | S1     |     | S2 | S3 |     |      | 6 | 7 |
| 2     |        | r4  | r4 |    | r4  |      |   |   |
| 3     | r6     | r6  | r6 | r6 | r6  |      |   |   |
| 4     | r1     | r1  | r1 |    | r1  |      |   |   |
| 5     | r2     | r2  | r2 | r2 | r2  |      |   |   |
| 6     |        | S8  |    |    |     |      |   |   |
| 7     |        | S9  |    |    |     |      |   |   |
| 8     |        | r3  |    |    |     |      |   |   |
| 9     |        | S10 |    |    |     |      |   |   |
| 10    |        | r5  |    |    | r5  |      |   |   |
| 11    |        |     |    |    | acc |      |   |   |

| Stack          | Input     | Action |
|----------------|-----------|--------|
| \$0            | aa1bbbb\$ | s1     |
| \$0a1          | a1bbbb\$  | s1     |
| \$0a1a1        | 1bbbb\$   | s3     |
| \$0a1a113      | bbbb\$    | r6     |
| \$0a1a1B7      | bbbb\$    | s9     |
| \$0a1a1B7b9    | bbb\$     | s10    |
| \$0a1a1B7b9b10 | bb\$      | r5     |
| \$0a1B7        | bb\$      | s9     |
| \$0a1B7b9      | b\$       | s10    |
| \$0a1B7b9b10   | \$        | r5     |
| \$0B5          | \$        | r2     |
| \$0S11         | \$        | acc    |

4. This question concerns the context-free grammar given below (capital letters denote non-terminals, small letters denote terminals,  $\epsilon$  denotes the empty string, and S denotes the start non-terminal)

- (1)  $S \rightarrow WAB \mid ABCS$
- (2)  $A \rightarrow B \mid WB$
- (3)  $B \rightarrow \epsilon \mid yB$
- (4)  $C \rightarrow z$
- (5)  $W \rightarrow x$

a. (10) Fill in the following table with the First and Follow sets for the grammar (i.e., the First sets of the non-terminals, production right-hand sides, and suffixes of right-hand sides, and the Follow sets of the non-terminals).

| X    | First(X)                         | Follow(X)        |
|------|----------------------------------|------------------|
| S    | <b>x,y,z</b>                     | <b>EOF</b>       |
| A    | <b>x,y,<math>\epsilon</math></b> | <b>EOF,y,z</b>   |
| B    | <b>y,<math>\epsilon</math></b>   | <b>EOF,y,z</b>   |
| C    | <b>z</b>                         | <b>x,y,z</b>     |
| W    | <b>x</b>                         | <b>EOF,x,y,z</b> |
| WAB  | <b>x</b>                         |                  |
| ABCS | <b>x,y,z</b>                     |                  |
| WB   | <b>x</b>                         |                  |
| yB   | <b>y</b>                         |                  |
| AB   | <b>x,y,<math>\epsilon</math></b> |                  |
| BCS  | <b>y,z</b>                       |                  |
| CS   | <b>z</b>                         |                  |

b. (10) Using the First and Follow sets in part (a), fill in the LL(1) parse table below:

|   | x                   | y               | z        | EOF   |
|---|---------------------|-----------------|----------|-------|
| S | S → WAB<br>S → ABCS | S → ABCS        | S → ABCS | error |
| A | A → WB              | A → B           | A → B    | A → B |
| B | error               | B → yB<br>B → ε | B → ε    | B → ε |
| C | error               | error           | C → z    | error |
| W | W → x               | error           | error    | error |

c. (5) Is the grammar LL(1)? Justify your answer.

**The grammar is not LL(1) because some entries in the above table have multiple entries.**

5. You are given the following grammar with the terminal symbols (, ), and **term** and non-terminals S, E and L.

- (1) S → E \$
- (2) E → **term**
- (3) E → ( L )
- (4) L → [ ]
- (5) L → E L

□

a. (5) If the terminal **term** matches the character *x*, give *three* well formed strings in this grammar. □

**Here are few examples of well formed strings in the grammar:**

**X ( ) ( X ) ( X X ) ( X ( X ) ( ) X ( ( X ) X X ) )**

b.(10) A nearly complete collection of LR(0) sets of items for the above grammar is given below. Fill in the missing items and GOTO information. Be sure you complete the entire set, including all necessary GOTOs.

$S_1: \{ S \square \bullet E$   
 $E \square \bullet \text{term}$   
 $E \square \bullet (L) \}$

$S_2: \text{GOTO}(S_1, E)$   
 $\{ S \square E \bullet \square \}$

$S_3: \text{GOTO}(S_1, \text{term})$   
 $\{ E \square \text{term} \bullet \square \}$

$S_4: \text{GOTO}(S_1, ($   
 $\{ E \square ( \bullet L )$   
 $L \square \bullet$   
 $L \square \bullet EL$   
 $E \square \bullet (L)$   
 $E \square \bullet \text{term} \}$

$\text{GOTO}(S_4, ( ) = S_4$

$\text{GOTO}(S_5, E) = S_5$

$\text{GOTO}(S_5, ( ) = S_4$

$\text{GOTO}(S_5, \text{term}) = S_3$

$\text{GOTO}(S_5, ( ) = S_3$

$S_5: \text{GOTO}(S_4, E)$

$\{ L \square E \bullet L$   
 $L \square \bullet$   
 $L \square \bullet EL$   
 $E \square \bullet \text{term}$   
 $E \square \bullet (L) \}$

$S_6: \text{GOTO}(S_4, L)$   
 $\{ E \square (L \bullet ) \}$

$S_7: \text{GOTO}(S_6, ( )$   
 $\{ E \square (L) \bullet \}$

$S_8: \text{GOTO}(S_5, L)$   
 $\{ L \square EL \bullet \square \}$

□

c. (10) Using the items in part (b), fill in the missing information for states 5 and 7 in the SLR(1) parse table. Note that the numbers in the reductions refer to the productions (1 – 5) as given above.

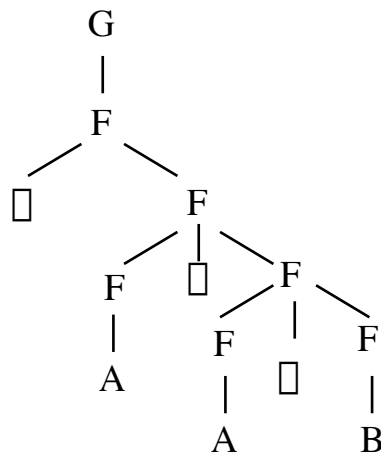
|   | (                                 | )               | term                              | \$              | E        | L        |
|---|-----------------------------------|-----------------|-----------------------------------|-----------------|----------|----------|
| 1 | Shift 4                           | error           | Shift 3                           | error           | 2        |          |
| 2 | error                             | error           | error                             | Accept          |          |          |
| 3 | Reduce 2                          | Reduce 2        | Reduce 2                          | Reduce 2        |          |          |
| 4 | Shift 4<br>Reduce 4               | Reduce 4        | Shift 3<br>Reduce 4               | Reduce 4        | 5        | 6        |
| 5 | <b>Shift 4</b><br><b>Reduce 4</b> | <b>Reduce 4</b> | <b>Shift 3</b><br><b>Reduce 4</b> | <b>Reduce 4</b> | <b>5</b> | <b>8</b> |
| 6 | error                             | Shift 7         | error                             | error           |          |          |
| 7 | <b>Reduce 3</b>                   | <b>Reduce 3</b> | <b>Reduce 3</b>                   | <b>Reduce 3</b> |          |          |
| 8 | Reduce 5                          | Reduce 5        | Reduce 5                          | Reduce 5        |          |          |

6. Consider the following grammar and associated semantic actions. In the actions, the operations And, Or, and Not are constructors for an abstract syntax tree data type.

|                                     |   |
|-------------------------------------|---|
| $G \sqsupseteq F$                   | $G.p = F.p$                                 |
| $F \sqsupseteq F_1 \sqsupseteq F_2$ | $F.p = \text{And}(F_1.p, F_2.p)$            |
| $F \sqsupseteq F_1 \quad F_2$       | $F.p = \text{Or}(F_1.p, F_2.p)$             |
| $F \sqsupseteq \neg F_1$            | $F.p = \text{Neg}(F_1.p)$                   |
| $F \sqsupseteq F_1 \sqsupseteq F_2$ | $F.p = \text{Or}(\text{Not}(F_1.p), F_2.p)$ |
| $F \sqsupseteq (F_1)$               | $F.p = F_1.p$                               |
| $F \sqsupseteq id$                  | $F.p = id.lexeme$                           |

- a. (5) Say whether each attribute of a non-terminal is *inherited* or *synthesized* and why.

**Both attributes G.p and F.p are synthesized. In every production, G.p and F.p are functions of attributes of their child nodes. The can be computed in a bottom-up parse. A simplified parse tree for the expression  $\neg (A \wedge (A \sqsupseteq B))$  is shown here:**



**Using the parse tree, the attribute values can be calculated from the bottom up by applying the semantic actions in the grammar.**

- b. (5) Give the value of the attributes of G after parsing  $\neg (A \sqsupseteq (A \sqsupseteq B))$ .

**The value of G.p is  $\text{Neg}(\text{And}(A, \text{Or}(\text{Not}(A), B)))$ .**

7. (5) Consider the following grammar:

$$\begin{aligned} S' &\rightarrow S \\ S &\rightarrow E = E \\ S &\rightarrow i \\ E &\rightarrow E + i \\ E &\rightarrow i \end{aligned}$$

The start state (the set of LR(0) items) for this grammar is as follows:

$$S_0 : \{ \begin{array}{l} S' \rightarrow \cdot S \\ S \rightarrow \cdot E = E \\ S \rightarrow \cdot i \\ E \rightarrow \cdot E + i \\ E \rightarrow \cdot i \end{array} \}$$

Also we have

$$S_3 = \text{GOTO}(S_0, i) \\ \{ \begin{array}{l} S \rightarrow i \cdot \\ E \rightarrow i \cdot \end{array} \}$$

Because  $\text{FOLLOW}(S) = \{\$, \}$  and  $\text{FOLLOW}(E) = \{\$, +, =\}$ , we have a REDUCE-REDUCE conflict for state 3 in the SLR(1) parse table. Give the corresponding sets of LR(1) items for  $S_0$  and  $S_3$ , which are used to construct the canonical LR parser, and explain why the conflict in the SLR(1) parse table is eliminated.

$$S_0 : \{ \begin{array}{l} S' \rightarrow \cdot S \\ S \rightarrow \cdot E = E, \$ \\ S \rightarrow \cdot i, \$ \\ E \rightarrow \cdot E + i, \{=, +\} \\ E \rightarrow \cdot i, \{=, +\} \end{array} \}$$

$$S_3 = \text{GOTO}(S_0, i) \\ \{ \begin{array}{l} S \rightarrow i \cdot, \$ \\ E \rightarrow i \cdot, \{=, +\} \end{array} \}$$

The lookahead symbol for each production is added for the context in which the production is added to the set of items. When the item  $S' \rightarrow \cdot S$  causes  $S \rightarrow \cdot E = E$  to be added to the set, the lookahead of  $\$$  is associated with the item since what follows the  $S$  is items in  $\text{FOLLOW}(S') = \{\$, \}$ . Next,  $E$  productions are added to  $S_0$  because of the item  $S \rightarrow \cdot E = E$ , and since the  $E$  in this production is followed by  $=$ , this becomes the lookahead symbol associated with the added productions. The  $E$  productions are also added (again) because of the item  $E \rightarrow \cdot E + i$ ; in this context,  $=$  follows the  $E$ , so this is also included as a lookahead symbol on the  $E$  productions.

In state 3, the lookahead symbols associated with each item brought from  $S_0$ . With a lookahead of  $\$$  associated with  $S \rightarrow i \cdot$ , and lookaheads of  $=$  and  $+$  associated with  $E \rightarrow i \cdot$ , the LR(1) parse table will contain REDUCE  $S \rightarrow i$  in cell  $[3, \$]$ , and REDUCE  $E \rightarrow i$  in cells  $[3, =]$  and  $[3, +]$ , thus eliminating the conflict.