M.E. ELECTRONICS AND TELE-COMMUNICATION ENGINEERING FIRST YEAR SECOND SEMESTER - 2018

Subject : COMPILER CONSTRUCTION (COMP) **Time :** 3 Hours **Full Marks :** 100

Instructions : Answer any five questions ; All questions carry equal marks

1.(a) Describe how an NFA may be derived from a given regular expression. Illustrate your answer with " (a|b|c)*ab" as an example regular expression.

10

1(b) Derive a DFA for the NFA obtained in part (a). Also minimize the state of the DFA.

10

2 (a) What are the difficulties encountered in top down parsing? Explain your answer with suitable example.

(b) What is left factoring? Why this is necessary? Explain your answer by a suitable example in connection with top down parsing

(c) Describe an algorithm that will eliminate left recursion from a given grammar. Use the same to eliminate the left recursion from the following grammar

$S \rightarrow A\,a \mid b$

$A \rightarrow A\,c \mid S\,d \mid e$

5+5+10

3. Define Dominator in connection with a flow graph. Give an algorithm for computing dominators. Determine the dominators for all the nodes in the flow graph shown in Fig 1. Also list all the back edges

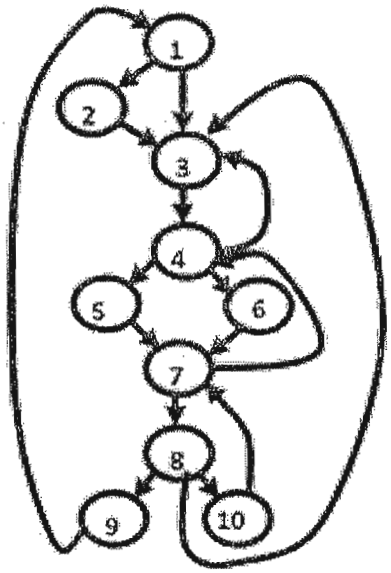

Fig. 1

20

4. Assume that E is a Boolean expression and **and** and **or** are logical connectives. Also assume that the connective **and** has higher precedence than **or**. These connectives may be considered as terminal symbols. Write down all the sets of LR(0) items for the following grammar.

$E' \rightarrow E$
$E \rightarrow E$ **or** E
$E \rightarrow E$ **and** E
$E \rightarrow (E)$
$E \rightarrow$ id

And derive the SLR parsing table for the same. Show that shift reduce conflict is present in the table. Resolve the conflict by using suitable reasoning. Use the modified table to parse a string
" id **and** id **or** id"

05+05+05+05

5. Consider the following grammar

$E \rightarrow E + T$
$T \rightarrow T * F \mid F$
F (E) | id

Construct a predictive parsing table for G. Explain the use of this table by suitable example

20

6 Consider the grammar G as follows:

$S' \rightarrow$ id :=E
$E \rightarrow E + E$
$E \rightarrow E * E$
$E \rightarrow (E)$
$E \rightarrow$ id

Obtain the SLR parsing Table for the above grammar. Parse the string "id:=id+id*id"

15+5

7.(a) Using the Syntax directed Translation scheme for Boolean expression shown in Fig 2, draw the parse tree for the Boolean expressions given below

P<Q or R<S and T<U

and hence find the intermediate code for the same.

10

7(b) Consider the grammar G as follows:

$S \rightarrow cAd$
$A \rightarrow ab \mid a$

Augment the above grammar by left-factoring and Construct a predictive parsing table for G. Explain the use of this table by suitable example

10

8 Consider the grammar

a) $S \rightarrow cAd$
b) $A \rightarrow ab$
c) $A \rightarrow a$

For the above grammar do the following

(i) Design SLR parser
(ii) Derive the operator precedence relations

Using your designed parsers parse the string "cad"

10+10

9 Define

(i) An Operator grammar
(ii) An operator precedence grammar

Show that the following grammar is an operator precedence grammar.

$S \rightarrow a\ A\ c\ B\ e$
$A \rightarrow A\ b \mid b$
$B \rightarrow d$

Obtain the precedence relations for the grammar. Using these relations parse the input string

"a b b c d e "

3+3+4+6+4

Syntax directed Translation scheme for Boolean expression

E → $E^{(1)}$ or M $E^{(2)}$
{ BACKPATCH ($E^{(1)}$.FALSE,M.QUAD);
E.TRUE := MERGE($E^{(1)}$.TRUE, $E^{(2)}$.TRUE);
E.FALSE := $E^{(2)}$.FALSE }

E → $E^{(1)}$ and M $E^{(2)}$
{ BACKPATCH ($E^{(1)}$.TRUE, M.QUAD);
E.TRUE := $E^{(2)}$.TRUE;
E.FALSE := MERGE($E^{(1)}$.FALSE , $E^{(2)}$.FALSE); }

E → not $E^{(1)}$
{ E.TRUE := $E^{(1)}$.FALSE;
E.FALSE := $E^{(1)}$.TRUE }

E → ($E^{(1)}$)
{ E.TRUE := $E^{(1)}$.TRUE;
E.FALSE := $E^{(1)}$.FALSE }

E → id
{ E.TRUE := MAKELIST(NEXTQUAD);
E.FALSE := MAKELIST(NEXTQUAD+!);
GEN(if id.PLACE goto _)
GEN(goto_ }

E → $id^{(1)}$relop $id^{(2)}$
{ E.TRUE := MAKELIST(NEXTQUAD);
E.FALSE := MAKELIST(NEXTQUAD+!);
GEN(if $id^{(1)}$.PLACE relop $id^{(2)}$.PLACE goto_);
GEN(goto_) }

M→ ∈
{ M. QUAD :=NEXT.QUAD}

Fig 2