Register Number

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | | |

**Sri Sivasubramaniya Nadar College of Engineering, Kalavakkam – 603 110**

(An Autonomous Institution, Affiliated to Anna University, Chennai)

Department of Computer Science and Engineering

**Continuous Assessment Test – II**

**Question Paper**

| | | | | | | |
|---|---|---|---|---|---|---|
| **Degree & Branch** | **BE (CSE)** | | | | **Semester** | VI |
| **Subject Code & Name** | **UCS1602 – Compiler Design** | | | | **Regulation:** | **2018** |
| **Academic Year** | 2021-2022 | **Batch** | 2019-2023 | **Date** | **03-05-2022** | **FN** |
| **Time: 90 Minutes**<br>**8.30 – 10.00 am** | **Answer All Questions** | | | | **Maximum: 50 Marks** | |

**Part – A (6×2 = 12 Marks)**

| | | |
|---|---|---|
| <KL1> | What is LR(k) parser?<br>LR parsers are also known as LR(k) parsers, where L stands for left-to-right scanning of the input stream; R stands for the construction of right-most derivation in reverse, and k denotes the number of lookahead symbols to make decisions. | <CO2> |
| <KL1> | How precedence and associativity are handled by YACC compiler?<br>**%left**, for left-associative or **%right** for right associative. The last definition listed has the highest precedence. | <CO2> |
| <KL2> | Explain handle pruning with suitable example.<br>This describes the process of identifying handles and reducing them to the appropriate left most non-terminals.<br>2+3*6 – E →E+T<br>→E+T*F<br>→E+T*6<br>→E+F*6<br>→E+3*6<br>→T+3*6<br>→F +3*6<br>→2+3*6 | <CO2> |
| <KL2> | Show FIRST & FOLLOW for the grammar.<br>S → ABBA<br>A → a \| ε<br>B → b \| ε<br>First(S) = {a,b,ε}, First(A) = {a, ε}, First(B)={b, ε}<br>Follow(S)={$}, Follow(A)={b,$} Follow(B)={a,b,$} | <CO2> |
| <KL1> | What is rule for finding closure{I}, where I is the set of items ?<br>• Initially, every item in I is added to closure(I).<br>• If A → α • B β is in closure (I) and B ⟶ ɤ is a production, then add the item B ⟶ •ɤ ti I, if it is not already in existence, we apply this rule until no more new items can be added to closure(I). | <CO2> |
| <KL2> | Explain the structure of LR parsing table.<br>Parsing table is divided into two parts- Action table and Go-To table. The **action** | <CO2> |

**table** gives a grammar rule to implement the given current state and current terminal in the input stream. There are four cases used in action table as follows.

## Part – B (3×6 = 18 Marks)

<KL3>

7. Consider the grammar G for declaration statements.

```
G: S → TL;
    T → int | float
    L → L,id | id
```

Develop a Syntax checker to recognize the following statements by writing suitable LEX & YACC specifications.

```
int a,b,c;
char e,f;
float h
```

**Lex Code**

```
%{
#include <stdlib.h> #include <stdio.h> #include "y.tab.h" void yyerror(char*);
extern int yylval; int yylineno;
%}

digit [0-9] letter [A-Za-z]
identifier {letter}({letter}|{digit})* number {digit}+
relop ("<"|"<="|">"|">="|"=="|"!=")
arithop ("+"|"\-"|"*"|"/"|"%")
%%
[ \t]+ {};
[\n] {yylineno++;} int { return INT;}
float { return FLOAT;} double { return DOUBLE;} char { return CHAR;}
if { return IF;} else { return ELSE;}
while { return WHILE;} for { return FOR;}
"=" { return ASG;}
{identifier}     { return ID;}
{number}         { return NUMBER;}
{arithop} { return ARITH_OP;}
{relop} { return REL_OP;}
. { return *yytext;}
%%
```

**Yacc Code**

```
%{
#include <stdlib.h> #include <stdio.h> #include <math.h> int yylex(void);
void yyerror(char *s); int yylineno;
#include "y.tab.h"

%}

%token NUMBER
%token ARITH_OP
%token REL_OP
%token ID
%token ASG
%token INT FLOAT DOUBLE CHAR IF ELSE WHILE FOR
```

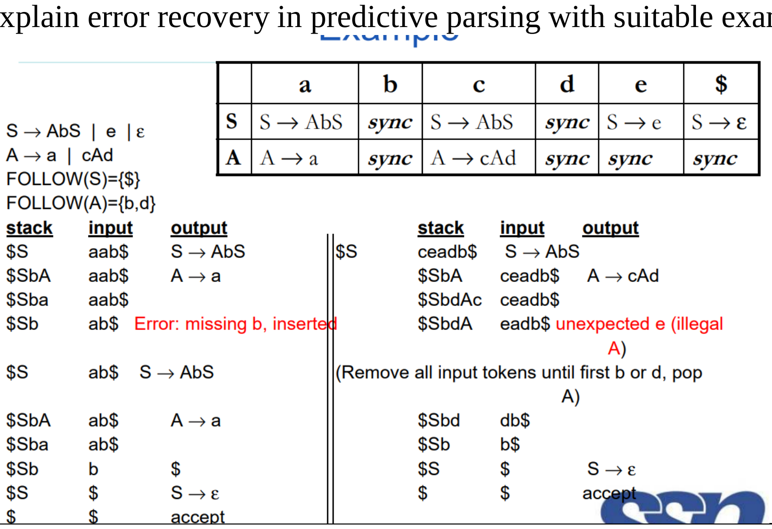
<CO2>

```
%nonassoc IFX
%nonassoc ELSE
%left ARITH_OP
%left REL_OP
%%

code: code stmt
| '{' code '}'
| stmt

stmt: declStmt ';'
| assgStmt ';'
| condStmt
| loopStmt declStmt: type declList
declList: declList ',' declInit
| declInit

declInit: ID
| assgStmt


condStmt: IF '(' expr ')' code
| IF '(' expr ')' code ELSE code

loopStmt: WHILE '(' expr ')' code
| FOR '(' forDecl ';' forCond ';' forUpda ')' code

forDecl: declStmt
| declList
| epsilon forCond: expr


| epsilon

forUpda: assgStmt
| epsilon


epsilon: ;


expr: expr REL_OP expr
| expr1
;

expr1: expr1 ARITH_OP expr1
|expr2
;

expr2:'(' expr ')'
| ID
| NUMBER
;

assgStmt: ID ASG expr ; type: INT
```

```
| FLOAT
| DOUBLE
| CHAR
;

%%

void yyerror(char *s)
{
fprintf(stderr,  "line  %d:  %s\n",  yylineno,  s); return;
}

int yywrap()
{
return(1);
}

int main(void)
{
if(!yyparse())
printf("Syntactically Correct!\n");
else
printf("Syntactically Incorrect!\n");
return 0;
}
```

<KL2>

8. Explain error recovery in predictive parsing with suitable examples.



Example

S → AbS | e | ε
A → a | cAd
FOLLOW(S)={$}
FOLLOW(A)={b,d}

| | a | b | c | d | e | $ |
|---|---|---|---|---|---|---|
| **S** | S → AbS | *sync* | S → AbS | *sync* | S → e | S → ε |
| **A** | A → a | *sync* | A → cAd | *sync* | *sync* | *sync* |

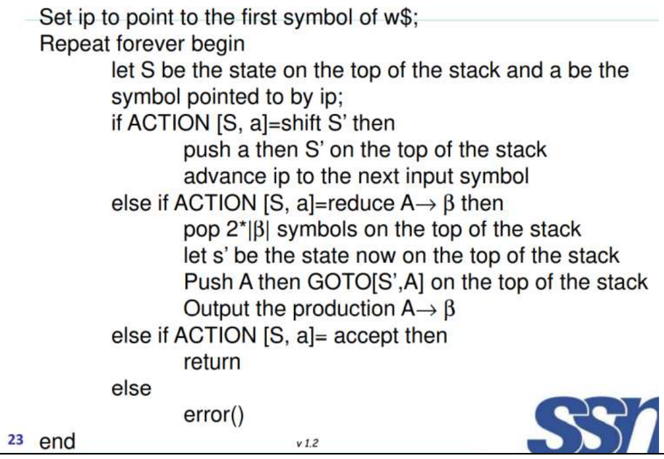
```
stack     input     output
$S        aab$      S → AbS
$SbA      aab$      A → a
$Sba      aab$
$Sb       ab$   Error: missing b, inserted

$S        ab$   S → AbS

$SbA      ab$       A → a
$Sba      ab$
$Sb       b         $
$S        $         S → ε
$         $         accept
```

```
          stack     input     output
$S        ceadb$     S → AbS
          $SbA      ceadb$    A → cAd
          $SbdAc    ceadb$
          $SbdA     eadb$ unexpected e (illegal
                              A)
(Remove all input tokens until first b or d, pop
                         A)
          $Sbd      db$
          $Sb       b$
          $S        $         S → ε
          $         $         accept
```

<CO2>

<KL2>

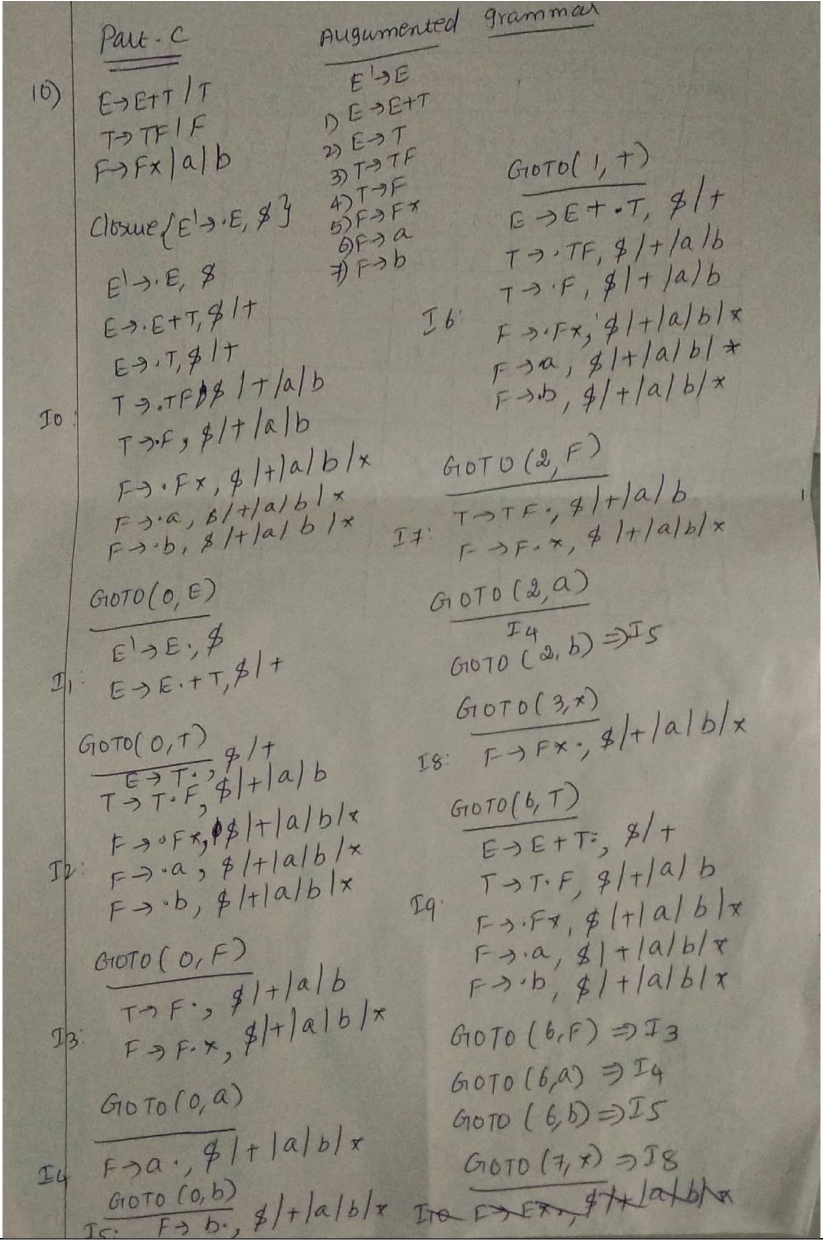
9. Write the LR parsing algorithm.

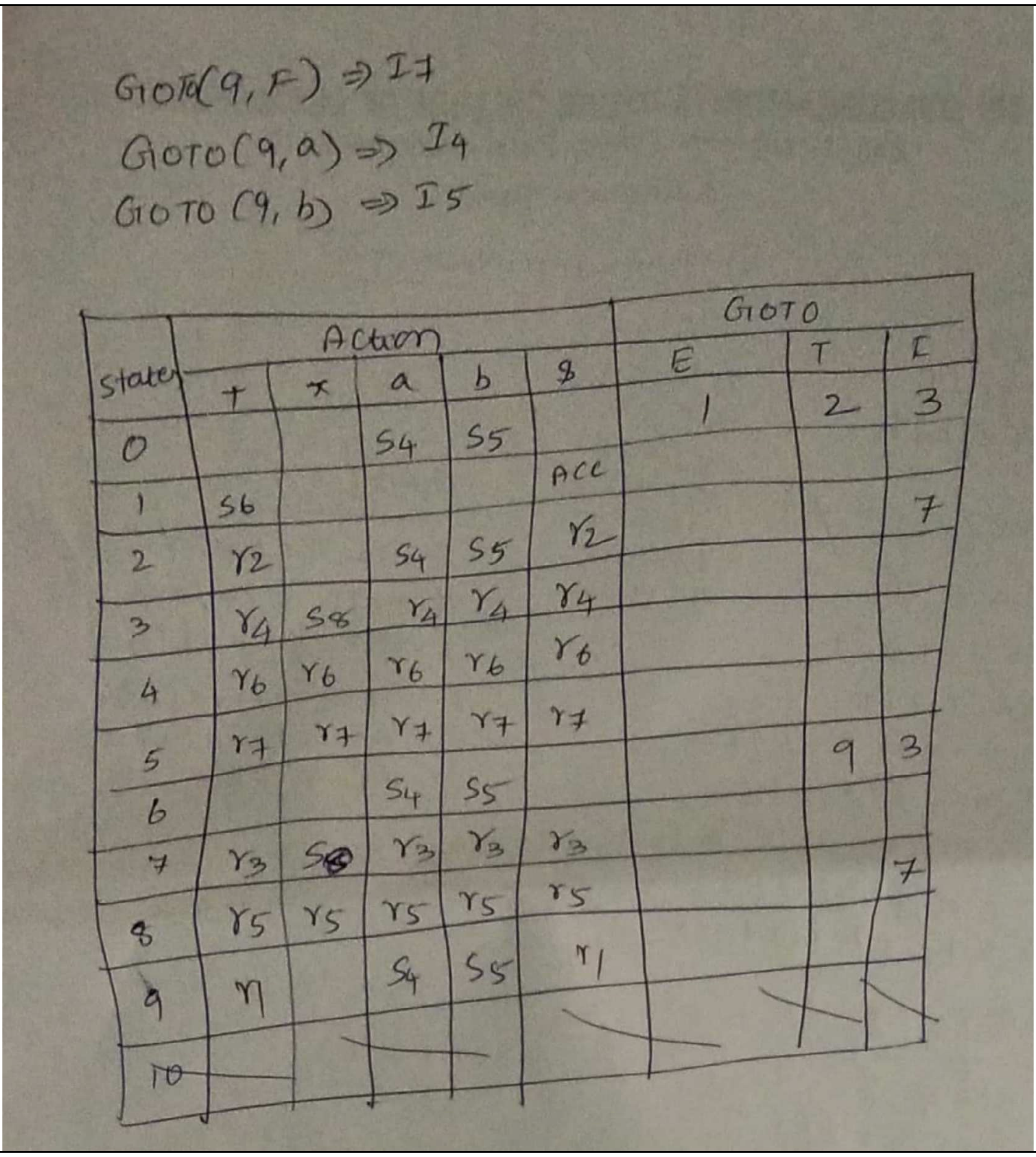


```
Set ip to point to the first symbol of w$;
Repeat forever begin
        let S be the state on the top of the stack and a be the
        symbol pointed to by ip;
        if ACTION [S, a]=shift S' then
                push a then S' on the top of the stack
                advance ip to the next input symbol
        else if ACTION [S, a]=reduce A→ β then
                pop 2*|β| symbols on the top of the stack
                let s' be the state now on the top of the stack
                Push A then GOTO[S',A] on the top of the stack
                Output the production A→ β
        else if ACTION [S, a]= accept then
                return
        else
                error()
end
```

<CO2>

## Part – C (2×10 = 20 Marks)

| | | |
|---|---|---|
| <KL3> | 10. Construct CLR parsing table for the grammar.<br>$E \rightarrow E + T \mid T$<br>$T \rightarrow TF \mid F$<br>$F \rightarrow F^{*} \mid a \mid b$<br><br>Part - C<br>10) $E \rightarrow E+T / T$<br>$T \rightarrow TF / F$<br>$F \rightarrow F* \mid a \mid b$<br>Closure$\{E' \rightarrow \cdot E, \$\}$<br>I0: $E' \rightarrow \cdot E, \$$<br>$E \rightarrow \cdot E+T, \$/+$<br>$E \rightarrow \cdot T, \$/+$<br>$T \rightarrow \cdot TF, \$/+/a/b$<br>$T \rightarrow \cdot F, \$/+/a/b$<br>$F \rightarrow \cdot F*, \$/+/a/b/*$<br>$F \rightarrow \cdot a, \$/+/a/b/*$<br>$F \rightarrow \cdot b, \$/+/a/b/*$<br>GOTO(0, E)<br>I1: $E' \rightarrow E \cdot, \$$<br>$E \rightarrow E \cdot +T, \$/+$<br>GOTO(0, T)<br>I2: $E \rightarrow T \cdot, \$/+$<br>$T \rightarrow T \cdot F, \$/+/a/b$<br>$F \rightarrow \cdot F*, \$/+/a/b/*$<br>$F \rightarrow \cdot a, \$/+/a/b/*$<br>$F \rightarrow \cdot b, \$/+/a/b/*$<br>GOTO(0, F)<br>I3: $T \rightarrow F \cdot, \$/+/a/b$<br>$F \rightarrow F \cdot *, \$/+/a/b/*$<br>GOTO(0, a)<br>I4 $F \rightarrow a \cdot, \$/+/a/b/*$<br>GOTO(0, b)<br>I5: $F \rightarrow b \cdot, \$/+/a/b/*$<br>Augumented grammar<br>$E' \rightarrow E$<br>1) $E \rightarrow E+T$<br>2) $E \rightarrow T$<br>3) $T \rightarrow TF$<br>4) $T \rightarrow F$<br>5) $F \rightarrow F*$<br>6) $F \rightarrow a$<br>7) $F \rightarrow b$<br>GOTO(1, +)<br>I6: $E \rightarrow E + \cdot T, \$/+$<br>$T \rightarrow \cdot TF, \$/+/a/b$<br>$T \rightarrow \cdot F, \$/+/a/b$<br>$F \rightarrow \cdot F*, \$/+/a/b/*$<br>$F \rightarrow a, \$/+/a/b/*$<br>$F \rightarrow \cdot b, \$/+/a/b/*$<br>GOTO(2, F)<br>I7: $T \rightarrow TF \cdot, \$/+/a/b$<br>$F \rightarrow F \cdot *, \$/+/a/b/*$<br>GOTO(2, a)<br>I4<br>GOTO(2, b) ⇒ I5<br>GOTO(3, *)<br>I8: $F \rightarrow F* \cdot, \$/+/a/b/*$<br>GOTO(6, T)<br>I9: $E \rightarrow E+T \cdot, \$/+$<br>$T \rightarrow T \cdot F, \$/+/a/b$<br>$F \rightarrow \cdot F*, \$/+/a/b/*$<br>$F \rightarrow \cdot a, \$/+/a/b/*$<br>$F \rightarrow \cdot b, \$/+/a/b/*$<br>GOTO(6, F) ⇒ I3<br>GOTO(6, a) ⇒ I4<br>GOTO(6, b) ⇒ I5<br>GOTO(7, *) ⇒ I8<br>~~I10 $F \rightarrow F*, \$/+/a/b/*$~~ | <CO2> |



GOTO(9, F) ⇒ I7

GOTO(9, a) ⇒ I4

GOTO (9, b) ⇒ I5

| state | Action | | | | | GOTO | | |
|---|---|---|---|---|---|---|---|---|
| | + | * | a | b | $ | E | T | F |
| 0 | | | S4 | S5 | | 1 | 2 | 3 |
| 1 | S6 | | | | ACC | | | 7 |
| 2 | r2 | | S4 | S5 | r2 | | | |
| 3 | r4 | S8 | r4 | r4 | r4 | | | |
| 4 | r6 | r6 | r6 | r6 | r6 | | | |
| 5 | r7 | r7 | r7 | r7 | r7 | | 9 | 3 |
| 6 | | | S4 | S5 | | | | |
| 7 | r3 | S8 | r3 | r3 | r3 | | | 7 |
| 8 | r5 | r5 | r5 | r5 | r5 | | | |
| 9 | r1 | | S4 | S5 | r1 | | | |
| 10 | | | | | | | | |

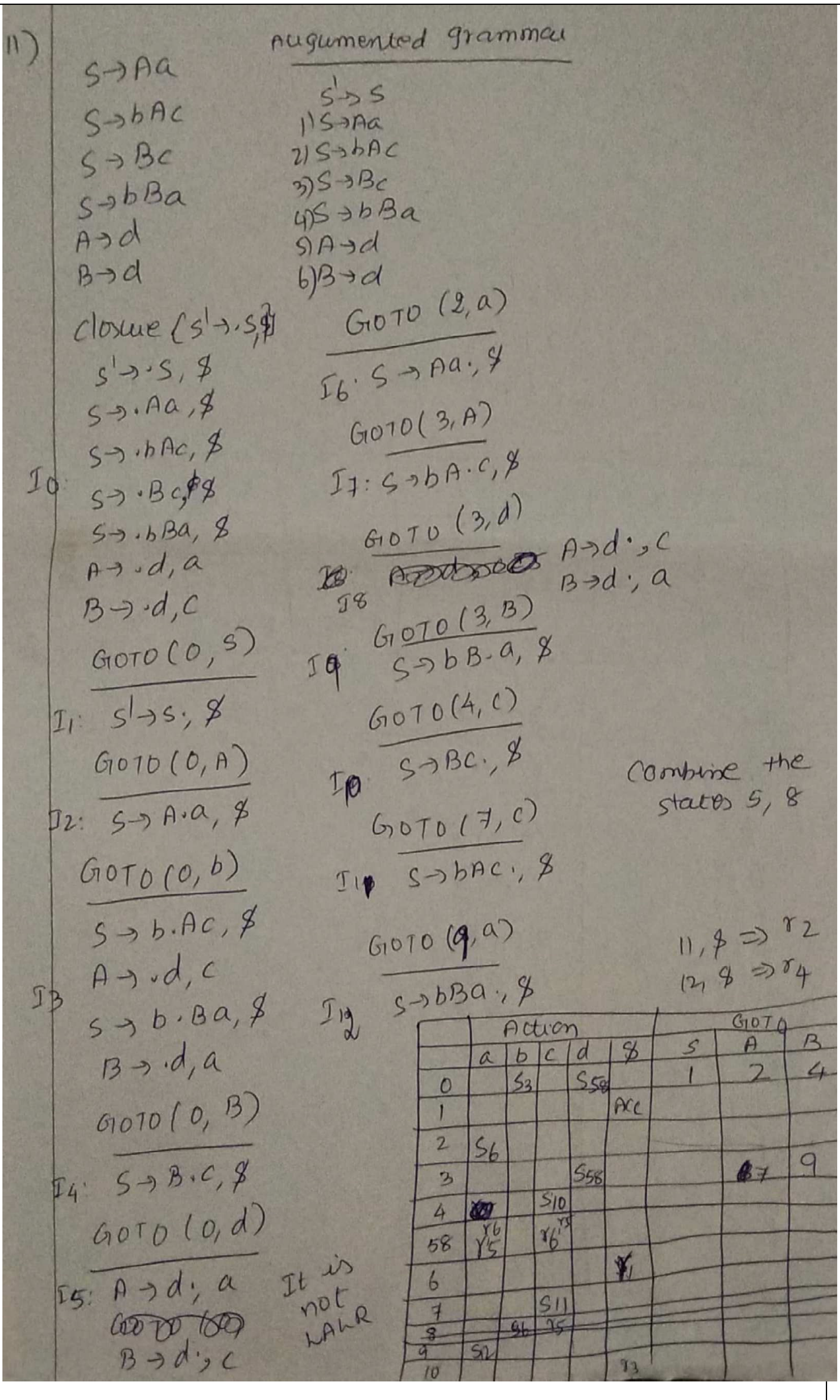
(OR)

<KL3>

11. Construct LALR parser for the grammar and show that the grammar is not LALR(1).

S → Aa |bAc |Bc |bBa<br>
A → d<br>
B → d

<CO2>



11)

Augumented grammar

S→Aa
S→bAc
S→Bc
S→bBa
A→d
B→d

S'→S
1) S→Aa
2) S→bAc
3) S→Bc
4) S→bBa
5) A→d
6) B→d

Closure {S'→.S,$}
S'→.S, $
S→.Aa, $
S→.bAc, $
I0 S→.Bc, $
S→.bBa, $
A→.d, a
B→.d, c

GOTO (0, S)
I1: S'→S., $

GOTO (0, A)
I2: S→A.a, $

GOTO (0, b)
S→b.Ac, $
A→.d, c
I3 S→b.Ba, $
B→.d, a

GOTO (0, B)
I4: S→B.c, $

GOTO (0, d)
I5: A→d., a
B→d., c

GOTO (2, a)
I6: S→Aa., $

GOTO (3, A)
I7: S→bA.c, $

GOTO (3, d)
I8 A→d., c
B→d., a

GOTO (3, B)
I9 S→bB.a, $

GOTO (4, c)
I10 S→Bc., $

GOTO (7, c)
I11 S→bAc., $

GOTO (9, a)
I12 S→bBa., $

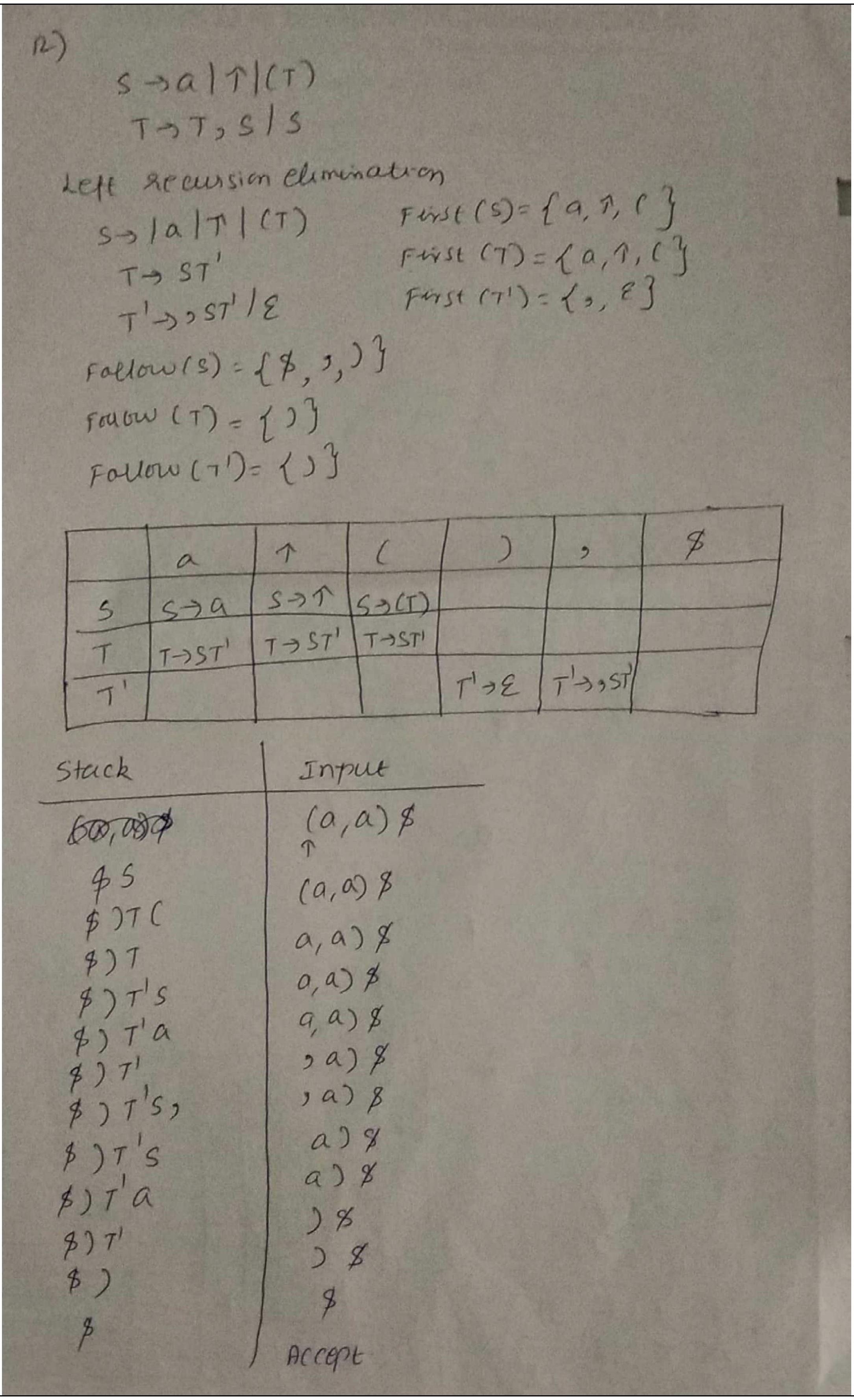
Combine the states 5, 8

11, $ ⇒ r2
12, $ ⇒ r4

It is not LALR

| | Action | | | | | GOTO | | |
|---|---|---|---|---|---|---|---|---|
| | a | b | c | d | $ | S | A | B |
| 0 | | S3 | | S58 | | 1 | 2 | 4 |
| 1 | | | | | Acc | | | |
| 2 | S6 | | | | | | | |
| 3 | | | | S58 | | | 7 | 9 |
| 4 | | | S10 | | | | | |
| 58 | r6<br>r5 | | r6<br>r5 | | | | | |
| 6 | | | | | r1 | | | |
| 7 | | | S11 | | | | | |
| 8 | | S6 | r5 | | | | | |
| 9 | S12 | | | | | | | |
| 10 | | | | | r3 | | | |

| | | |
|---|---|---|
| <KL3> | 12. Construct Predictive parsing table for the given grammar and parse the sentence ***(a,a)***<br>S → a \| ↑ \| (T)<br>T → T , S \| S | <CO2> |



12)

S → a | ↑ | (T)

T → T , S | S

Left Recursion elimination

S → | a | ↑ | (T)

T → ST'

T' → , ST' | ε

First(S) = { a, ↑, ( }

First(T) = { a, ↑, ( }

First(T') = { , , ε }

Follow(S) = { $, , , ) }

Follow(T) = { ) }

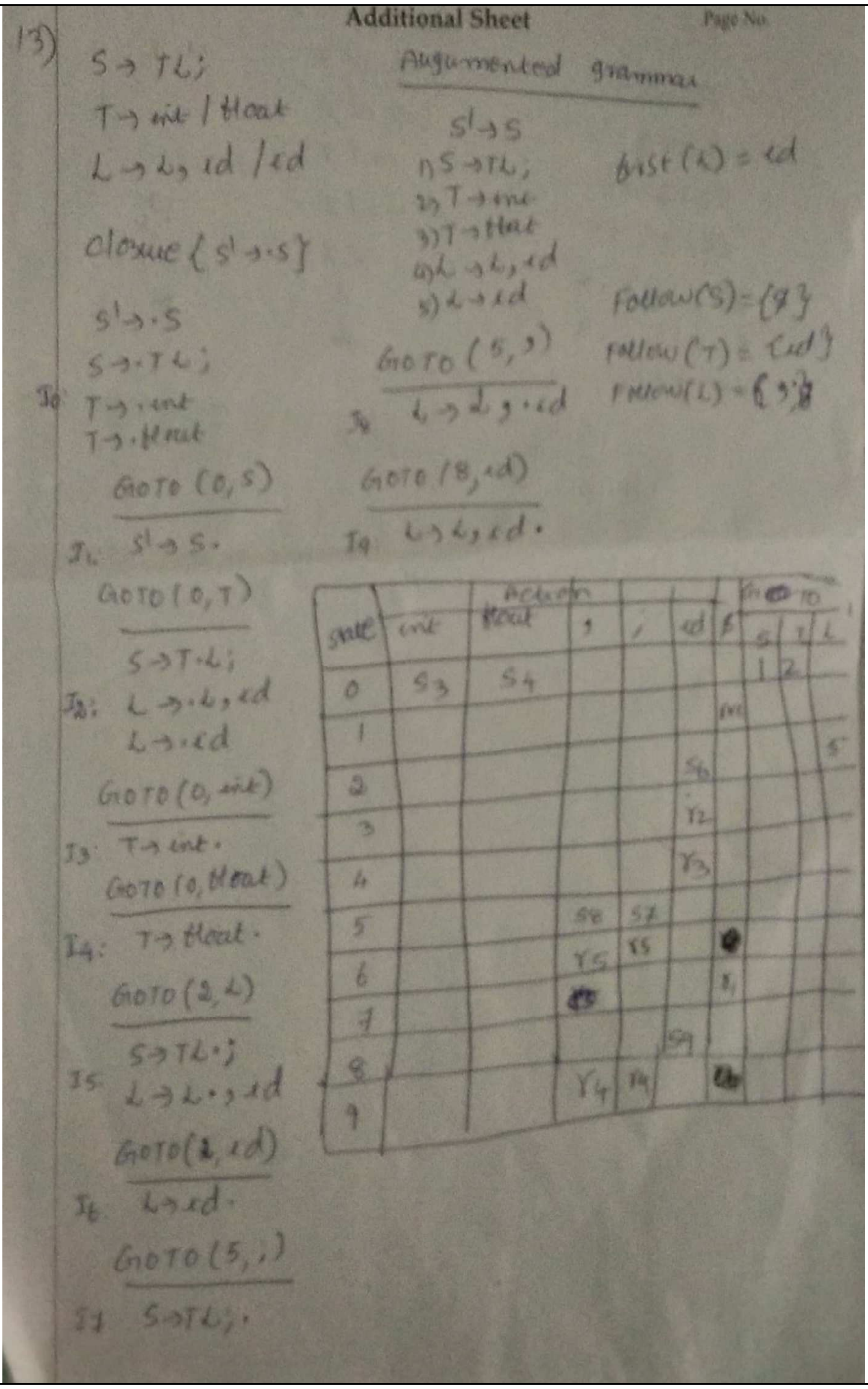
Follow(T') = { ) }

| | a | ↑ | ( | ) | , | $ |
|---|---|---|---|---|---|---|
| S | S → a | S → ↑ | S → (T) | | | |
| T | T → ST' | T → ST' | T → ST' | | | |
| T' | | | | T' → ε | T' → , ST' | |

| Stack | Input |
|---|---|
| ~~(a,a)$~~ | (a,a)$<br>↑ |
| $S | (a,a)$ |
| $)T( | a,a)$ |
| $)T | a,a)$ |
| $)T'S | a,a)$ |
| $)T'a | a,a)$ |
| $)T' | ,a)$ |
| $)T'S, | ,a)$ |
| $)T'S | a)$ |
| $)T'a | a)$ |
| $)T' | )$ |
| $) | )$ |
| $ | $ |
| | Accept |

(OR)

| | | |
|---|---|---|
| <KL3> | 13. Construct SLR parser for the grammar G. Parse the string ***int id,id***<br>G: S → TL;<br>T → int \| float<br>L → L,id \| id | <CO2> |



**Additional Sheet** Page No.

13) S → TL;

T → int / float

L → L, id / id

closure { S' → .S }

S' → .S

S → .TL;

I0 T → .int

T → .float

GOTO (0, S)

I1 S' → S.

GOTO (0, T)

S → T.L;

I2: L → .L, id

L → .id

GOTO (0, int)

I3 T → int.

GOTO (0, float)

I4: T → float.

GOTO (2, L)

S → TL.;

I5 L → L., id

GOTO (2, id)

I6 L → id.

GOTO (5, ;)

I7 S → TL;.

Augumented grammar

S' → S

1) S → TL;

2) T → int

3) T → float

4) L → L, id

5) L → id

GOTO (5, ,)

I8 L → L, .id

GOTO (8, id)

I9 L → L, id.

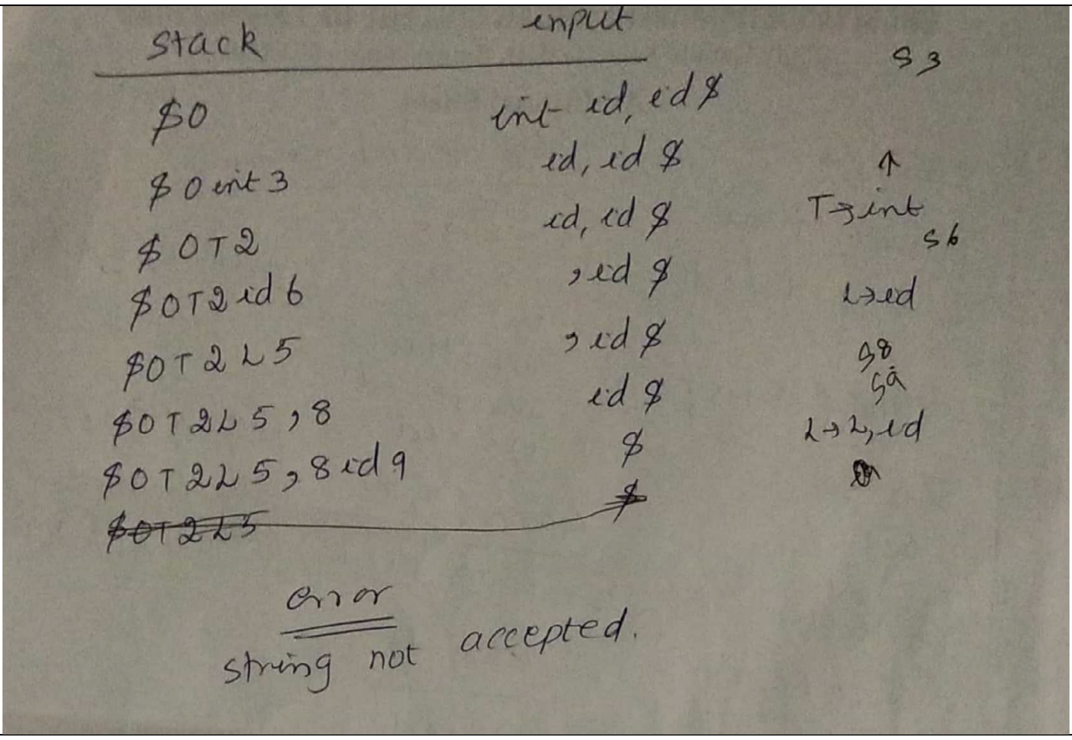
first (L) = id

Follow(S) = {$}

Follow (T) = {id}

Follow(L) = {;,}

| state | Action | | | | | | Goto | | |
|---|---|---|---|---|---|---|---|---|---|
| | int | float | , | ; | id | $ | S | T | L |
| 0 | S3 | S4 | | | | | 1 | 2 | |
| 1 | | | | | | acc | | | |
| 2 | | | | | S6 | | | | 5 |
| 3 | | | | | r2 | | | | |
| 4 | | | | | r3 | | | | |
| 5 | | | S8 | S7 | | | | | |
| 6 | | | r5 | r5 | | [illegible] | | | |
| 7 | | | [illegible] | | | r1 | | | |
| 8 | | | | | S9 | | | | |
| 9 | | | r4 | r4 | | [illegible] | | | |

| | |
|---|---|
| |  |

| stack | input | |
|---|---|---|
| $0 | int id, id $ | S3 |
| $0 int 3 | id, id $ | ↑ |
| $0T2 | id, id $ | T→int |
| $0T2id6 | , id $ | S6<br>L→id |
| $0T2L5 | , id $ | S8<br>S9 |
| $0T2L5,8 | id $ | L→L, id |
| $0T2L5,8id9 | $ | |
| ~~$0T2L5~~ | $ | |

error

string not accepted.

--------------

| Prepared By | Reviewed By | Approved By |
|---|---|---|
| Course Coordinator | PAC Team | HOD |