Roll No.: _

Amrita Vishwa Vidyapeetham

Amrita School of Engineering, Coimbatore

B.Tech. Second Assessment Examinations – February 2019

Sixth Semester

Computer Science and Engineering

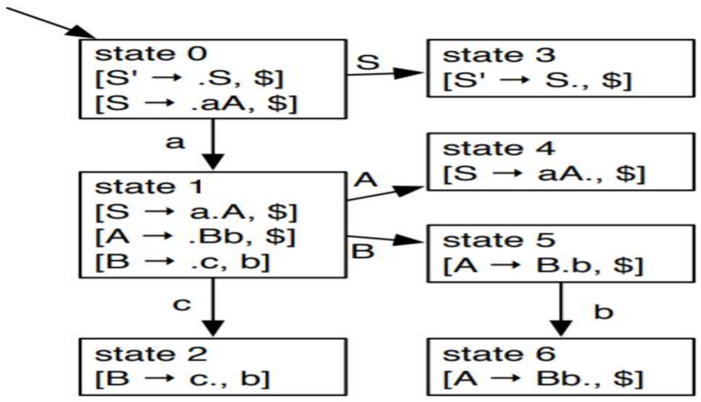
15CSE311 Compiler Design

[Time: Two hours   Maximum: 50 Marks]

CO	Course Outcomes
CO01	Understand modularization and apply theoretical concepts to compiler construction
CO02	Apply algorithms for analyzing lexemes and structural correctness of source programs
CO03	Analyze the design of type systems and check the correctness of code semantics
CO04	Experiment intermediate representation and perform code control flow checking
CO05	Analyze function executions using activation records

Answer all questions

1. Given a grammar G and an input string S that belongs to the grammar: [3] [CO02]
   a. Generalize the total number of parse tree nodes using only the LR(1) actions (say, formula 1) and state the relationship between number of parse tree nodes (formula 1) and the number stack operations performed (say, formula 2). (1 Mark)
   Note: write formula 1 and 2 based on the above description.
   b. Relate the dot (**.**) operator in LR(1) item to the general right-sentential form: (NT | T)* T*. (Hint: Name of the position in right-sentential form) (1 Mark)
   c. Give the formal definition of handle in bottom-up parser. (1 Mark)
2. Given the following augmented grammar, construct the DFA showing canonical collection of LR(1) items: [8] [CO02]
   0. S’ $\rightarrow$ S
   1. S $\rightarrow$AA
   2. A $\rightarrow$ aA | b
3. Given the following canonical collection of LR(1) items, construct the ***action and goto*** tables and **write the *grammar*** that generates this canonical set of LR(1) items. [5] [CO02]



4. Consider the following states in a LR(1) parser for a grammar G. Which states would be merged in a LALR(1) parser? Is the grammar G an LALR(1) grammar? Justify. [3] [CO02]

State 1	State 2	State 3	State 4
[A → ab •, c]	[A → ab •, c]	[A → ab •, c]	[A → ab •, b]
[A → b •, a]	[A → b •, b]	[A → • b , c]	[A → b •, c]
[A → • ba, b]	[A → b • a, c]	[A → b • a, c]	[A → b • a, c]

5. Given the following grammar and its corresponding LR(1) parse table, perform LR(1) parsing showing the stack, input and action at each stage of parsing of the input string: **array 5 of int x** (Note: SL',SL,VL and Idx are individual non-terminals each.) [5] [CO02]

$1. SL' \rightarrow SL$

$2. SL \rightarrow S\ ;\ SL$

$3. SL \rightarrow S$

$4. S \rightarrow T\ VL$

$5. T \rightarrow$ **array** Idx **of** T

$6. T \rightarrow$ **int**

$7. VL \rightarrow$ **id** , VL

$8. VL \rightarrow$ **id**

$9. Idx \rightarrow$ **num**

	action								goto				
states	**array**	**int**	**num**	**id**	**,**	**;**	**of**	**$**	*SL*	*S*	*VL*	*T*	*Idx*
0	s1	s2							3	4		5	
1			s7										6
2				r6									
3								acc					
4						s8		r3					
5				s10							9		
6							s11						
7							r9						
8	s1	s2							12	4		5	
9						r4		r4					
10					s13	r8		r8					
11	s1	s2										14	
12								r2					
13				s10							15		
14				r5									
15						r7		r7					

6. Given the following items in a single state of an LR(1) parser, identify if any conflicts exist and if so specify for what look ahead symbols they occur. [3] [CO02]

[A → • dAd, b]
[B → dab • h, a]
[C → b • ad, c]
[D → ab • d, h]
[E → cab •, a]
[F → aAab • , c]
[G → hab • , c]
[H → aA • , b]

7. Given the following 'C' program, identify the semantic errors (both compile time and runtime) and write down only the line number, the error and the type of semantic error (as static/dynamic semantic error). [5] [CO03]

```
1.  #include<stdio.h>
2.  int f(int k, char s[10])
3.  {
4.    int char=0,k;
5.     for(i=0;i<k;i++)
6.       {
7.        float x=5.5;
8.        char=printf("\n%d%c",i+1,s[i]);
9.       }
10.  printf("\n%c",s[20]);
11.  return char;
12.  printf("\n%f",x);
13. }
14. int main()
15. {
16. int size,i=0;
17. char *s=NULL,c[10];
18. printf("enter the string");
19. scanf("%s",s);
20. printf("\n%s",s);
21. int count;
22.   while(s[i]!='\0')
23.    {
24.       size++;
25.       i++;
26.    }
27. printf("\n%d",size);
28. c=f(s);
29. printf("\n%d",c);
30. }
```

8. Given the following attribute grammar, parse and evaluate the string **.1001**. **Identify** the inherited and synthesized attributes and **justify** as to why they are inherited or synthesized. [4] [CO03]

S.No	Production	Semantic Rule
1.	N → **.**L	{N.v = L.v; L.c = -1}
2.	L → B L_1	{L.v=B.v+L_1.v; L_1 .c=L.c-1; B.c=L.c}
3.	L → ε	{L.v=0}
4.	B → 0	{B.v=0}
5.	B → 1	{B.v=2*B.c}

9. Write an attribute grammar that computes the number of executed statements for a program conforming to the below grammar: [8] [CO03]

Program → Stmts
Stmts → Stmt Stmts | Stmt
Stmt → AssignSt | ForSt
AssignSt → a=b
ForSt → FOR x := Number TO Number DO Stmts END
Number → 1 | 2 |…..| 10

List down each attribute in your attribute grammar and specify as to whether that attribute is synthesized or inherited **with proper reasons** for the input string:

FOR x:=1 TO 5 DO a=b END

For the same input string given above, show the annotated parse tree, dependence graph and the evaluation order using topological sort.

10. Consider the following attribute grammar: [6] [CO03]

S.No.	Production	Semantic Rules
1.	N → D	N.trans := spell(D.val)
2.	N → D S	S.in ::= D.val N.trans ::= S.trans

3.	S → D	S.val := **if** D.val = 0 **then** *decade*(S.in)
		else if S.in ≤ 1 **then** *spell*(10*S.in +D.val)
		else *decade*(P.in) || *spell*(D.val)

4.	D → 0	D.val := 0
5.	D → 1	D.val := 1
6.	D → 2	D.val := 2
7.	D → 3	D.val := 3
8.	D → 4	D.val := 4
9.	D → 5	D.val := 5
10.	D → 6	D.val := 6
11.	D → 7	D.val := 7
12.	D → 8	D.val := 8
13.	D → 9	D.val := 9

The functions *spell* and *decade* works as follows*:*
spell(1) = one, *spell*(2) = two, …, *spell*(19) = nineteen
decade(0) = zero, *decade*(1) = ten, …, *decade*(9) = ninety

Show the annotated parse tree, dependency graph and evaluation order using topological sort for the input string **92**. Provide the list of synthesized and inherited attributes.

****