USN | | | | | | | | | | |

CMRIT – CMR INSTITUTE OF TECHNOLOGY, BENGALURU. ACCREDITED WITH A+ GRADE BY NAAC

Internal Assessment Test II – June, 2021

Sub:	System Software & Compilers					Sub Code:	18CS61	Branch:	CSE
Date:	22/06/2021	Duration:	90 min's	Max Marks:	50	Sem/Sec:	6/CSE(A,B,C)	OBE	

Answer any FIVE FULL Questions MARKS | CO | RBT

1. **Given the grammar: E -> E - T | T , T -> T / F | F, F -> (E) | NUM** [10] CO2 L3

Construct the predictive parsing table and show the moves made by predictive parser on the input string 6 / 3 – 1

Solution

	FIRST	FOLLOW
E→TE'	**(, NUM**	**$,)**
E'→-TE'	**- , Ɛ**	**$,)**
T→FT'	**(, NUM**	**-, $,)**
T'→/FT'|Ɛ	**/ , Ɛ**	**-, $,)**
F→(E) / NUM	**(, NUM**	**/, -, $,)**

Construction of predictive parsing table

	-	/	(	)	NUM	$
E			**E→TE'**		**E→TE'**	
E'	**E'→-TE'**			**E'→ Ɛ**		**E'→ Ɛ**
T			**T→FT'**		**T→FT'**	
T'	**T'→ Ɛ**	**T'→/FT'**		**T'→ Ɛ**		**T'→ Ɛ**
F			**F→(E)**		**F→NUM**	

Parsing of input string **6 / 3 – 1**

Stack	Input	Action
E$	NUM / NUM - NUM $	Apply E→TE'
TE'$	NUM / NUM - NUM $	Apply T→FT'
FT'E'$	NUM / NUM - NUM $	Apply F→NUM
NUMT'E'$	NUM / NUM - NUM $	Match NUM
T'E'$	/ NUM - NUM $	Apply T'→/FT'
/FT'E'$	/ NUM - NUM $	Match /
FT'E'$	NUM - NUM $	Apply F→ NUM
NUMT'E'$	NUM - NUM $	Match NUM
T'E'$	- NUM $	Apply T'→ Ɛ
E'$	- NUM $	Apply E'→-TE'
-TE'$	- NUM $	Match -
TE'$	NUM $	Apply T→FT'
FT'E'$	NUM $	Apply F→NUM
NUMT'E'$	NUM $	Match NUM
T'E'$	$	Apply T'→ Ɛ
E'$	$	Apply E'→ Ɛ
$	$	Match

2. **Explain handle pruning? Consider the following grammar.** [10] CO2 L3
S→ T L;
T→ int | float
L→ L,id | id
Parse the input string int a, b; using shift-reduce parser and draw the bottom-up parse tree.

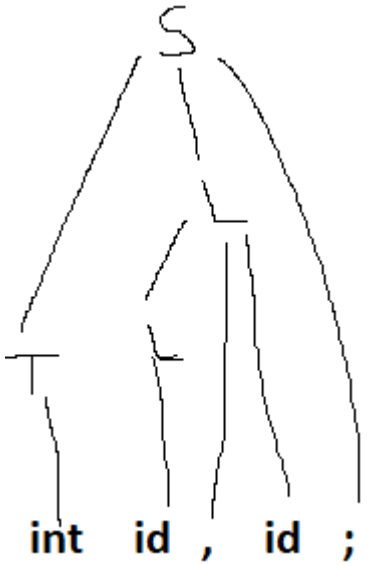
Solution

Handle Purning

The general approach used in shift-and-reduce parsing. A Handle is a substring that matches the body of a production. Handle reduction is a step in the reverse of rightmost derivation. A rightmost derivation in reverse can be obtained by handle pruning

Stack	**Input string**	**Action**
$	Int id,id; $	Shift int
$int	id,id; $	Reduce T→int
$T	id,id; $	Shift a
$T id	, id; $	Reduce L→id
$T L	, id; $	Shift ,
$T L,	id;$	Shift b
$T L,id	;$	Reduce L→L,id
$T L	;$	Shift ;
$TL;	$	Reduce S→TL;
$S	$	Accept

Construction of Bottom up parse tree

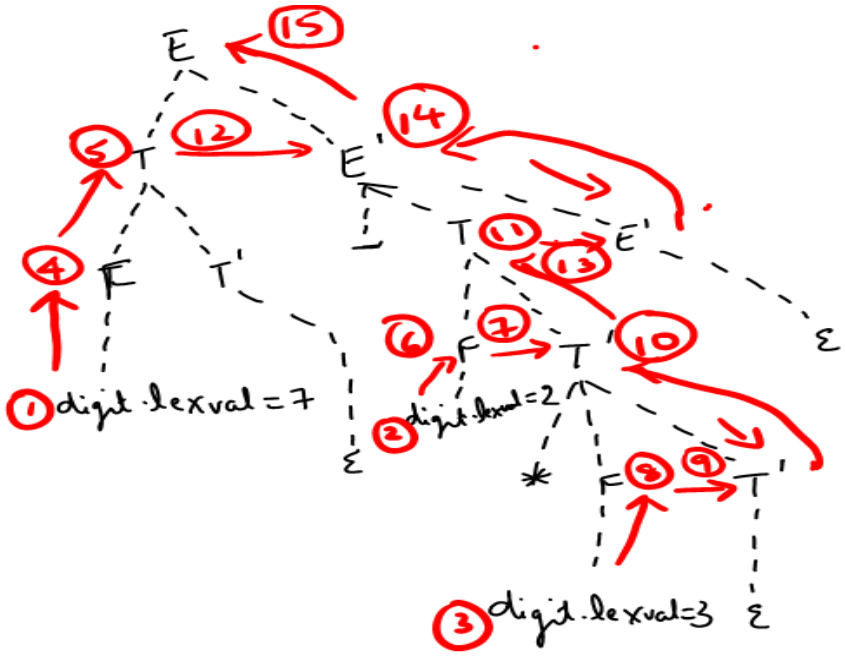


3. **Consider the following grammar. Draw a dependency graph for the expression 7-2*3 in Top down parsing. Also write the order of evaluation.** [10] CO2 L3

E→E-T | T T→T*F | F F→ digit

Solution

PRODUCTIONS	SEMANTIC RULE
E→TE'	E'.inh=T.val E.val=E'.syn
E'→-TE1'	E1',inh=E'.inh E1'.inh=E1'.inh-T.val E1'.syn=E1'.inh
E'→ ε	E'.syn= E1'.syn
T→FT'	T'.inh=F.val T.val=T'.syn
T'→*FT1'	T1'.inh=T'.inh T1'.inh= T1'.inh* F.val T1'.syn=T1'.inh
T'→ ε	T'.syn=T1'.syn
F→digit	F.val.=digit.lexval



Order of Evaluation is: 1 4 2 6 3 8 5 7 9 10 11 12 13 14 15

4. **Define SDD and SDT. Write SDD for simple desk calculator and show annotated parse tree for the expression 7+8* (6-5)** [10] CO2 L3

Solution

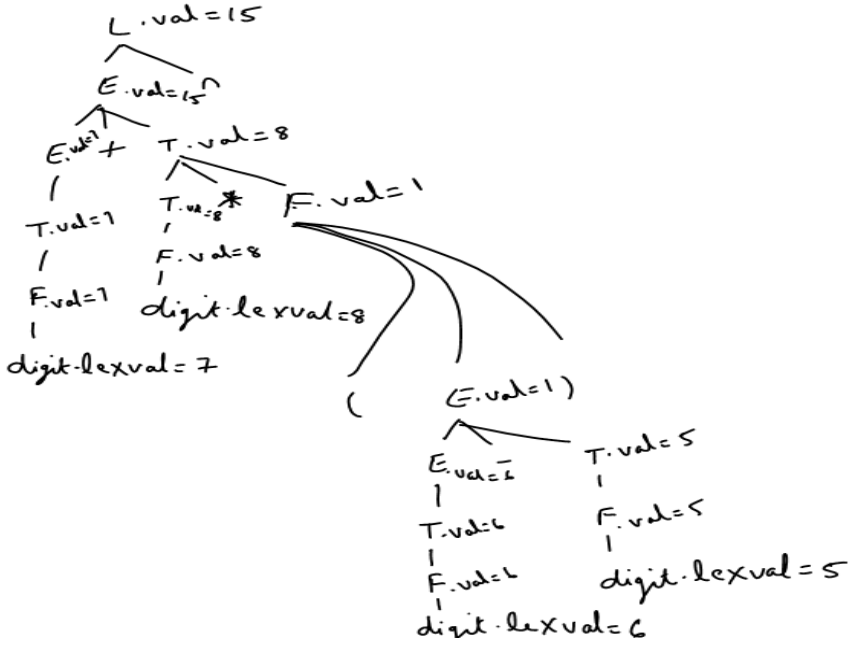
A syntax-directed definition (SDD) is a context-free grammar together with attributes and rules. Attributes are associated with grammar symbols and rules are associated with productions.

A syntax directed Translation (SDT): Syntax Directed Translation are augmented rules to the grammar that facilitate semantic analysis. SDT involves passing information bottom-up and/or top-down the parse tree in form of attributes attached to the nodes. Syntax directed translation rules use 1) lexical values of nodes, 2) constants & 3) attributes associated to the non-terminals in their definitions.

Production	Semantic Rules
L→En	L.val=E.val

E→E1+T	E.val=E1.val+T.val
E→T	E.val=T.val
T→T*F	T.val=T1.val*F.val
T→F	T.val=F.val
F→(E)	F.val=E.val
F→digit	F.val=digit.lexval

Annotated Parse Tree for the expression 7+8*(6-5)



5. **Explain the following with an example 1) quadruple 2)triple 3)indirect triple 4) single assignment form** [10] CO2 L2

Solution

1)Quadruple
A quadruple (or just \quad") has four fields: op, arg_1, arg_2, and result

Example:
a := b * -c + b * -c

#	*Op*	*Arg1*	*Arg2*	*Res*
(0)	uminus	c		t1
(1)	*	b	t1	t2
(2)	uminus	c		t3
(3)	*	b	t3	t4
(4)	+	t2	t4	t5
(5)	:=	t5		a

2)Triples
A triple has only three fields: op, arg_1, and arg_2.

#	*Op*	*Arg1*	*Arg2*
(0)	uminus	c	
(1)	*	b	(0)
(2)	uminus	c	
(3)	*	b	(2)
(4)	+	(1)	(3)
(5)	:=	a	(4)

3)Indirect Triples
Indirect triples consist of a listing of pointers to triples, rather than a listing of triples themselves

	Stmt
(0)	(14)
(1)	(15)
(2)	(16)
(3)	(17)
(4)	(18)
(5)	(19)

4)Single assignment Form
single-assignment form (SSA) is an intermediate representation that facilitates certain code optimizations.

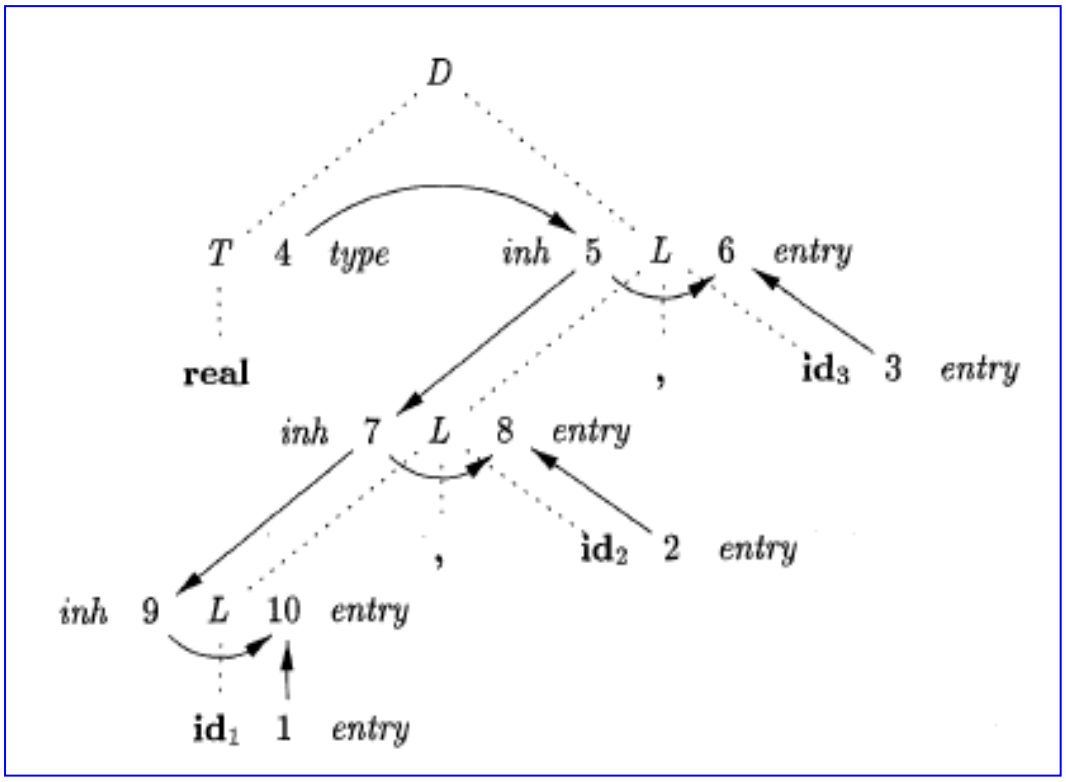
p=a+b
q=p-c
p=q*d
p=e-p
q=p+q

p1=a+b
q1=p1-c
p2=q1 *d
p3=e-p2
q2=p3 +q1

6. **a)Consider the grammar given in Q.2 and construct dependency graph for declaration float id1,id2,id3;** [5] | CO2 | L3

Solution

Production	**Semantic Rules**
D→TL	L.inh = T.type
T→int	T.type = integer
T→float	T.type = float
L→L,id	L1.inh = L.inh addType(id.entry; L.inh)
L→id	addType(id.entry; L.inh)



b) Define 1) Synthesis attribute 2) Inherited attribute with example [5] | CO2 | L1

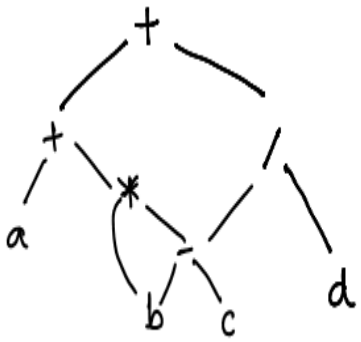
1. **synthesized attribute:** At a parse-tree node N is defined by a semantic rule associated with the production at N . Note that the production must have A as its head. A synthesized attribute at node N is defined only in terms of attribute values at the children of N and at N itself.

2. **Inherited attribute**: At a parse-tree node N is defined by a semantic rule associated with the production at the parent of N . An inherited attribute at node N is defined only in terms of attribute values at N 's parent, N itself, and N 's siblings.

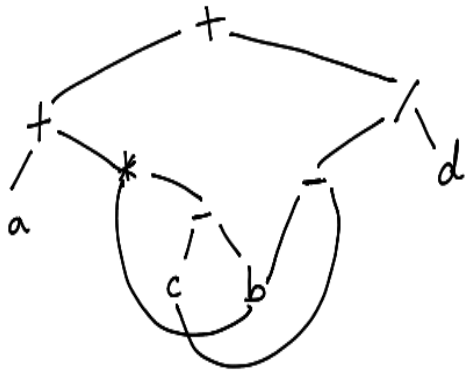
7. **a)Obtain DAG for the expression a + b *(b-c) +(b-c) / d and write the 3-address code for the constructed DAG.** [5] | CO2 | L3

Solution

a + b *(b-c) +(b-c) / d
T1=b-c
T2=b*T1
T3=T1/d
T4=a+T2
T5=T3+T4



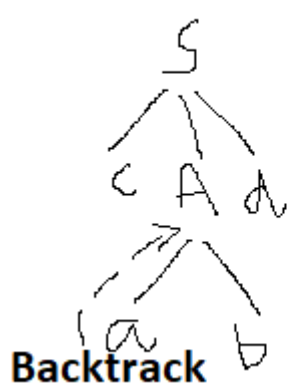
a + b *(c-b) +(b-c) / d
T1=c-b
T2=b*T1
T3=b-c
T4=T3/d
T5=a+T2
T6= T5+T4



b) What are the key problems with top down parsing? Write a recursive descent parser for the grammar: **S→ cAd ,** **A→ ab | a** for the input string **w=cad**. [5] | CO2 | L2

Problems with the Top-Down Parser

1. Problems with left-recursive rules.
2. Problems with back tracking



Recursive Decent parsing
S→cAd
A→ab|a

```
S()
{
   If(l==’c’)
    Match(‘c’);
  A();
If(l==’d’)
    Match(‘d’);
}
A()
{
If(l==’a’)
{
    Match(‘a’);
Else if(l==’b’)
Match(b);
}
Else (l==’a’)
    Match(‘a’);
}
Match(char t)
{
   If(l==t)
l = getchar();
    }
    else
        printf("Error");
}
Main()
{
S();
if (l == '$')
        printf("Parsing Successful");
}
```