* If any SDD has only synthesized attributes then the SDD is S-attributed (bottom-up approach)

DATE 06 05 2021

UNIT-IV

* Syntax Directed Definition (SDD) :- Same as that of CFG but with attributes & rules.
→ Attributes are associated with variables & Terminals
→ Rules are nothing but actions associated with productions

Eg:

Production	Semantic Rule
$E \rightarrow E_1 + T$	E.code ⇒ E_1.code || T.code || '+'

(E.code: attribute; ⇒: translation; ||: concatenation)

* Two types of attributes

⇒ Synthesized attribute:- The attribute value of Node N is associated with / defined only in terms of attribute values of children at node N and N itself.

⇒ Inherited attribute :- The attribute value of Node N is defined only in terms of the attribute values of its siblings, its parents, and itself.

Ex 1]

Production	Semantic Rules
$L \rightarrow E\,n$ (n → input-end marker (just like $)	L.val = E.val
$E \rightarrow E_1 + T$	E.val = E_1.val + T.val
$E \rightarrow T$ (T_1 is same as T)	E.val = T.val
$T \rightarrow T_1 * F$	T.val = T_1.val * F.val
$T \rightarrow F$	T.val = F.val
$F \rightarrow (E)$	F.val = E.val
$F \rightarrow$ digit	F.val = digit.lexval

This Grammar represents Desktop Calculator

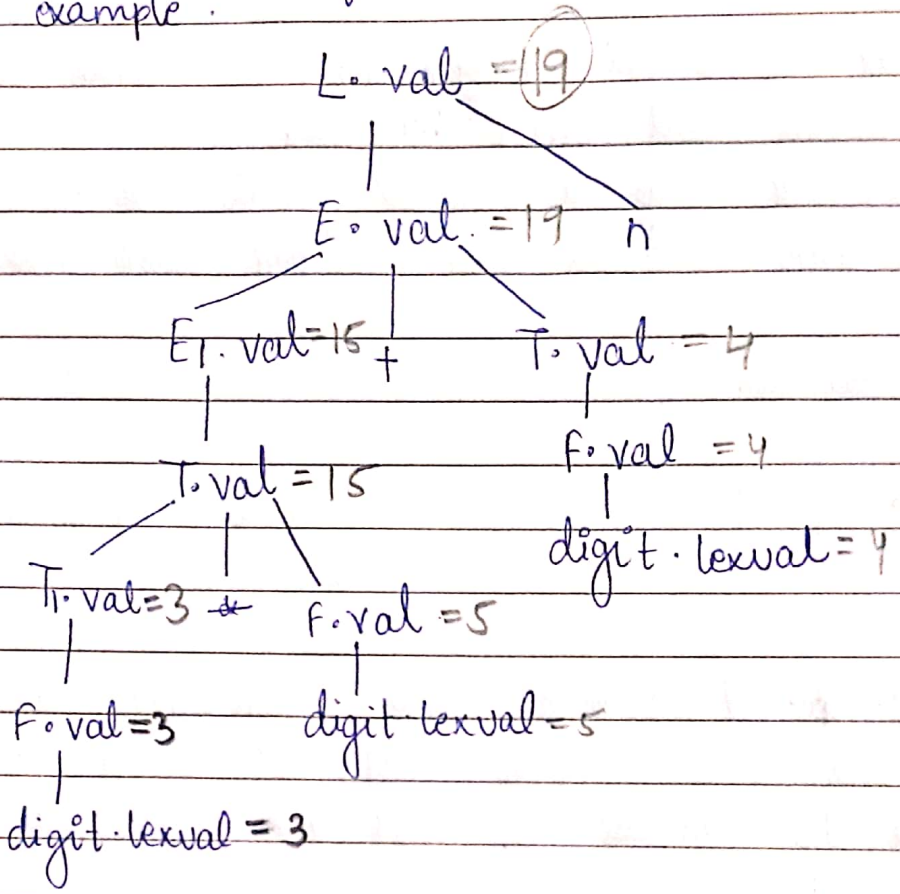
classmate

PAGE 001

DATE

i) Construct parse tree for input 3 * 5 + 4n, using the previous example.

Soln:



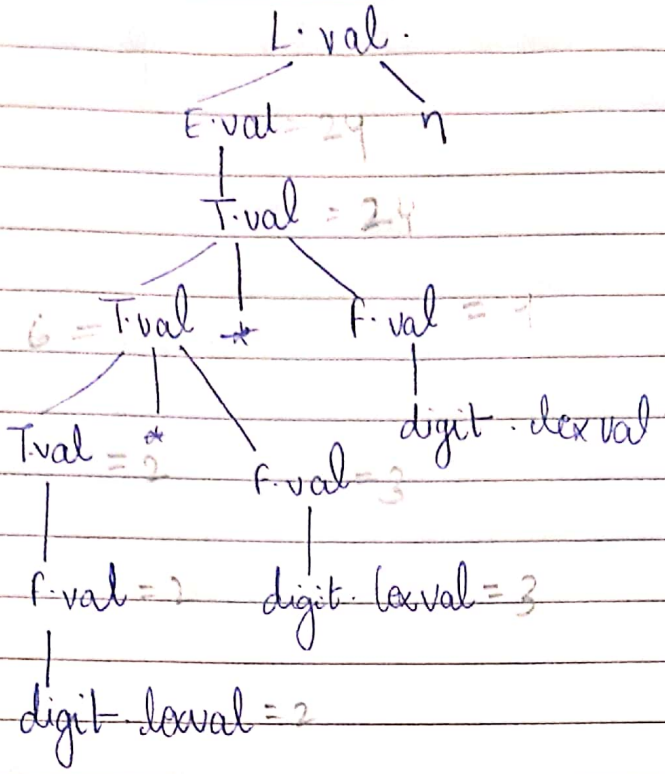
The above tree is Annotated Parse Tree.

→ If each node contains the values of attribute, then the parse tree is called "Annotated parse tree"

→ Here the value of each node is computed as the value of its children or that node itself, and hence its "Synthesized attribute".

ii) Construct parse tree for input 2 * 3 * 4, using previous example. It should be annotated parse tree.

Soln



→ This SDD is S-attributed.

[Ex 2]

	Production	Semantic rule
1)	$T \rightarrow FT'$	$T'.inh = F.val$ $T.val = T'.syn$
2)	$T' \rightarrow *FT_1'$	$T_1'.inh = T'.inh \times F.val$ $T'.syn = T_1'.syn$
3)	$T' \rightarrow \epsilon$	$T'.syn = T'.inh$
4)	$F \rightarrow digit$	$F.val = digit.lexval$

val → synthesized
inh → inherited.

This SDD has both synthesized & inherited attribute

DATE

i) Construct parse tree for the input 3 * 5, using previous example. (both top down & bottom

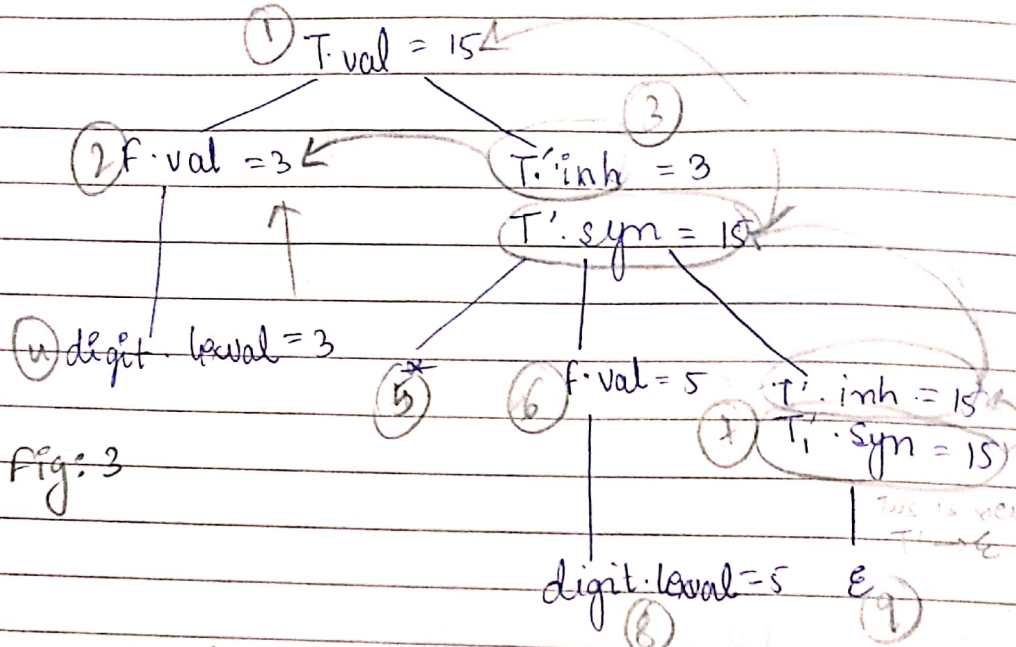
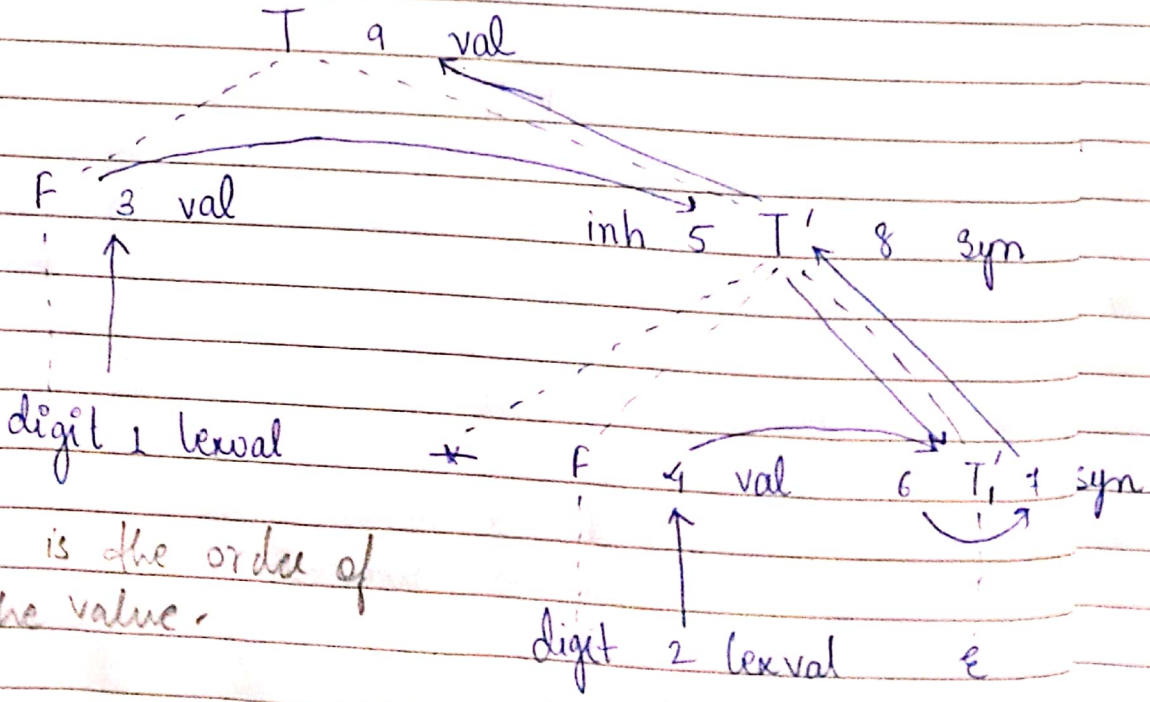


Fig: 3

* Dependancy Graph: Depicts the flow of information of attribute in a parse tree (Evaluation order for SDD)

ii) Construct Dependency graph for the above example



1, 2, 3 ... 9 is the order of assigning the value.

classmate

PAGE 004

DATE

* L-attributed SDD (dependency-graph edges can go from left to right, but not from right to left, hence L-attributed).

→ The desktop calculator example is both S-attributed and L-attributed.

→ Check whether the following example SDD/Grammar is L-attributed

1)

Production	Semantic Rules
i) $T \rightarrow FT'$	$T'.inh = F.val$ $T.val = T'.syn$
ii) $T' \rightarrow *FT_1'$	$T_1'.inh = T'.inh \times F.val$ $T'.syn = T_1'.syn$
iii) $T' \rightarrow \varepsilon$	$T'.syn = T'.inh$
iv) $F \rightarrow digit$	$F.val = digit.lexval$

There are inherited attribute → check whether these follows the three rules for L-attributed SDD.

Sol:- See fig: 3, ⇒ T'.inh = 3 which is taken/dependent on the value of F.val. Here the dependency Graph is satisfied. So.

⇒ T'.inh = 15 is evaluated from node (6) & (3) which is it's sibling and parent respectively. Also, the dependency graph is from left to right and is hence L-attributed.

∴ The Grammar L-attributed.

syn

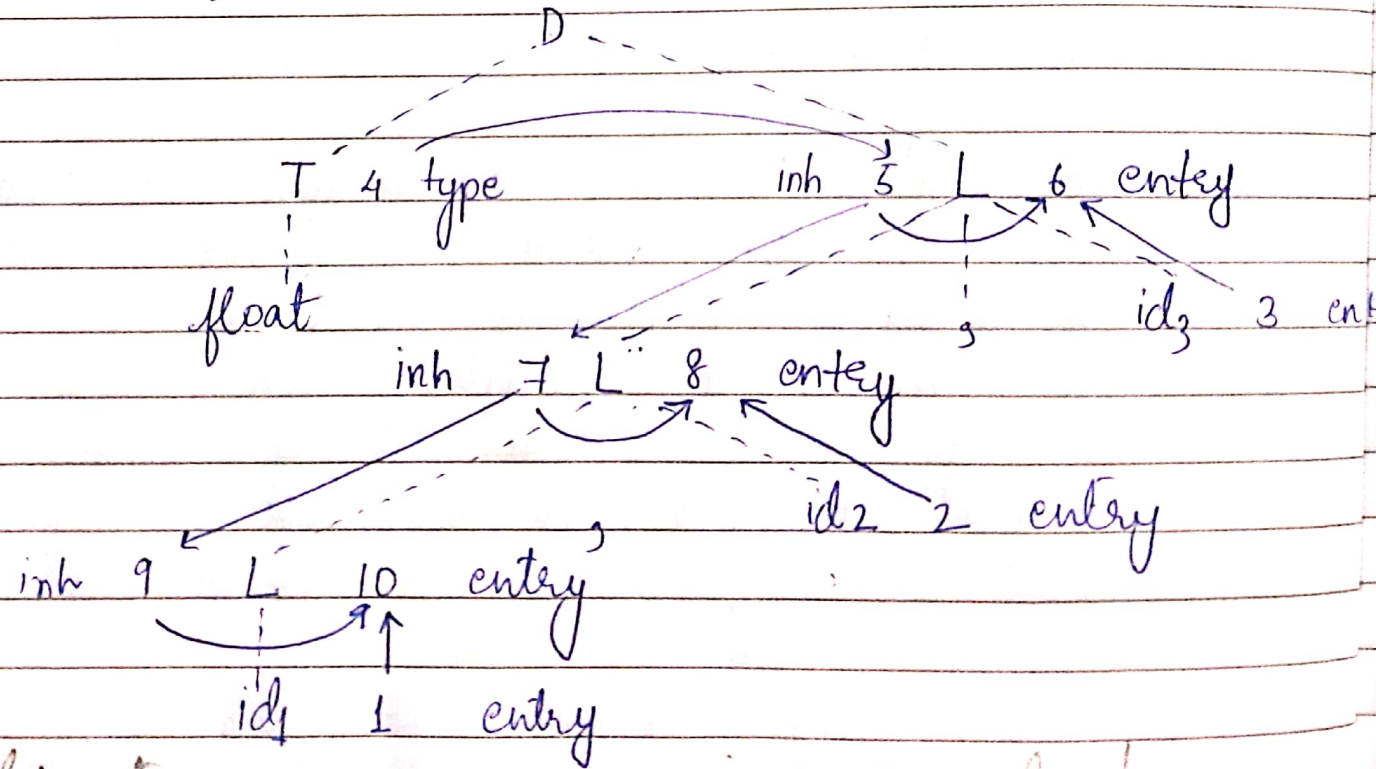
DATE

2) SDD for type checking,

Production	Semantic Rules
$D \rightarrow TL$	$L.inh = T.type$
$T \rightarrow int$	$T.type = integer$
$T \rightarrow float$	$T.type = float$
$L \rightarrow L_1, id$	$L_1.inh = L.inh$ addType (id.entry, L.inh) ← add to Symbol table
$L \rightarrow id$	addType (id.entry, L.inh)

Input: float id_1, id_2, id_3

Soln: Dependency Graph:



Explanation same as previous example for L-attributed

classmate

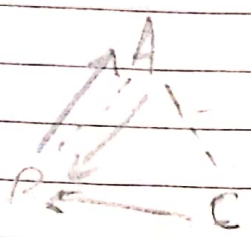
PAGE 006

DATE

3)

Production	Semantic Rules
$A \rightarrow BC$	$A.syn = B.b$ $B.inh = f(C.c, A.s)$

Soln:



Here the dependency of B is on C i.e from right to left, so hence it is L-attributed.

4) Take the example on Pg 003, and construct a parse tree and depedency graph for the input 4 * 5 * 6.

Soln:

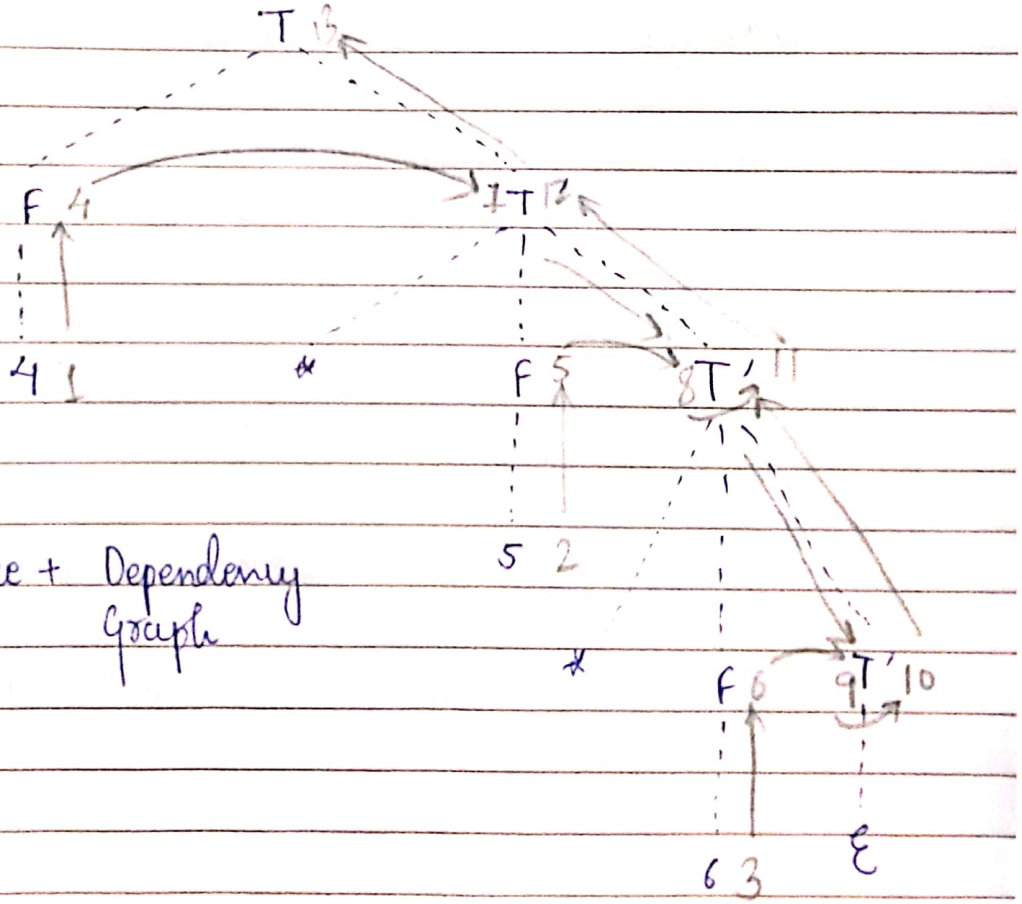


Fig: Parse tree + Dependency Graph

→ It's bottom up approach when only synthesized is used otherwise top-down when synthesized & inherited

DATE 10/05/2021

* Applications of Syntax Directed Translation (SDT)

1. Construction of syntax trees (syntax tree are used to derive intermediate representation)

For eg:- i) $E \rightarrow E+T$ ii) $E+T$

⇒ Each node in syntax tree is treated as object and are viewed as records (see in textbook regarding leaf node & interior node)

Example 1 : Simple Expression (Syntax trees only for synthesized attributes).

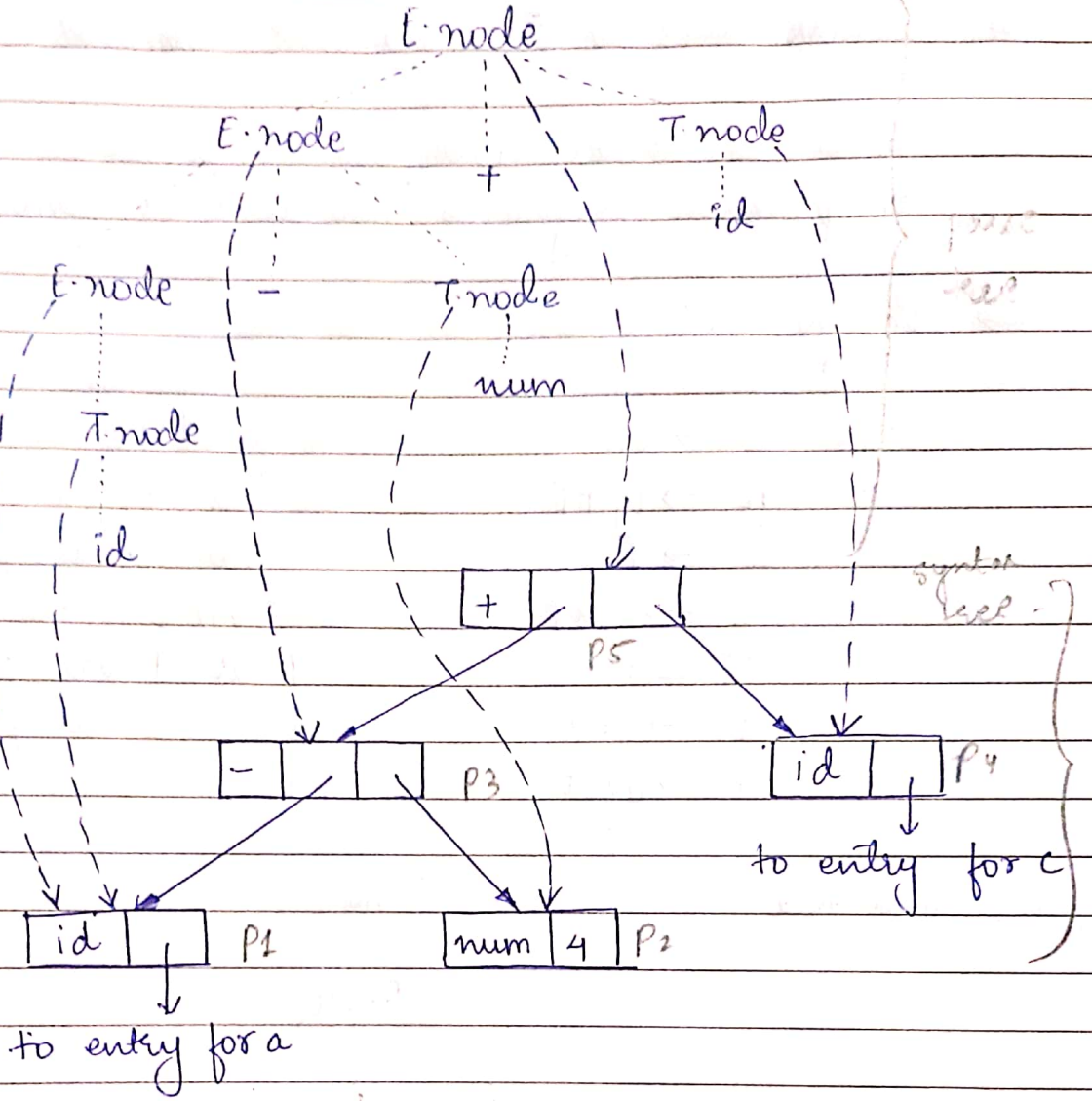
i) Construct syntax tree using the following Grammar for the input a-4+c

Production	Semantic Rules
1) $E \rightarrow E_1 + T$	E.node = new Node('+', E_1.node, T.node)
2) $E \rightarrow E_1 - T$	E.node = new Node('-', E_1.node, T.node)
3) $E \rightarrow T$	E.node = T.node
4) $T \rightarrow (E)$	T.node = E.node
5) $T \rightarrow id$	T.node = new Leaf(id, id.entry)
6) $T \rightarrow num$	T.node = new Leaf(num, num.val)

In the exam only production will be given, semantic rule should be written on our own.

DATE

Soln:



* Steps for the above syntax tree :-

1) p1 = new Leaf (id, entry·a);
2) p2 = new Leaf (num, 4);
3) p3 = new Node ('−', p1, p2);
4) p4 = new Leaf (id, entry·c);
5) p5 = new Node ('+', p3, p4);

} follows bottom up approach

ii) Construct syntax tree for (a+b+c−d) for the above grammar (Soln is in the next page)

DATE

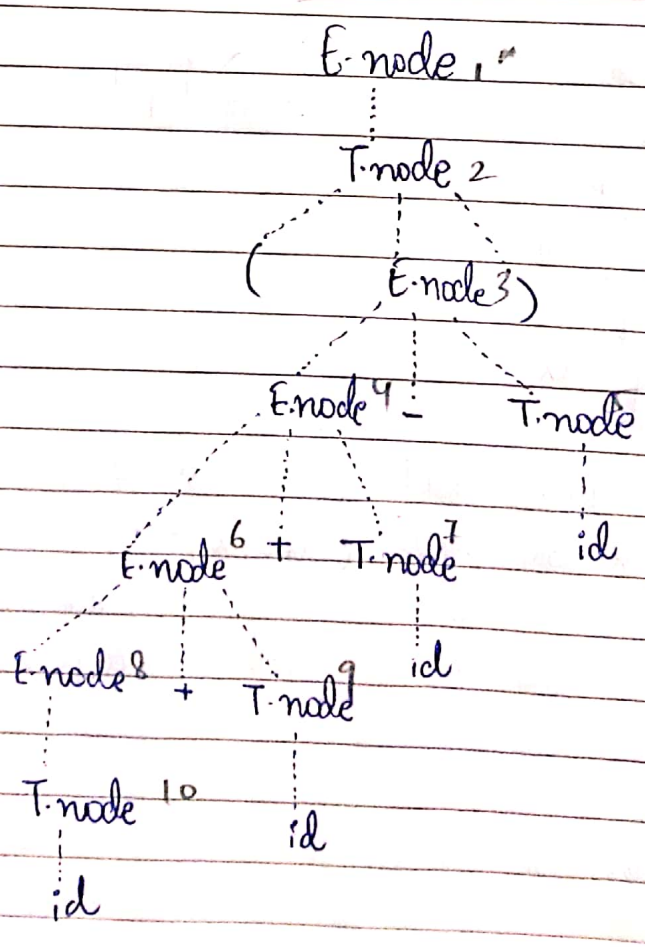
Syntax directed Translational Schemes (SDT)

⇒ Same as Context Free Grammar with the addition of program fragments (also called as semantic actions).

⇒ **Postfix SDD**: Defⁿ in TB

Eg:-	Productions	Semantic Actions
	$L \rightarrow En$	{ print (E.val); }
	$E \rightarrow E_1 + T$	{ E.val = E_1.val + T.val; }
	$E \rightarrow T$	{ E.val = T.val; }
	$T \rightarrow T_1 * F$	{ T.val = T_1.val × F.val; }
	$T \rightarrow F$	{ T.val = F.val; }
	$F \rightarrow (E)$	{ F.val = E.val; }
	$F \rightarrow digit$	{ F.val = digit.lexval; }

Solu: Parse tree: (for ii)

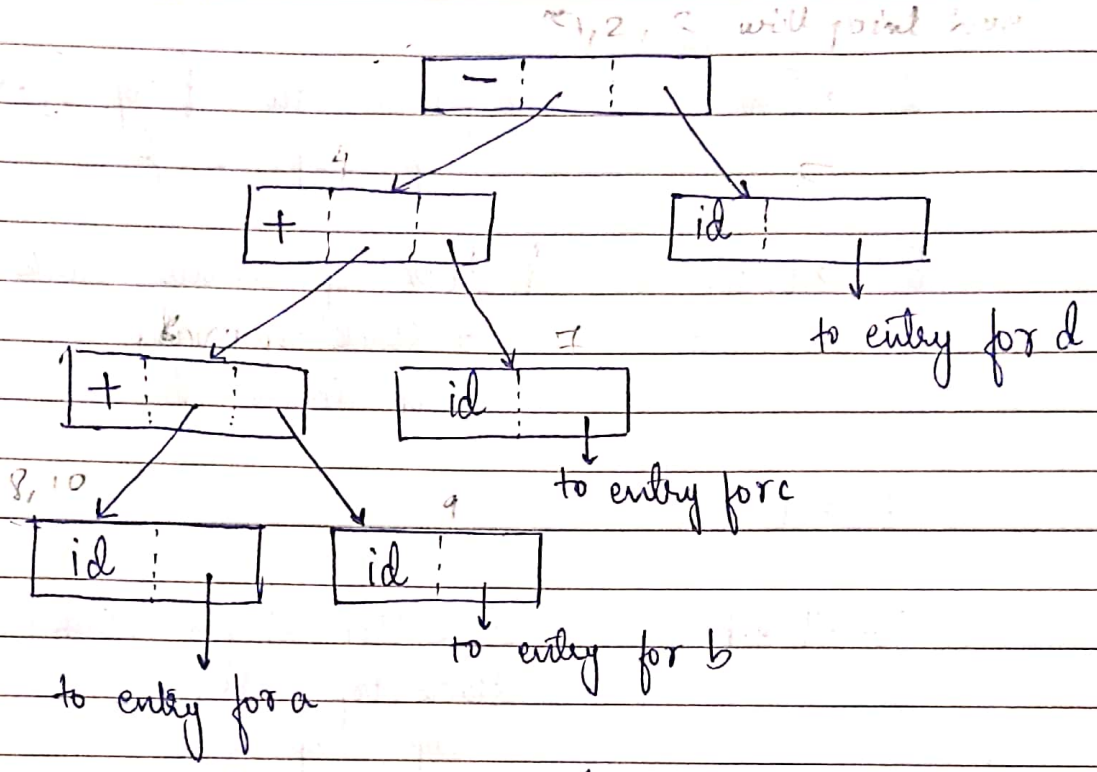


classmate

PAGE

DATE 11/05/2021

Syntax tree:



⇒ Steps for above syntax tree:

1. p1 = new Leaf (id, entry·a);
2. p2 = new Leaf (id, entry·b);
3. p3 = new Node ('+', p1, p2);
4. p4 = new Leaf (id, entry·c)
5. p5 = new Node ('+', p3, p4);
6. p6 = new Leaf (id, entry·d);
7. p7 = new Node ('-', p5, p6);

Postfix SDD (contd. ...)

X	Y	Z	State / Grammar symbol
X·x	Y·y	Z·z	Synthesized attributes

top

DATE

Production	Actions
$L \rightarrow En$	{ print (stack [top-1] . val); top = top - 1; }
$E \rightarrow E_1 + T$	{ stack [top-2] . val = stack [top-2] . val + stack [top] . val; top = top - 2; }
$E \rightarrow T$	
$T \rightarrow T_1 * F$	{ stack [top-2] . val = stack [top-2] . val x stack [top] . val; top = top - 2; }
$T \rightarrow F$	
$F \rightarrow (E)$	{ stack [top-2] . val = stack [top-1] . val; top = top - 2; }
$F \rightarrow$ digit	

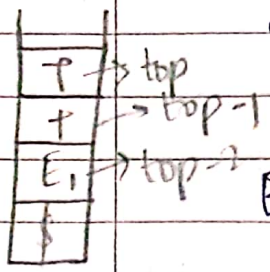


fig: Parser - stack Implementation of Postfix SDD.