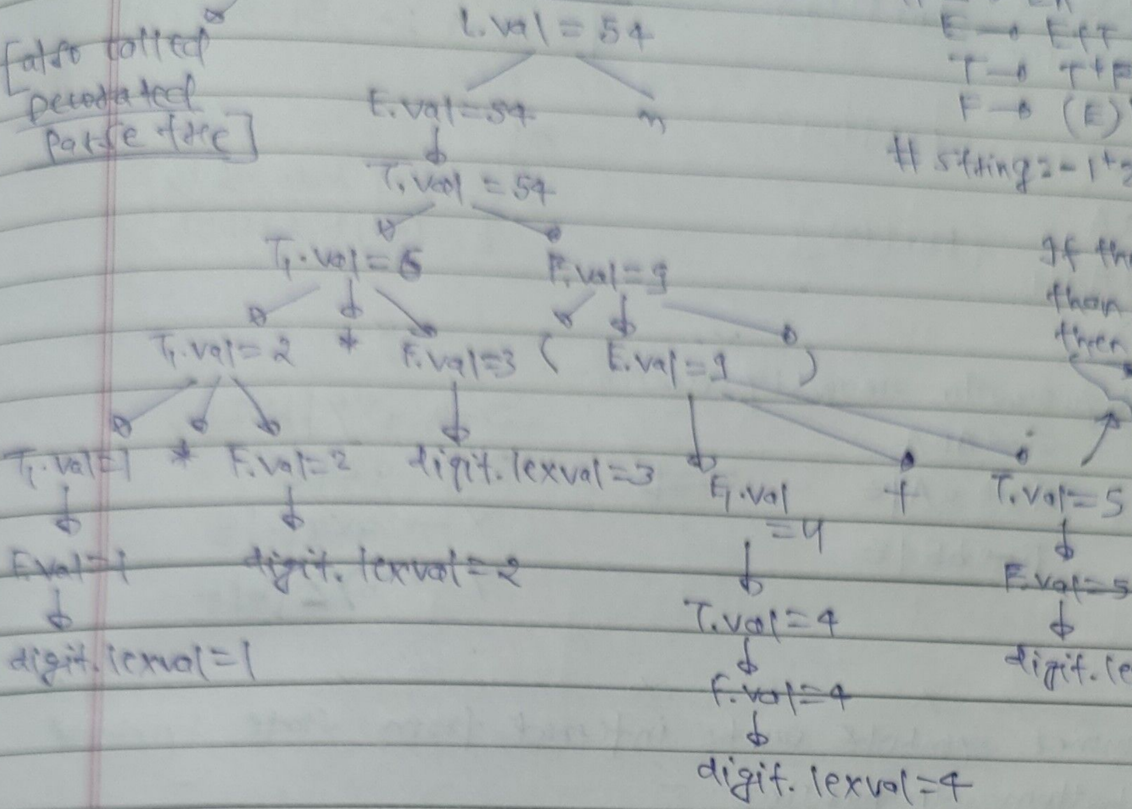
MTE Practice

Annotated Parse tree :-

[also called decorated parse tree]

- # $L \to En$
  - $E \to E+T \mid T$
  - $T \to T*F \mid F$
  - $F \to (E) \mid digit$
- # string :- $1*2*3*(4+5)$



If there are more than 1 attr, then :

T.x = 5
T.c = 7

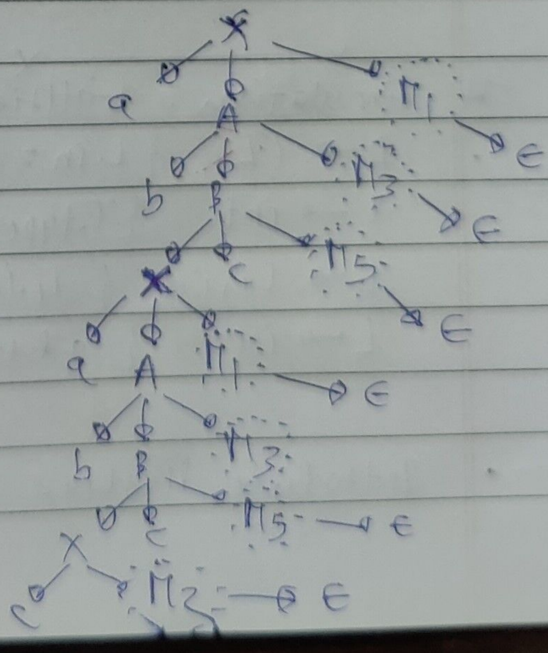
Bottom up parsing :-

- $X \to aAM_1 \mid cM_2$
  - $A \to bBM_3 \mid aM_4$
  - $B \to xcM_5 \mid cM_2$
  - $M_1 \to \epsilon$ {print(3)}
  - $M_2 \to \epsilon$ {print(5)}
  - $M_3 \to \epsilon$ {print(2)}
  - $M_4 \to \epsilon$ {print(4)}
  - $M_5 \to \epsilon$ {print(1)}

Introduce Marker symbols and make rightmost derivation of string to be parsed.

Bottom-up will be in reverse

- ababccc
  - ababcM₂cc {print(5)}
  - ababXcc
  - ababXcM₅c {print(1)}
  - ababBc
  - ababBM₃c {print(2)}
  - abaAc
  - abaAM₁c {print(3)}



PAGE No.
DATE

abX⊂ {print(1)}
abX⊂M5
abB
abBM3 {print(2)}
aA
aAM1 {print(5)}
X

→ {string reduced to start-symbol}

Output :- 51235123

Ans

Ambiguous to unambiguous :-

S → AFA
A → aA | ∈
B → bB | ∈

⟹

S → aXY | bYZ | ∈
Z → aZ | ∈
X → aXY | a | ∈
Y → bXZ | b | ∈

Grammer symbols can be inferred from state symbols in bottom-up parsing :-

- state stack ⇒ 0, 2, 4, 7 {At some point of time, the config. of state stack}.
  - From state 0 to 2, we have 'T' as the transition
  - From 2 to 4, '*'
  - From 4 to 7, 'F'

Then state 2 will correspond to 'T'
4 will be '*'
7 will be 'F'

Simulation : L-attributed + LR-parsing (bottom-up) :-

- D → TL {L.in = T.type}
  T → int {T.type = int}
  L → L1, id {L1.in = L.in; addtype(id.entry, L.in);}
  L → id {addtype(id.entry, L.in);}

- Introduce Markers and form SDT :-

PAGE No.
DATE

$D \rightarrow T\ M_1\ L$

$T \rightarrow int$

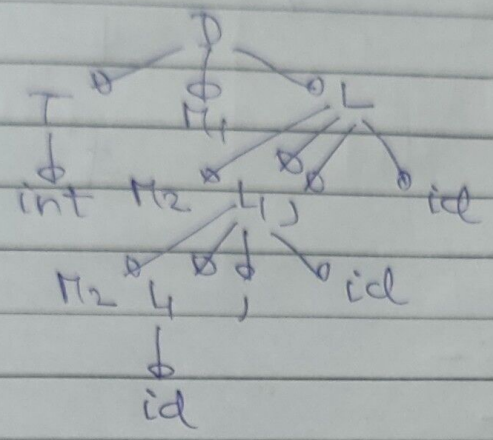
$L \rightarrow M_2\ L_1, id \quad \{addtype(id.entry, L.in);\}$

$L \rightarrow id \quad \{addtype(id.entry, L.in);\}$

$M_1 \rightarrow \in \quad \{M_1.i = T.type,\ M_1.s = M_1.i\}$

$M_2 \rightarrow \in \quad \{M_2.i = L.inh,\ M_2.s = M_2.i\}$

- For simulation, do rightmost derivation of the string :-



$\equiv$ int id, id, id

$\equiv$ T M_1 M_2 M_2 id, id, id

stack	input
$	int id, id, id $
$ int	id, id, id $
$ T	id, id, id $
$ T M_1	id, id, id $
$ T M_1 M_2	id, id, id $
$ T M_1 M_2 M_2	id, id, id $
$ T M_1 M_2 M_2 id	, id, id $
$ T M_1 M_2 M_2 id ,	id, id $
$ T M_1 M_2 M_2 id , id	, id $
$ T M_1 M_2 M_2 id , id ,	id $
$ T M_1 M_2 M_2 id , id , id	$

[Symbol stack] [attribute or value stack] [undefined]

M_i will take value from top-1 always

PAGE No. DATE

[from "top"]

$ T M1 M2 M2 L ,	id, id $	{ addtype(id.entry, L.in) }
- int int int int int -		[from "top-1"]
$ T M1 M2 M2 L , id	, id $	
- int int int int int - id.entry		reduce it {id.entry at top and L.in will be at top-4}
$ T M1 M2 L	, id $	
- int int int int		
$ T M1 M2 L , id	$	
- int int int int - id.entry		
$ T M1 L	$	
- int int int		
$ D	$	⇒ [String reduced to start symbol]
- int		

Grammar will be :-

[use synthesized here]

D → T M1 L

I → int {addtype(value[top].entry, value[top-1].in)}

L → M2 L , id {addtype((value[top].entry), value[top-4].in)}

L → id {addtype(value[top].entry, value[top-1].in)}

M1 → ∈ { value[top].in = value[top-1].type; value[top].syn = value[top].inh }

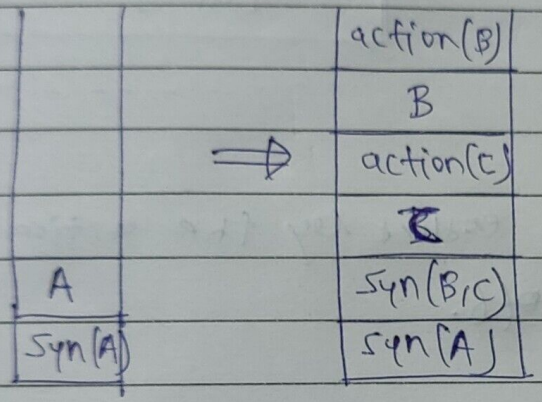
M2 → ∈ { value[top].in = value[top-1].syn; value[top].syn = value[top].inh }

PAGE No.
DATE

L-attributed + LL-Grammar (Top-down parsing):-

- "action records" will have pointer to actions
  - these actions will copy the value in the bottom of the stack and execute some print statements or side effects above the symbol.
- "synthesized Records"
  - will contain synthesized attribute.
  - these will be present with the symbol itself. (just below it)

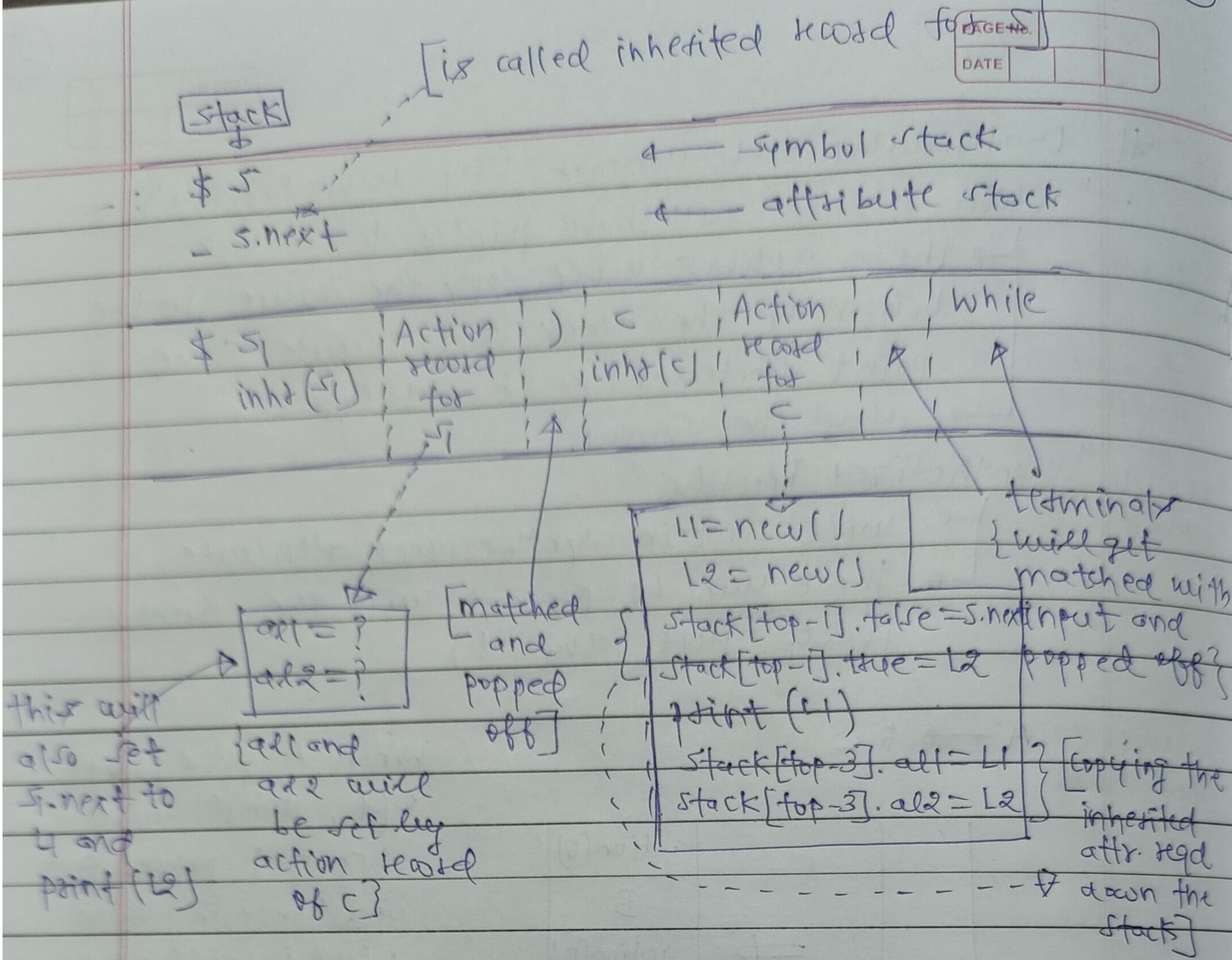
- A → B C



When 'A' gets popped off, inh(A) will also get popped out and hence the value of A.i will have copied to all the locations where it is required.
- action(B), action(C)

- S → while ({ L1 = newl()
  L2 = newl()
  C.false = S.next
  C.true = L2
  print (label L1) }
  C) { S_1.next = L1;
  print(label L2); }
  S_1

- {L1 and L2 are labels for conditional jumps.}



S.next will be set earlier by the action record of S before popping off.

S-attributed + LR grammar → Bottom-up parsing
L-attributed + LL grammar → top-down parsing
L-attributed + LR grammar → No parsing
↳ We can't do syntax direction translation along with parsing here.

	LR	LL
S-attr.	B	B/T
L-attr.	X	B/T