Syntax-Directed Translation

- ✓ What is SDD?
- ✓ Types of attributes
- ✓ Types of SDD
- ✓ Examples (SDD for evaluation of expression ~~grammar~~, simple type declarations)

1 What is Syntax-Directed definition (SDD)?

[SDD = CFG + Attributes Semantic Rules]

→ A SDD is a Context free Grammar together with attributes and Semantic Rules.

→ Attributes are associated with grammar symbols and Semantic rules are associated with production.

→ If 'X' is a symbol and 'a' is one of its attribute then X.a denotes value at node 'X'.

→ Attributes may be numbers, strings, references, datatypes.

~~For eg.~~ Example 1

Productions	Semantic Rules
$E \rightarrow E + T$	$E.val = E.val + T.val$
$E \rightarrow T$	$E.val = T.val$

→ E, T are grammar symbols

→ val is an attribute

→ E.val or T.val means attribute (val) is associated with grammar symbol (E or T).

→ Every symbol must contain an attribute.

Semantic Rules: It provides meaning to the corresponding production

② Types of Attributes in SDD

① **Synthesized Attribute** If a node takes value from its children then it is a synthesized attribute.

Ex:

$A \rightarrow BCD$, A be a parent node
B, C, D are children nodes.

$$\left.\begin{array}{l} A.S = B.S \\ A.S = C.S \\ A.S = D.S \end{array}\right\}$$

parent node A taking value from its children B, C, D.

② **Inherited Attribute** If a node takes value from its parent or siblings.

Ex:

$A \rightarrow BCD$

$C.i = A.i$ → parent node
$C.i = B.i$ → sibling node
$C.i = D.i$ → sibling node

3 Types of SDD

1 S- Attributed SDD
or
S- Attributed definitions
or
S- Attributed Grammar

→ A SDD that ~~uses~~ uses only synthesized attributes is called as S-Attributed SDD.

Ex: $A \rightarrow BCD$

$A.S = B.S$

$A.S = C.S$

$A.S = D.S$

→ Semantic ~~Synthesized~~ actions are always placed at right end of the production. It is also called as "postfix SDD".

→ Attributes are evaluated with Bottom-up parser.

(2) L-Attributed SDD

(or)

L-Attributed definitions

(or)

L-Attributed Grammar

→ A SDD that uses both synthesized & inherited attributes is called as L-Attributed SDD but each inherited attribute is restricted to inherit from parent or left sibling only.

Ex $A \rightarrow XYZ$

$Y.S = A.S$ ✓

$Y.S = X.S$ ✓

$Y.S = Z.S$ ✗

→ Semantic actions are placed anywhere on R.H.S.

→ Attributes are evaluated by traversing parse tree depth first, left to right order. (top-down, left to right order)

Q. Write the ↓

$\boxed{\#1}$ Syntax-Directed definition of simple desk calculator

or

SDD for evaluation of expression

or

Annotated parse tree for 3 * 5 + 4n

Expression Grammar ↓ Example ① ↓

Productions	Semantic Rule
$L \rightarrow En$	$L.val = E.val$
$E \rightarrow E_1 + T$	$E.val = E_1.val + T.val$
$E \rightarrow T$	$E.val = T.val$
$T \rightarrow T_1 * F$	$T.val = T_1.val * F.val$
$T \rightarrow F$	$T.val = F.val$
$F \rightarrow digit$	$F.val = digit.lexval$

↑ A symbol generated by the lexical analyzer.

En ~~new line~~ n is like new line marker / ends with a new line marker

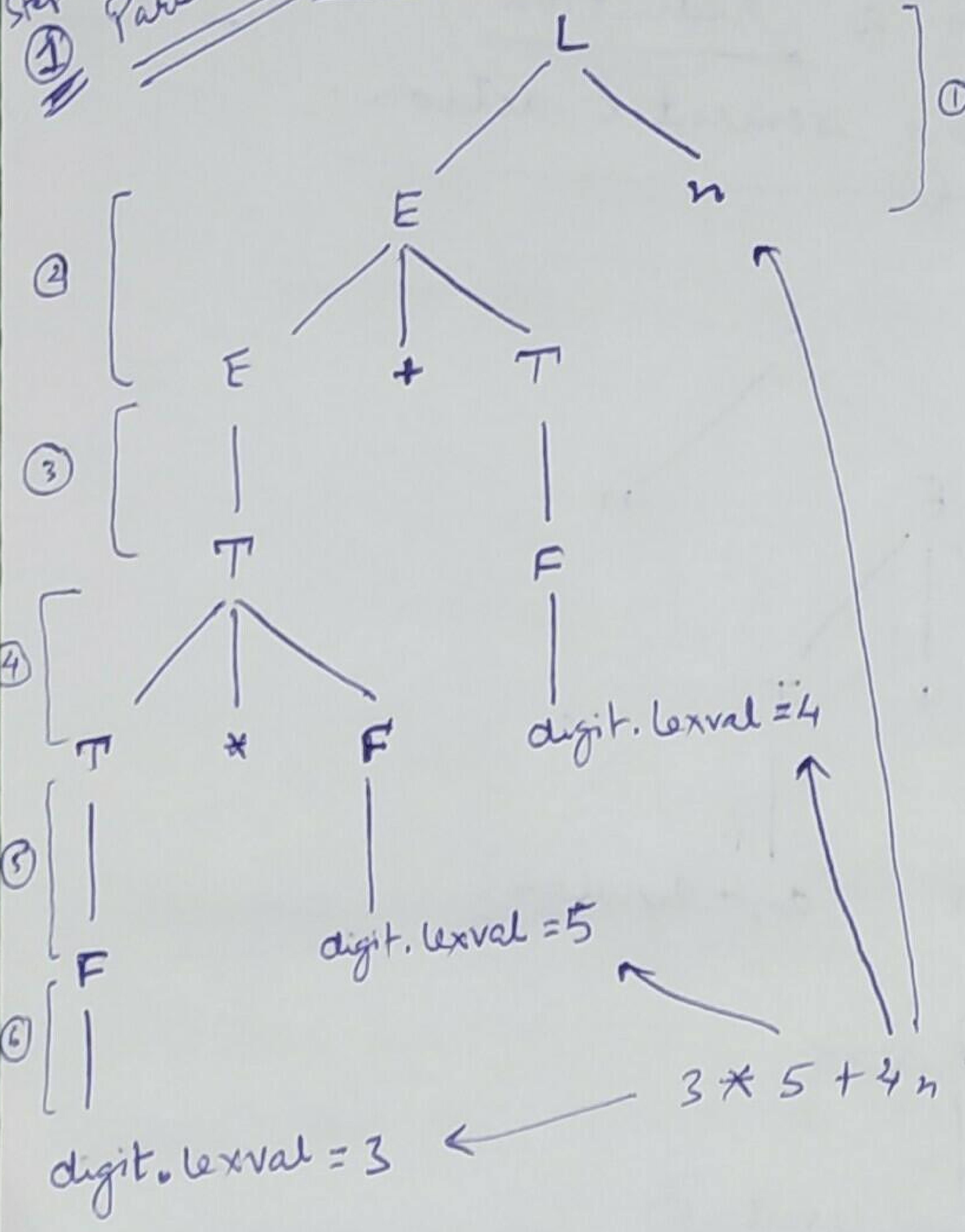
To solve this problem ←

step ① Construct Parse tree (depending upon the input string)

step ② then ~~state~~ create Annotated Parse tree

↳ contains values at each node

step ① Parse Tree $(3 * 5 + 4n)$



Derivation or construction of Parse tree starts from start symbol.

→ // Directly with the help of E we can't perform 3×5. There is no production as E * ? (or multiplication operation)

3 * 5 + 4 n

↑ ↑ ↑

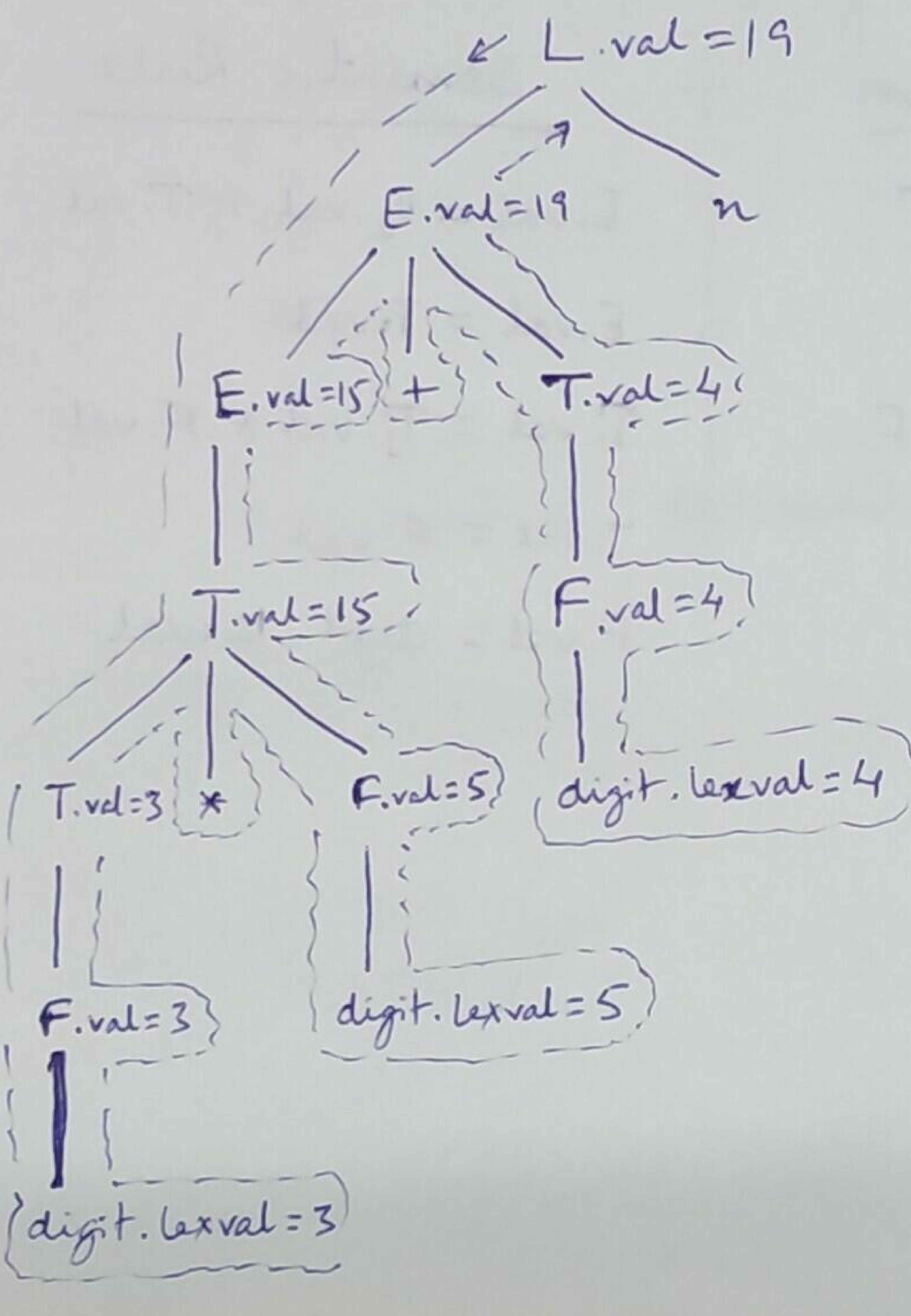
D_1 D_2 D_3

step 2 ~~Make~~ Create Annotated Parse Tree

→ In order to construct APT we have to

#1 perform top-down, left-to-right traversing.

#2 If there is a reduction, then execute corresponding semantic action.



Example (2)

#2 SDD of a Simple Desk Calculator

or

SDD for evaluation of expressions

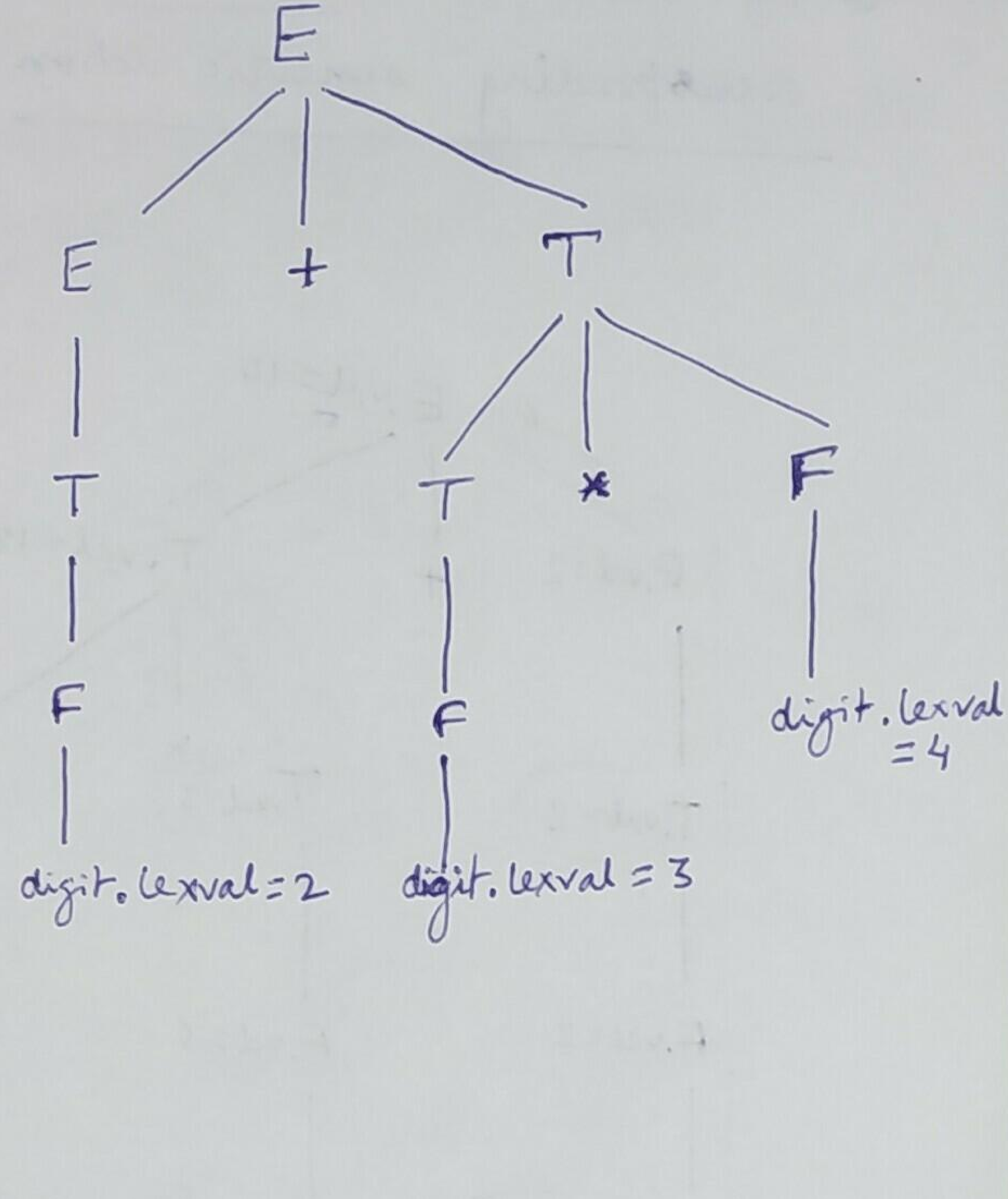
or

Annotated Parse Tree for 2+3*4

Expression Grammar ↓

Productions	Semantic Rule
$E \rightarrow E_1 + T$	$E.val = E_1.val + T.val$
$E \rightarrow T$	$E.val = T.val$
$T \rightarrow T_1 * F$	$T.val = T_1.val * F.val$
$T \rightarrow F$	$T.val = F.val$
$F \rightarrow digit$	$F.val = digit.lexval$

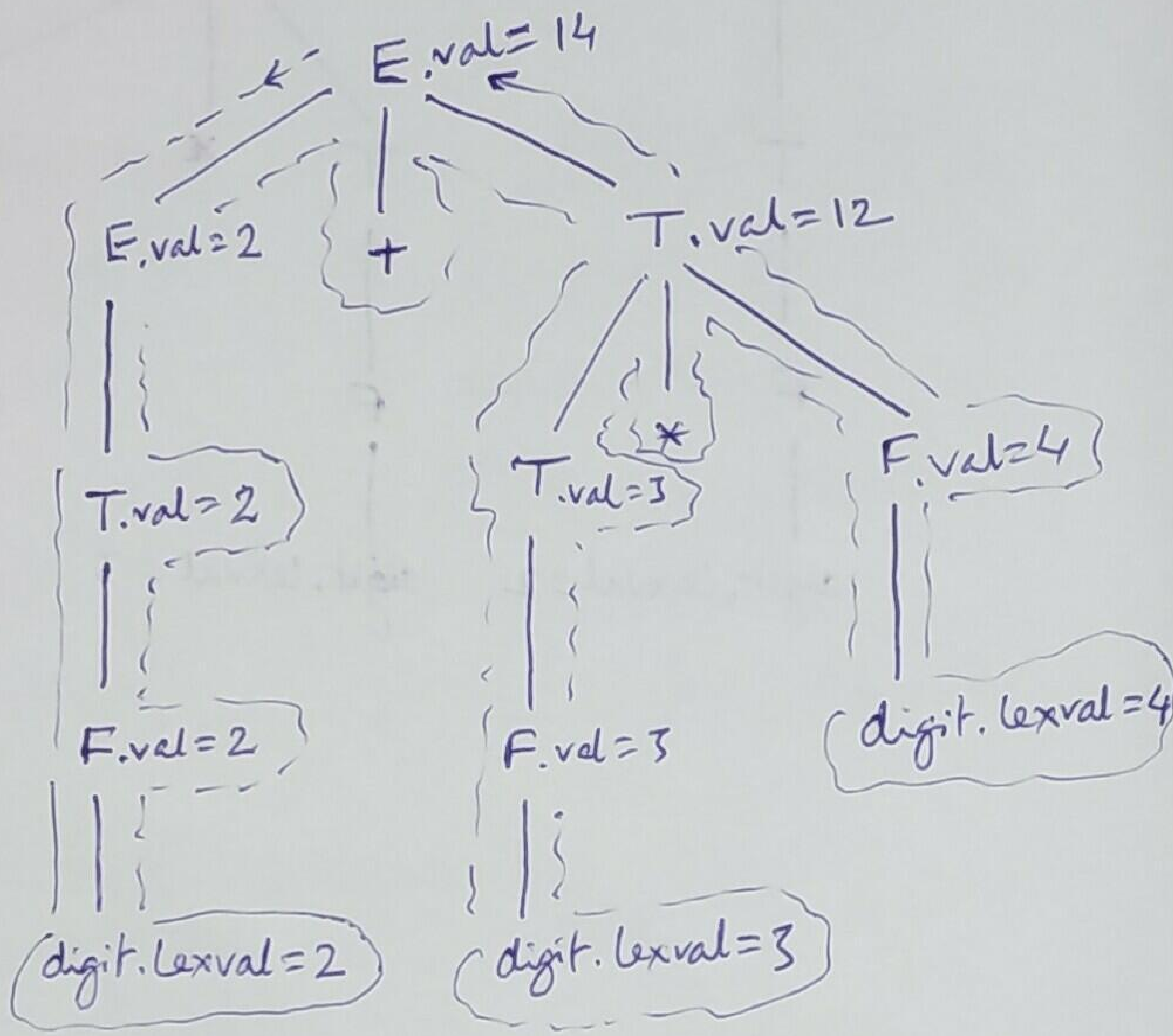
step 1 Constanct Parse Tree input string (2 + 3 * 4)



Step 2 ~~Make~~ create the annotated Parse Tree

① Top-down, left-to-right traversing

② If there is a reduction, then execute corresponding semantic action.



Example 3

#3 SDD for simple type declarations

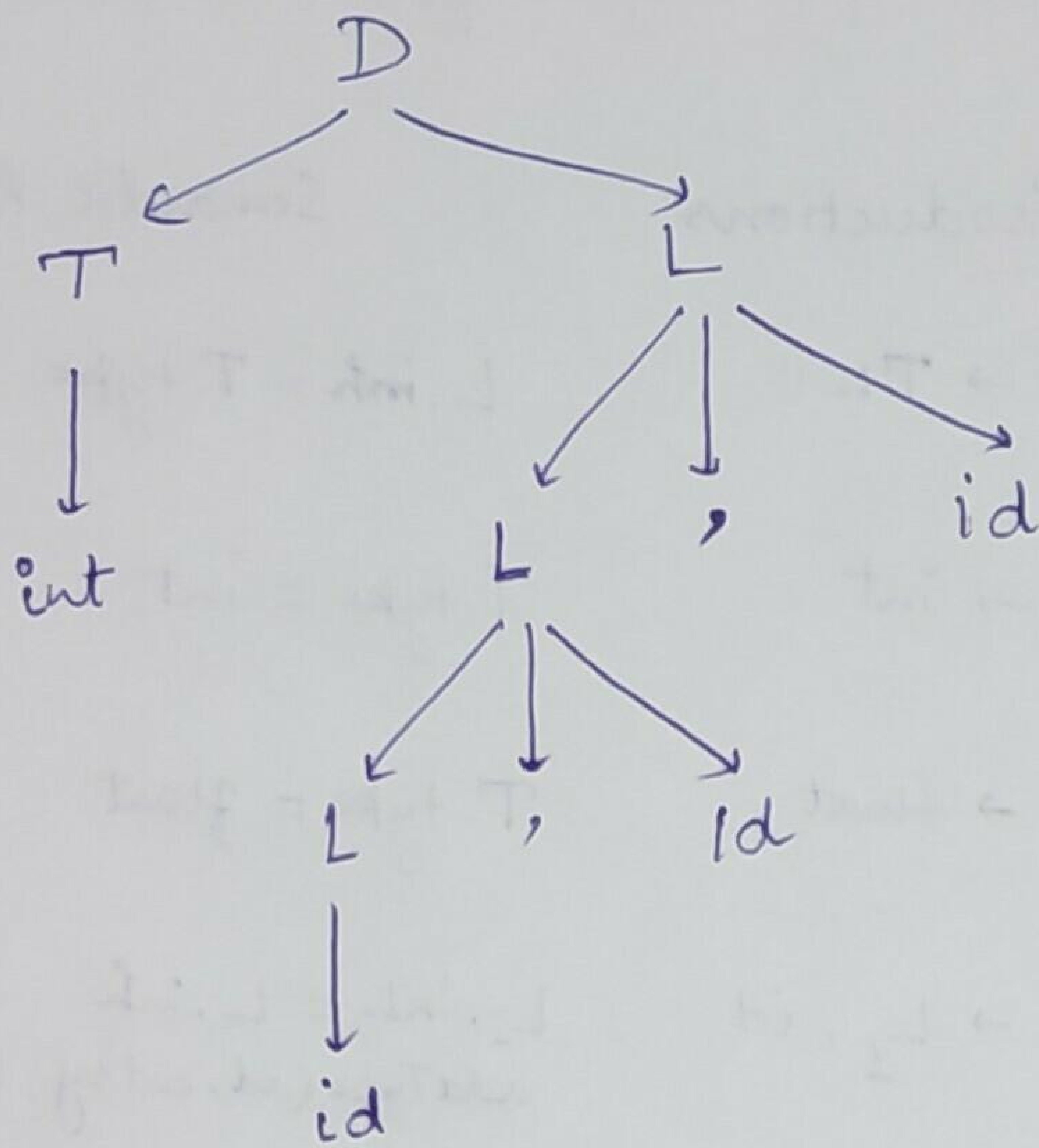
Productions	Semantic Rules
1. $D \rightarrow TL$	$L.inh = T.type$
2. $T \rightarrow int$	$T.type = int$
3. $T \rightarrow float$	$T.type = float$
4. $L \rightarrow L_1, id$	$L_1.inh = L.inh$ addType(id.entry, L.inh)
5. $L \rightarrow id$	addType(id.entry, L.inh)

Q: Construct Annotated Parse Tree for int a, b, c

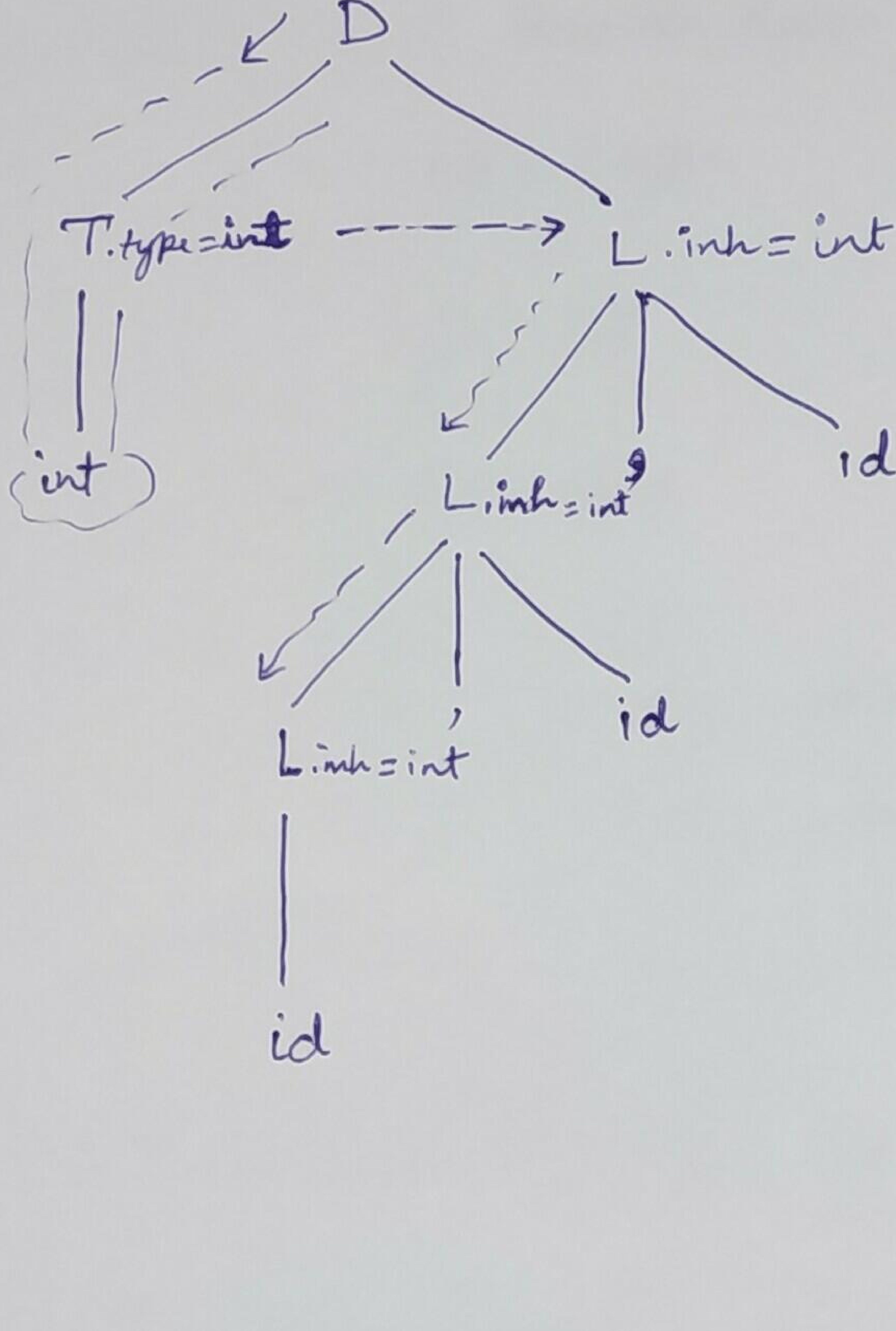
\# inh → Inherited attribute, node taking value from its parent or sibling

\# syn → Synthesized attribute, node takes value from its children nodes

Step 1 Construct Parse Tree (int a, b, c)



Step 2 Create the annotated Parse Tree



Dependency Graphs

→ It represents the flow of information among the attributes in a parse tree.

→ It is useful for determining the evaluation order for attributes in a parse tree.

→ While an Annotated parse tree shows the values of Attributes, a Dependency Graph (DG) determines how those values can be computed.

Expression Grammar

Productions	Semantic Rules
$E \rightarrow E + T$	$E.val = E_1.val + T.val$
$E \rightarrow T$	$E.val = T.val$
$T \rightarrow T * F$	$T.val = T.val * F.val$
$T \rightarrow F$	$T.val = F.val$
$F \rightarrow digit$	$F.val = digit.lexval$

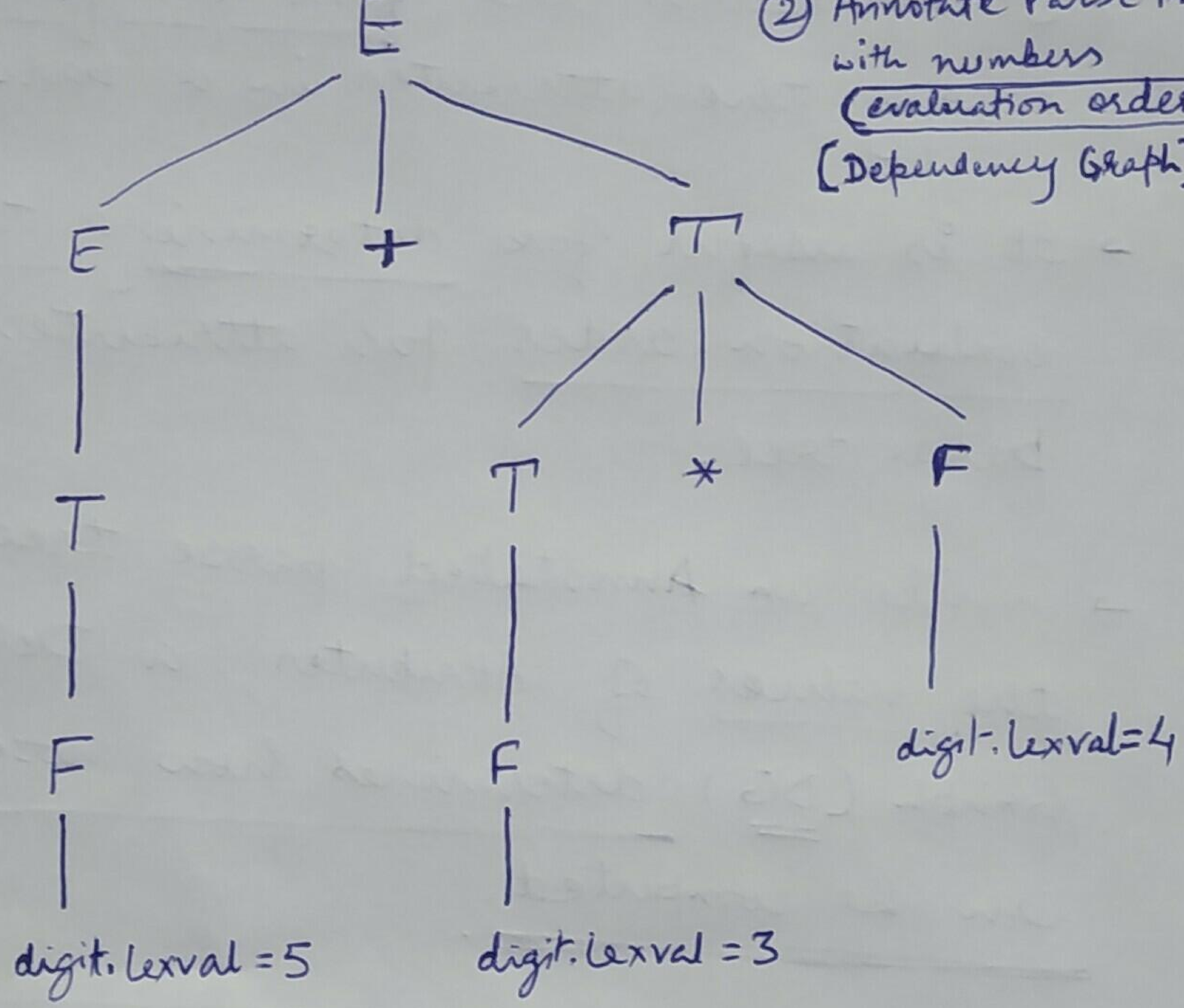
⇒ Draw the Dependency Graph for $5+3*4$

Step ②

① construct Parse Tree

② Annotate Parse Tree with numbers (evaluation order) [Dependency Graph]

Step ① PT



Step (2) DG for 5+3*4