Unit 3: Syntax Directed Definition (SDD)

- SDD = CGF + Semantic rules + attributes

→ Attributes are associated with grammar symbols and semantic rules are associated with productions.

→ 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, mem_loc, datatypes, etc.

productions	Semantic rule:
$E \rightarrow E+T$	$E.val = E.val + T.val$
$E \rightarrow T$	$E.val = T.val$

(grammar symbol; semantic rule; attributes)

Semantic means → it provides meaning to the corresponding productions.

Types of attributes:

① **Synthesized Attribute (SA):** If a node takes value from its children then it is called S.A

Ex: $A \rightarrow BCD$ A is a parent node

B, C, D → are children nodes

$A.S = B.S$
$A.S = C.S$
$A.S = D.S$

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

② **Inherited Attribute (IA):** If a node takes value from its parent or siblings.

$A \rightarrow BCD$

$C.i = A.i$ → parent node

$C.i = B.i$ sibling node

$C.i = D.i$

i → attributes associated with B, C, A, nodes.

Types of SDD :

(1) S-Attributed SDD or S-Attributed Definitions or S-Attributed grammars.

(2) L-attributed SDD or L-attributed Definitions or L-attributed grammars.

S-attributed SDD

(1) A SDD that uses only synthesized attributes is called as S-attributed SDD.

Ex: $A \rightarrow BCD$

$A.S \rightarrow B.S$

$A.S \rightarrow C.S$

$A.S \rightarrow D.S$

(2) Semantic actions are always placed at right end of the production. (ABC)

It is also called as "postfix SDD".

(3) Atts. are evaluated with Bottom up parsing.

L-attributed SDD

(1) A SDD that uses both Synthesized & inherited atts is called as L-att.d SDD but each inherited att is restricted to inherit from parent or left sibling only.

Ex: $A \rightarrow XYZ \{ Y.S = A.S,\ Y.S = X.S,\ Y.S = Z.S \}$ (Z.S ×)

(2) Semantic actions are placed anywhere on RHS.

(3) Atts, are evaluated by traversing parse tree.

* no guarantee of order of evaluation of atts.

SDD of a Simple Desk Calculator: (EXP: 3*5+4n)

or

SDD for evaluation of Expression:

or Annotated Parse tree.

productions	Semantic Rules:
augmented prod^n $L \to E_n$ (LHS variable) → print (E.val)	L.val = E.val
$E \to E_1 + T$	E.val = E_1.val + T.val
$E \to T$	E.val = T.val
$T \to T_1 * F$	T.val = T_1.val * F.val
$T \to F$	T.val = F.val
$F \to digit$	F.val = digit.lexval
$F \to (E)$	F.val = E.val

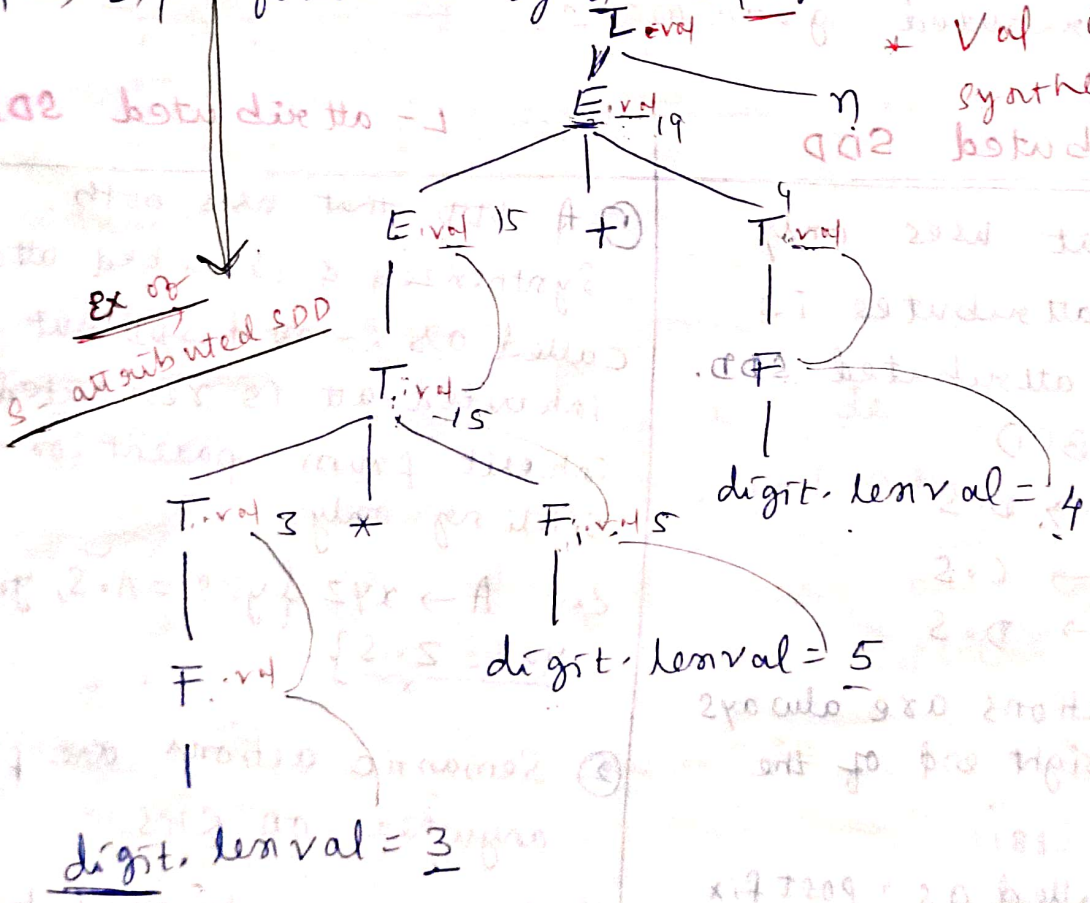
Exp^n grammar

$E_n \to Exp^n$ followed by n i.e $|n$.

* terminal

* Val is a synthesized att

Ex of S-attributed SDD



* The token digit has a synthesized att lexval whose value is assumed to be supplied by the LA.

Main goal: of Semantic analysis is to check for correctness of program & enable proper execution.

* Job of the parser is only to verify that the i/p pgm consists of tokens arranged in syntactically valid combination.

* In semantic Analysis, we check whether they form a sensible set of instructions in the pgmg. language.

Ex] type of RHS = type of LHS.

```
int a[] = 10+20; // valid
char b[] = "Hello"; // valid
int a = 10+b; // semantically invalid.
```

* (Se.A) checks whether the type & no. of parameters in the fn defn & fn call are same or not. If not, display app. error msg.

* checks whether type of operands are same in an arithmetic operation.

* App. type conversion is done in this phase.

* Semantic analysis is the third phase of the Compiler which acts as an interface b/w SA phase & code generation phase. It accepts the parse tree from SA phase & adds the semantic info to the parse tree & performs certain checks based on this info. It also helps constructing the symbol table with app. info.

The semantics of a language can be described using two notations namely:

1. Syntax Directed Definition (SDD)
2. ——————— Translation (SDT).

(APT).

Annotated Parse Tree: A parse tree showing the attribute values of each node. ie The terminals in the annotated parse tree can have only Synthesized attribute values & they are obtained directly from the LA. So there are no semantic rules in SDD to get the lexical values into terminals of the APT.

General approach to SDT / Evaluating an SDD at the nodes of a parse Tree

1. Construct a parse tree.
2. Compute the values of attributes at the node, using the rules.
3. Obtain the att. values for each variable/non-terminals & write the semantic rules for each prod?. When complete annotated parse tree is ready, we will have the complete SDD.

In what order do we evaluate attributes?

* If we want to evaluate an attribute of a node of a parse tree, it is necessary to evaluate all the atts. upon which its value depends.
* S-attributed SDD: evaluate atts in bottom up order.
* L-——————: there is no single order in which the atts. have to be evaluated. There can be one or more orders.

Ex: Inherited Attributes

Production	Semantic Rules
(1) $T \rightarrow FT'$	$T'.inh = F.val$ $T.val = T'.syn$
$T' \rightarrow * FT_1'$	$T_1'.inh = T'.inh \times F.val$ $T'.syn = T_1'.syn$
$T' \rightarrow \epsilon$	$T'.syn = T'.inh$
$F \rightarrow digit$	$F.val = digit.lexval$

Terminal digit has the attribute lexval.

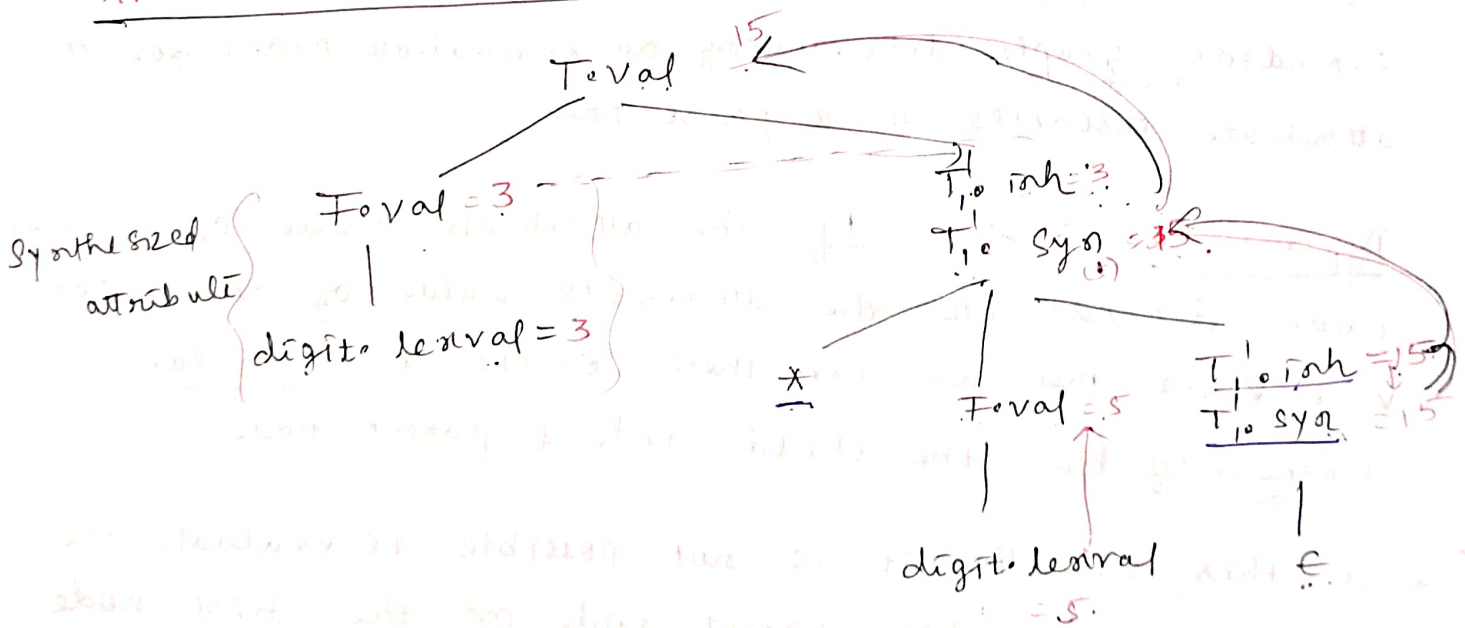
Attribute T & F has the attribute val.

T' has 2 attributes : syn - synthesized
inh - inherited.

$A \rightarrow A\alpha \mid \beta$
$A \rightarrow \beta A'$
$A' \rightarrow \alpha A' \mid \epsilon$

Syn & inh

Annotated Parse tree for 3*5:



SDT = grammar + semantic rules
production

Applications:

* Executing arithmetic exp^n
* conversion from infix to postfix
* " " to prefix
* " " binary to decimal
* counting no of reductions
* Creating syntax tree
* Generating intermediate code
* Type checking
* Storing type info into symbol table.

Evaluation Orders for SDD

Dependency graphs: determining an evaluation order for the attribute instances in a parse tree.

Dependency graph: If the attribute value of a parent node depends on the attribute value of child node & vice versa then we say there exists a circular dependency b/w the child node & parent node.

* In this situation it is not possible to evaluate the attribute of either parent node or the child node since one value depends on another value.

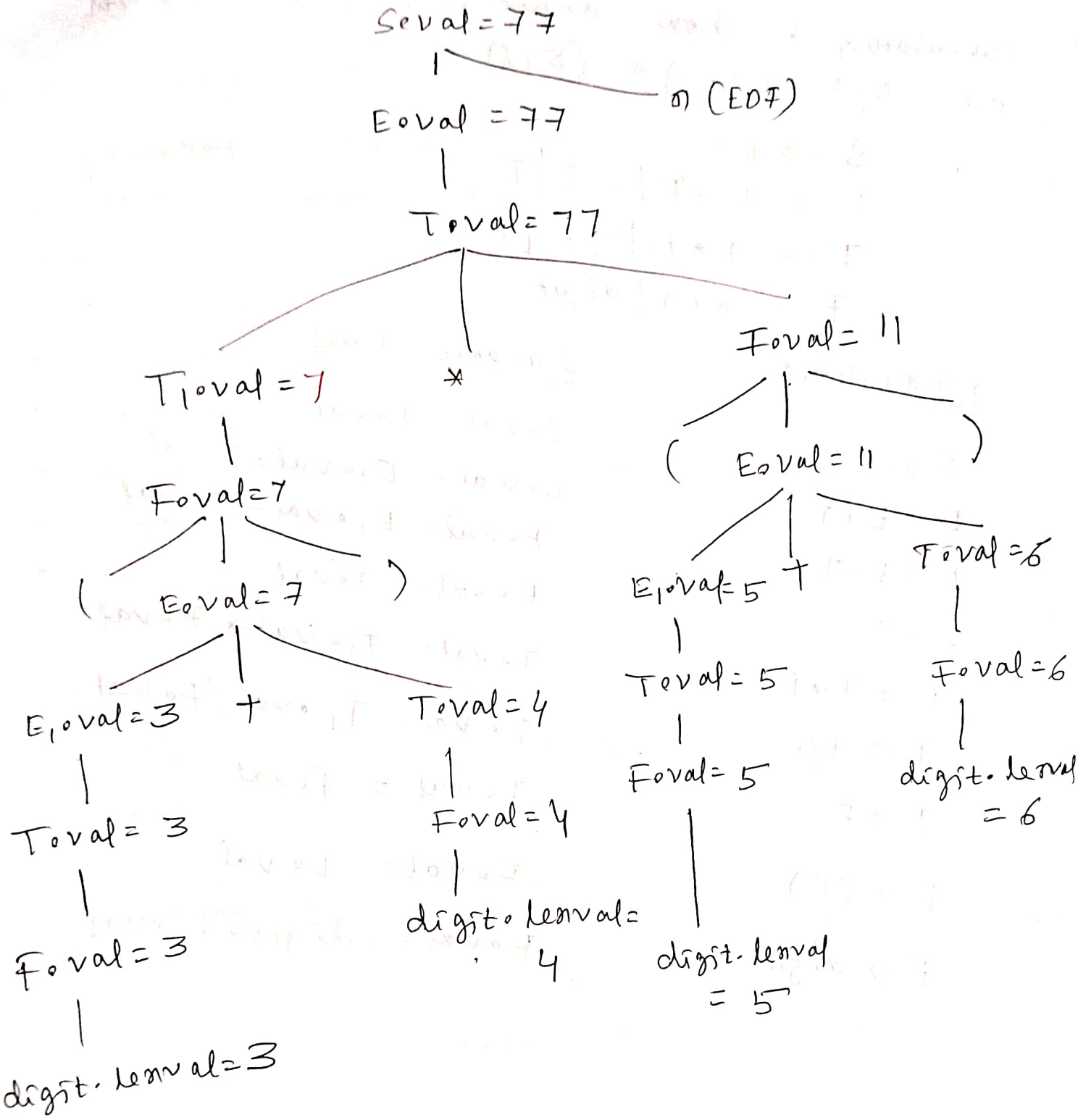
production	Semantic Rule	Partial annotated PT
$A \to B$	$A.s = B.i$ $B.i = A.s + 6$	$A.s$ $B.i$

Ex 2] Write grammar & SDD for a simple desk Calculator & show annotated parse tree for the exp^n (3+4) * (5+6).

$S \rightarrow En$
$E \rightarrow E+T \mid E-T \mid T$
$T \rightarrow T*F \mid T/F \mid F$
$F \rightarrow (E) \mid digit$

Productions	Semantic Rules
$S \rightarrow En$	$S.val = E.val$
$E \rightarrow E+T$	$E.val = E_1.val + T.val$
$E \rightarrow E-T$	$E.val = E_1.val - T.val$
$E \rightarrow T$	$E.val = T.val$
$T \rightarrow T*F$	$T.val = T_1.val * F.val$
$T \rightarrow T/F$	$T.val = T_1.val / F.val$
$T \rightarrow F$	$T.val = F.val$
$F \rightarrow (E)$	$F.val = E.val$
$F \rightarrow digit$	$F.val = digit.lexval$

(3+4) * (5+6)



3) Obtain SDD for the below grammar using Top down approach.

$S \to En$

$E \to E+T \mid T$

$T \to T*F \mid F$

$F \to (E) \mid digit$

Sol^n: $S \to En$

$E \to TE'$

$E' \to +TE_1' \mid \epsilon$

$T \to FT'$

$T' \to *FT_1' \mid \epsilon$

$F \to (E) \mid digit$.

The variables S, E, T & F are present both in given grammar & grammar obtained after left recursion. So, only for the variables S, E, T & F we use the att name v (stands for val) & for all the other variables we use s for synthesized att & i for inherited att.

Consider: $S \to En$
$F \to (E) \mid digit$
} no left recursion. & are retained in the grammar after eliminating left recursion. i.e the value of LHS can be comp uted from the att values of RHS (i.e children). Hence they have synthesized atts.

production	S.R	Type
$S \to En$	$S.v = E.v$	synthesized
$F \to (E)$	$F.v = E.v$	
$F \to digit$	$F.v = d.lexval$	

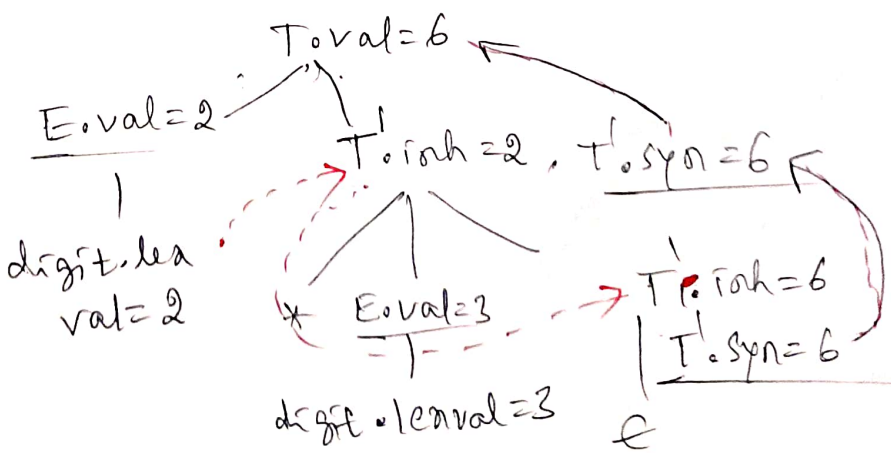
(2*3)

productions

$T \to FT'$

$T' \to *FT' \mid \epsilon$

$F \to digit$

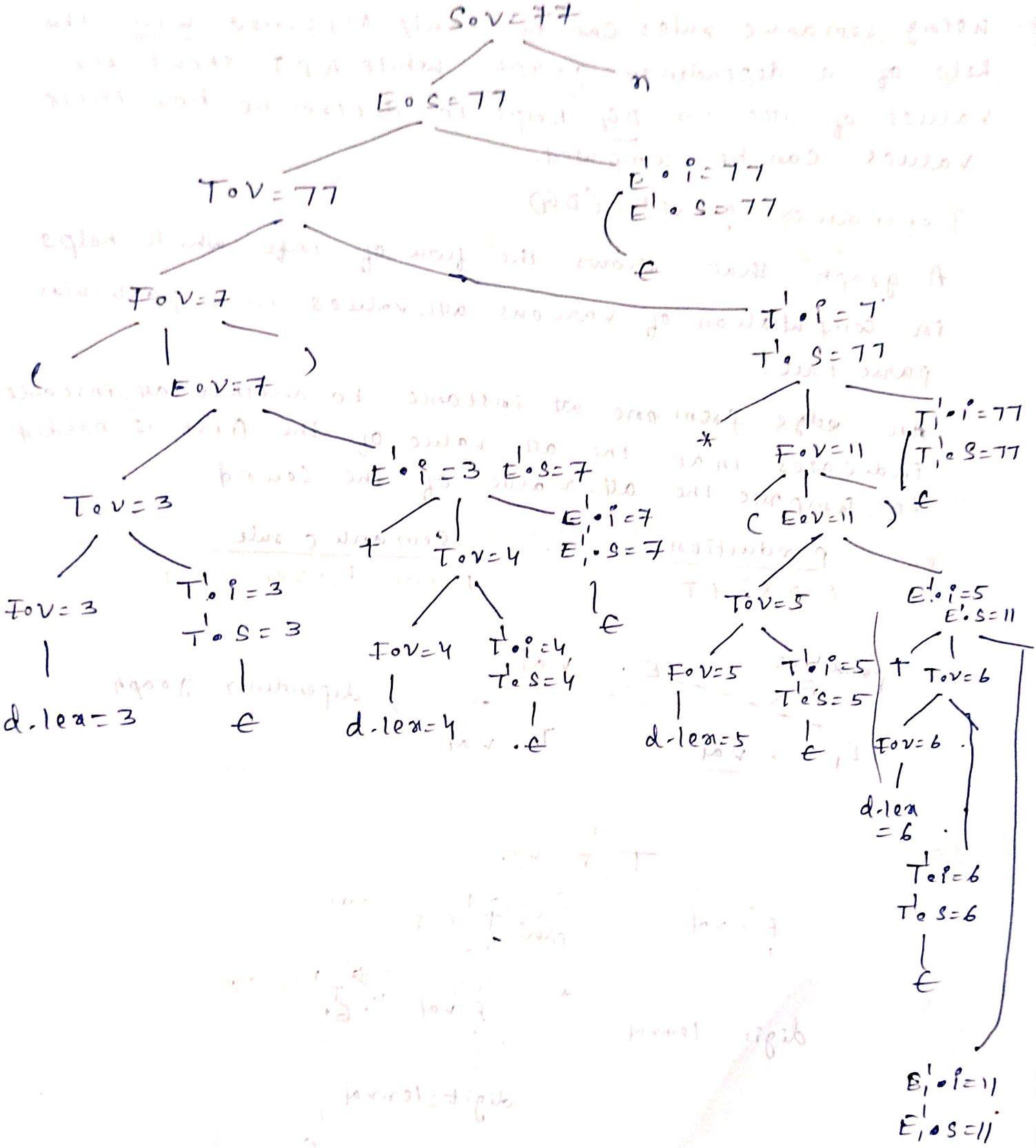


By following the dotted arrow lines, we can write various Semantic rules.

Combining all productions, the final SDD is shown below:

production	Semantic Rule	Type:
		Synthesized (S)
		Inherited (I)
$S \to E\$$	$S.V = E.V$	S
$E \to TE'$	$E'.inh = T.val$	I
	$E.val = E'.syn$	S
$E' \to +TE'$	$E'_1.inh = E'.inh + T.val$	I
	$E'.syn = E'_1.syn$	S
$E' \to \epsilon$	$E'.syn = E'_1.inh$	S
$T \to FT'$	$T'.inh = F.val$	I
	$T.val = T'.syn$	S
$T' \to *FT'$	$T'_1.inh = T'.inh * F.val$	I
	$T'.syn = T'_1.syn$	S
$T' \to \epsilon$	$T'.syn = T'_1.inh$	S
$F \to (E)$	$F.val = E.val$	S
$F \to digit$	$F.val = d.lexval.$	S

(3+4) * (5+6) Annotated Parse Tree:



Evaluation order for SDD's

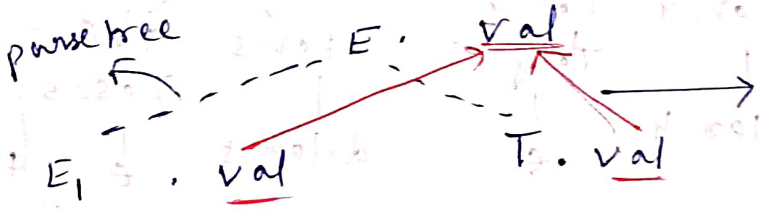
The evaluation order to find att. values in a parse tree using semantic rules can be easily obtained with the help of a dependency graph. While A.P.T shows the values of atts, a DG helps to determine how those values can be computed.

Dependency graphs (DG)

A graph that shows the flow of info which helps in computation of various atts, values for a particular parse tree.

An edge from one att instance to another att. instance indicates that the att. value of the first is needed to compute the att. value of the second.

Ex: production $E \to E_1 + T$ — Semantic rule $E.val = E_1.val + T.val$



Ex:-