Raiyan Jahangir
Roll-201914039

Ans no. 1

Ans:

The syntax directed definition shown in SDD-I

Production	Semantic Rule	Comment
1 $S \rightarrow xTU$	$S.val = T.val - x.lexval + U.val$	synthesized as parent takes value of children
2 $S \rightarrow lX$	$S.val = X.val + l.lexval$	DO
3 $S \rightarrow x$	$S.val = x.val$	DO
4 $T \rightarrow c$	$T.val = c.lexval$	DO
5 $T \rightarrow l$	$T.val = l.lexval$	DO
6 $X \rightarrow xX_1$	$X.val = X_1.pval + x.lenval$	DO
7 $X \rightarrow U$	$X.val = U.val$	DO
8 $U \rightarrow iY$	$U.val = Y.val - i.lenval$	DO
9 $U \rightarrow vI$	$U.val = I.val + v.lexval$	DO
10 $U \rightarrow I$	$U.val = I.val$	DO
11 $Y \rightarrow x$	$Y.val = x.lexval$	DO
12 $Y \rightarrow v$	$Y.val = v.lexval$	DO
13 $I \rightarrow iI_1$	$I.val = I_1.val + i.lenval$	DO
14 $I \rightarrow \varepsilon$	$I.val = 0$	DO

Here, all ~~val~~ attributes are synthesized attributes.

So, the subclass of this SDD is S-attributed SDD.

As, S-attributed SDD is a subset of L-attributed SDD, the given SDD is also L-attributed.

Raiyan Jahangir

Roll - 201914039

Ans no. 3

My roll = 201914039 = odd

∴ Input = xlvii

∴ $\boxed{S.val = 42}$

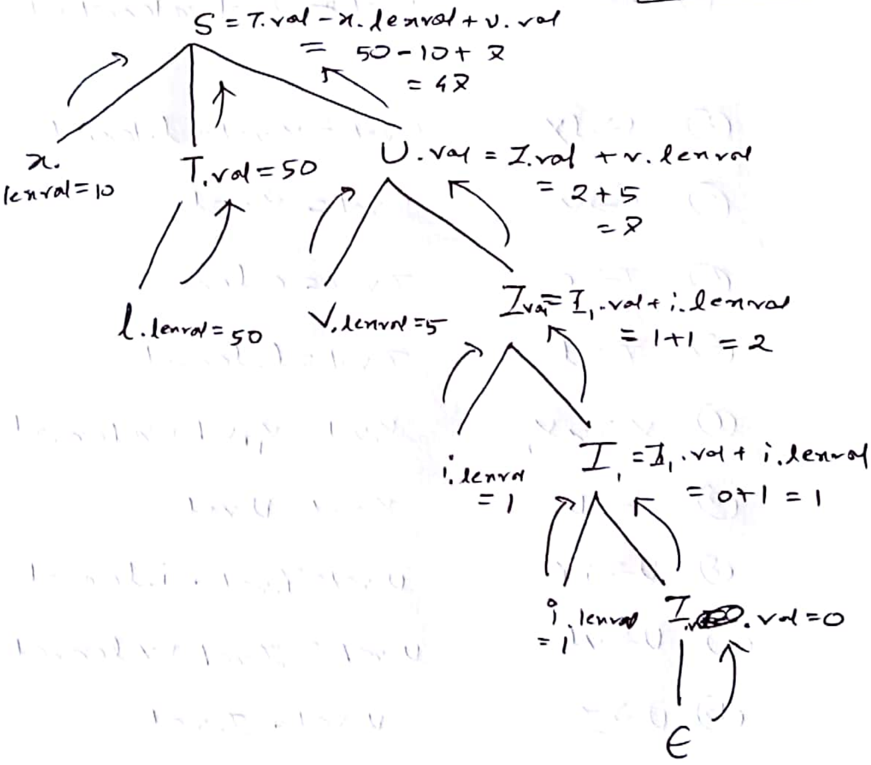


Fig: Annotated Parse tree of xlvii

Ans no. 4

Ans:

Applications of SDT:

A syntax-directed translation scheme is a content-free grammer with program fragments embedded within production bodies. The program fragments are called semantic actions and can be present at any position of a production body. For example,

$E \rightarrow E_1 + T$ {print '+'}

Here, along with the production, an action is given, which is the program fragment. By convention, we place curly braces around actions.

By SDTs, we can implement various arithmetic expressions like

① Converting infix to postfix expression
② " " " prefix "
③ For type checking and intermediate code generation
④ Construct parse tree with grammer and semantic actions.

⑤ Constructing syntax trees

Raiyan Jahangir

Roll-201914039

Modifying the SDD-2 so that it can construct syntax trees:-

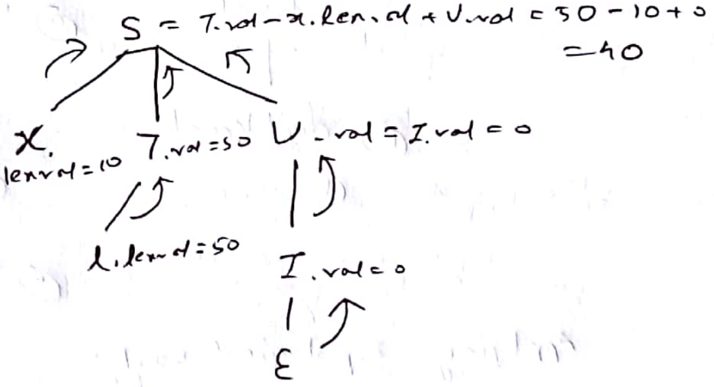
Productions	Semantic Rules
① $T \to FT'$	$T'.inh = F.val$ $T.val = T'.syn$
② $T' \to * F T_1'$	~~$F_1.inh$~~ $T_1'.inh =$ new Node ('*', $F'.inh$, $T'.node$) $T_{\cancel{1}}'.syn = T_1'.syn$
③ $T' \to \epsilon$	$T'.syn = T'.inh$
④ $F \to digit$	$F.node =$ new leaf (digit, digit entry)

Ans no. 2

Ans: By observing the SDD of SDD-1 and SDD-2, it is clear that SDD-1 is both S-attributed and L-attributed and SDD-2 is only L-attributed

In synthecized attributes, we carry-out bottom-up parsing. Because, in synthesized or S-attributted SDD, a node only takes values of itself or its children. so, we have to go down to leaf nodes, find the value of the leaf nodes and eventually reach the parent or root nodes. So, the evaluation order of SDD-I is bottom-up parsing.

Example:



Here, values are flowing from bottom to top. And finding values ~~at~~ any order from leaf nodes is alright.

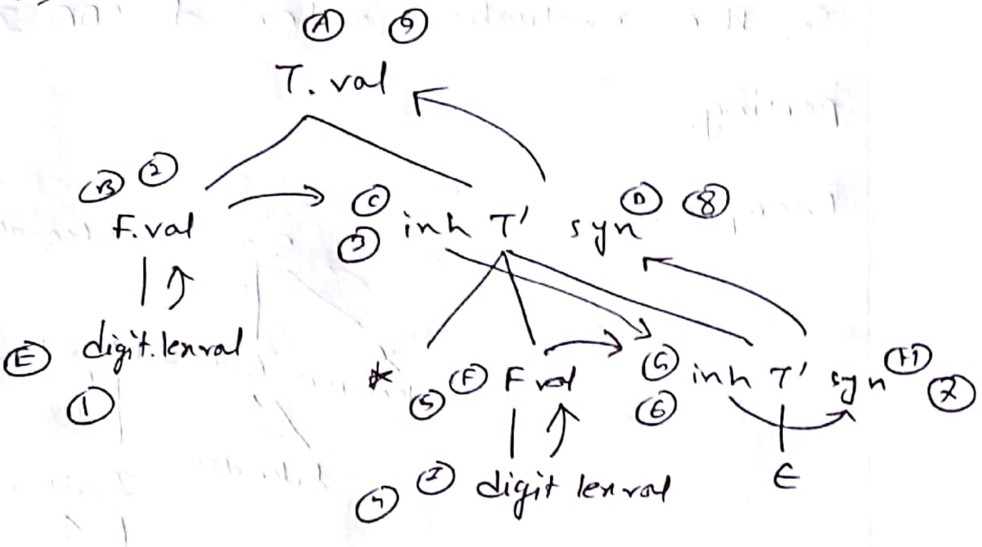
SDD-2 has both synthesized and inherited attributes. And thus, it is suitable for top-down parsing. In top down parsing, ~~it is somethin~~ the values are determined from root to leaves. Sometimes it is hard to determine whose value is to ~~come~~ be ~~ear~~ determined earlier. In such a case, we use topological sort.

Ⓒ

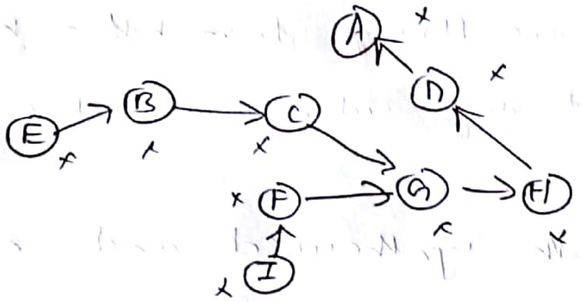
Raiyan Jahangir
Roll-201914039

Example :

Let the nodes be



Applying topological sort :



Order : E, B, C, I, F, G, H, D, A

∴ Evaluation order is given in the figure above ~~by~~ after applying topological sort