Syntax Directed Translation

A *syntax-directed definition* (SDD) is a context-free grammar together with attributes and rules. Attributes are associated with grammar symbols and rules are associated with productions. If X is a symbol and a is one of its attributes, then we write $X.a$ to denote the value of a at a particular parse-tree node labeled X. If we implement the nodes of the parse tree by records or objects, then the attributes of X can be implemented by data fields in the records that represent the nodes for X. Attributes may be of any kind: numbers, types, table references, or strings, for instance. The strings may even be long sequences of code, say code in the intermediate language used by a compiler.

Types of Attributes in SDT scheme

1. A *synthesized attribute* for a nonterminal A at a parse-tree node N is defined by a semantic rule associated with the production at N. Note that the production must have A as its head. A synthesized attribute at node N is defined only in terms of attribute values at the children of N and at N itself.

2. An *inherited attribute* for a nonterminal B at a parse-tree node N is defined by a semantic rule associated with the production at the parent of N. Note that the production must have B as a symbol in its body. An inherited attribute at node N is defined only in terms of attribute values at N's parent, N itself, and N's siblings.

Ex: 1. Consider the grammar and find the annotated parse tree.

	PRODUCTION	SEMANTIC RULES
1)	$L \rightarrow E$ **n**	$L.val = E.val$
2)	$E \rightarrow E_1 + T$	$E.val = E_1.val + T.val$
3)	$E \rightarrow T$	$E.val = T.val$
4)	$T \rightarrow T_1 * F$	$T.val = T_1.val \times F.val$
5)	$T \rightarrow F$	$T.val = F.val$
6)	$F \rightarrow (E)$	$F.val = E.val$
7)	$F \rightarrow$ **digit**	$F.val =$ **digit**.lexval

Figure 5.1: Syntax-directed definition of a simple desk calculator

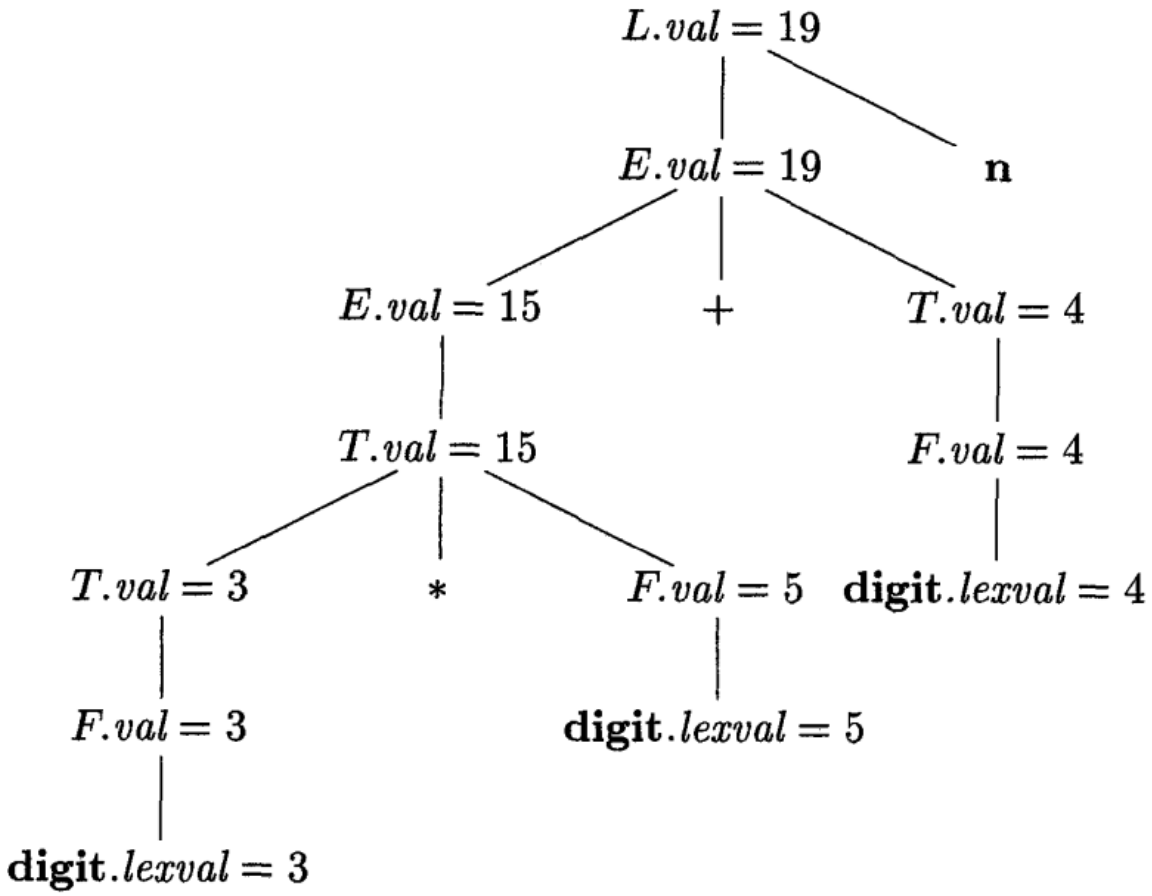
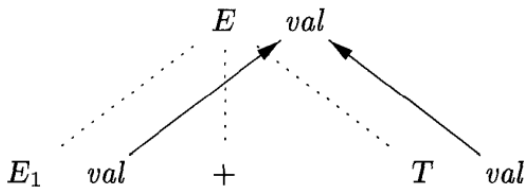


Figure 5.3: Annotated parse tree for $3 * 5 + 4$ **n**

Dependency Graph

- It depicts the flow of of information among the attribute instances in a particular parse tree.
- An edge from one attribute to another one means value of the first is needed to compute second.
- Annotated parse shows the values of the attributes, whereas dependency graph helps to determine how those values can be computed.

PRODUCTION	SEMANTIC RULE
$E \rightarrow E_1 + T$	$E.val = E_1.val + T.val$

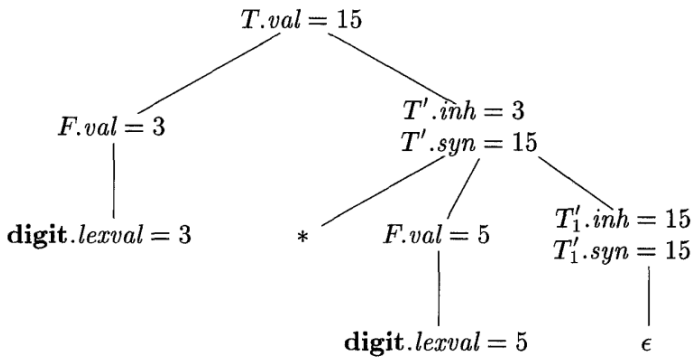


: $E.val$ is synthesized from $E_1.val$ and $E_2.val$

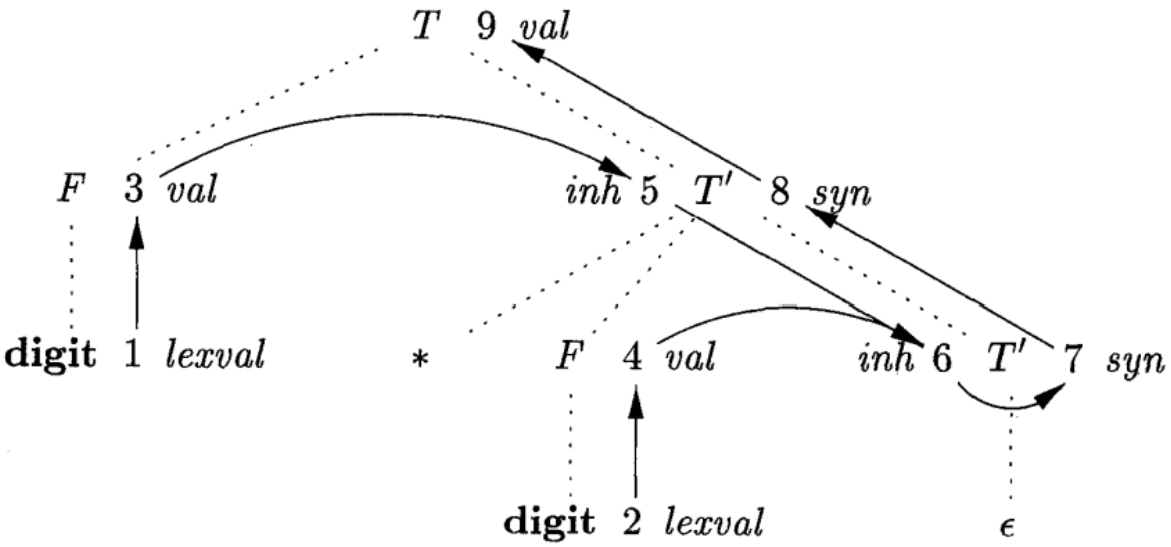
Ex: 2.

	PRODUCTION	SEMANTIC RULES
1)	$T \rightarrow F\,T'$	$T'.inh = F.val$ $T.val = T'.syn$
2)	$T' \rightarrow *\,F\,T'_1$	$T'_1.inh = T'.inh \times F.val$ $T'.syn = T'_1.syn$
3)	$T' \rightarrow \epsilon$	$T'.syn = T'.inh$
4)	$F \rightarrow$ **digit**	$F.val =$ **digit**$.lexval$

: An SDD based on a grammar suitable for top-down parsing



Annotated parse tree for 3*5



Types of SDD

S-Attributed Definition	L-Attributed Definition
An SDT uses only synthesized attributes	An SDT uses both synthesized attributes and inherited attributes
S-attributed SDTs are evaluated in bottom-up parsing,	Attributes in L-attributed SDTs are evaluated by depth-first and left-to-right parsing manner. (Top-Down)
Semantic actions are placed anywhere in RHS	Semantic actions are placed anywhere in RHS
The values of the parent nodes depend upon the values of the child nodes.	Inherited attribute can inherit values from left siblings only

Ex: 4.

Example – Consider the given below SDT.

```
P1: S -> MN  {S.val= M.val + N.val}
P2: M -> PQ  {M.val = P.val * Q.val  and P.val =Q.val}
```

Select the correct option.

A. Both P1 and P2 are S attributed.

B. P1 is S attributed and P2 is L-attributed.

C. P1 is L attributed but P2 is not L-attributed.

D. None of the above

Ex: 5.

Example 5.9 : Any SDD containing the following production and rules cannot be L-attributed:

PRODUCTION	SEMANTIC RULES
$A \rightarrow B\ C$	$A.s = B.b;$
	$B.i = f(C.c, A.s)$

Ex: 6. Construct the parse tree and show the dependency among the attributes.

	PRODUCTION	SEMANTIC RULES
1)	$D \rightarrow T\ L$	$L.inh = T.type$
2)	$T \rightarrow$ **int**	$T.type =$ integer
3)	$T \rightarrow$ **float**	$T.type =$ float
4)	$L \rightarrow L_1$, **id**	$L_1.inh = L.inh$
		$addType(\textbf{id}.entry, L.inh)$
5)	$L \rightarrow$ **id**	$addType(\textbf{id}.entry, L.inh)$