# Chapter 5

# Syntax-Directed Translation

This chapter develops the theme of Section 2.3: the translation of languages guided by context-free grammars. The translation techniques in this chapter will be applied in Chapter 6 to type checking and intermediate-code generation. The techniques are also useful for implementing little languages for specialized tasks; this chapter includes an example from typesetting.

We associate information with a language construct by attaching *attributes* to the grammar symbol(s) representing the construct, as discussed in Section 2.3.2. A syntax-directed definition specifies the values of attributes by associating semantic rules with the grammar productions. For example, an infix-to-postfix translator might have a production and rule

| PRODUCTION | SEMANTIC RULE | (5.1) |
|---|---|---|
| $E \rightarrow E_1 + T$ | $E.code = E_1.code \parallel T.code \parallel \text{'+'}$ | |

This production has two nonterminals, $E$ and $T$; the subscript in $E_1$ distinguishes the occurrence of $E$ in the production body from the occurrence of $E$ as the head. Both $E$ and $T$ have a string-valued attribute *code*. The semantic rule specifies that the string $E.code$ is formed by concatenating $E_1.code$, $T.code$, and the character '+'. While the rule makes it explicit that the translation of $E$ is built up from the translations of $E_1$, $T$, and '+', it may be inefficient to implement the translation directly by manipulating strings.

From Section 2.3.5, a syntax-directed translation scheme embeds program fragments called semantic actions within production bodies, as in

$$E \rightarrow E_1 + T \quad \{ \text{ print '+' } \} \tag{5.2}$$

By convention, semantic actions are enclosed within curly braces. (If curly braces occur as grammar symbols, we enclose them within single quotes, as in

'{' and '}'.) The position of a semantic action in a production body determines the order in which the action is executed. In production (5.2), the action occurs at the end, after all the grammar symbols; in general, semantic actions may occur at any position in a production body.

Between the two notations, syntax-directed definitions can be more readable, and hence more useful for specifications. However, translation schemes can be more efficient, and hence more useful for implementations.

The most general approach to syntax-directed translation is to construct a parse tree or a syntax tree, and then to compute the values of attributes at the nodes of the tree by visiting the nodes of the tree. In many cases, translation can be done during parsing, without building an explicit tree. We shall therefore study a class of syntax-directed translations called "L-attributed translations" (L for left-to-right), which encompass virtually all translations that can be performed during parsing. We also study a smaller class, called "S-attributed translations" (S for synthesized), which can be performed easily in connection with a bottom-up parse.

## 5.1 Syntax-Directed Definitions

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.

### 5.1.1 Inherited and Synthesized Attributes

We shall deal with two kinds of attributes for nonterminals:

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.

| | PRODUCTION | SEMANTIC RULES |
|---|---|---|
| 1) | $L \to E\ \mathbf{n}$ | $L.val = E.val$ |
| 2) | $E \to E_1\ +\ T$ | $E.val = E_1.val + T.val$ |
| 3) | $E \to T$ | $E.val = T.val$ |
| 4) | $T \to T_1\ *\ F$ | $T.val = T_1.val \times F.val$ |
| 5) | $T \to F$ | $T.val = F.val$ |
| 6) | $F \to (\ E\ )$ | $F.val = E.val$ |
| 7) | $F \to \mathbf{digit}$ | $F.val = \mathbf{digit}.\text{lexval}$ |

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

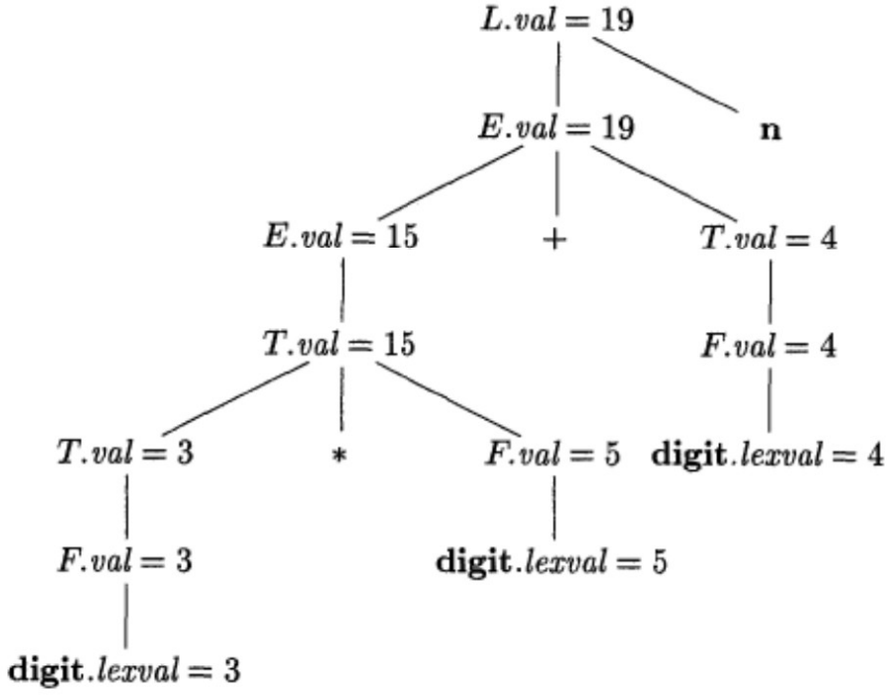


Figure 5.3: Annotated parse tree for $3 * 5 + 4\ \mathbf{n}$

**Example 5.3:** The SDD in Fig. 5.4 computes terms like $3 * 5$ and $3 * 5 * 7$. The top-down parse of input $3 * 5$ begins with the production $T \to F\,T'$. Here, $F$ generates the digit 3, but the operator $*$ is generated by $T'$. Thus, the left operand 3 appears in a different subtree of the parse tree from $*$. An inherited attribute will therefore be used to pass the operand to the operator.

The grammar in this example is an excerpt from a non-left-recursive version of the familiar expression grammar; we used such a grammar as a running example to illustrate top-down parsing in Section 4.4.

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

Figure 5.4: An SDD based on a grammar suitable for top-down parsing

Each of the nonterminals $T$ and $F$ has a synthesized attribute *val*; the terminal **digit** has a synthesized attribute *lexval*. The nonterminal $T'$ has two attributes: an inherited attribute *inh* and a synthesized attribute *syn*.

The semantic rules are based on the idea that the left operand of the operator $*$ is inherited. More precisely, the head $T'$ of the production $T' \rightarrow * F\, T'_1$ inherits the left operand of $*$ in the production body. Given a term $x * y * z$, the root of the subtree for $* y * z$ inherits $x$. Then, the root of the subtree for $* z$ inherits the value of $x * y$, and so on, if there are more factors in the term. Once all the factors have been accumulated, the result is passed back up the tree using synthesized attributes.

To see how the semantic rules are used, consider the annotated parse tree for $3 * 5$ in Fig. 5.5. The leftmost leaf in the parse tree, labeled **digit**, has attribute value $lexval = 3$, where the 3 is supplied by the lexical analyzer. Its parent is for production 4, $F \rightarrow$ **digit**. The only semantic rule associated with this production defines $F.val =$ **digit**$.lexval$, which equals 3.

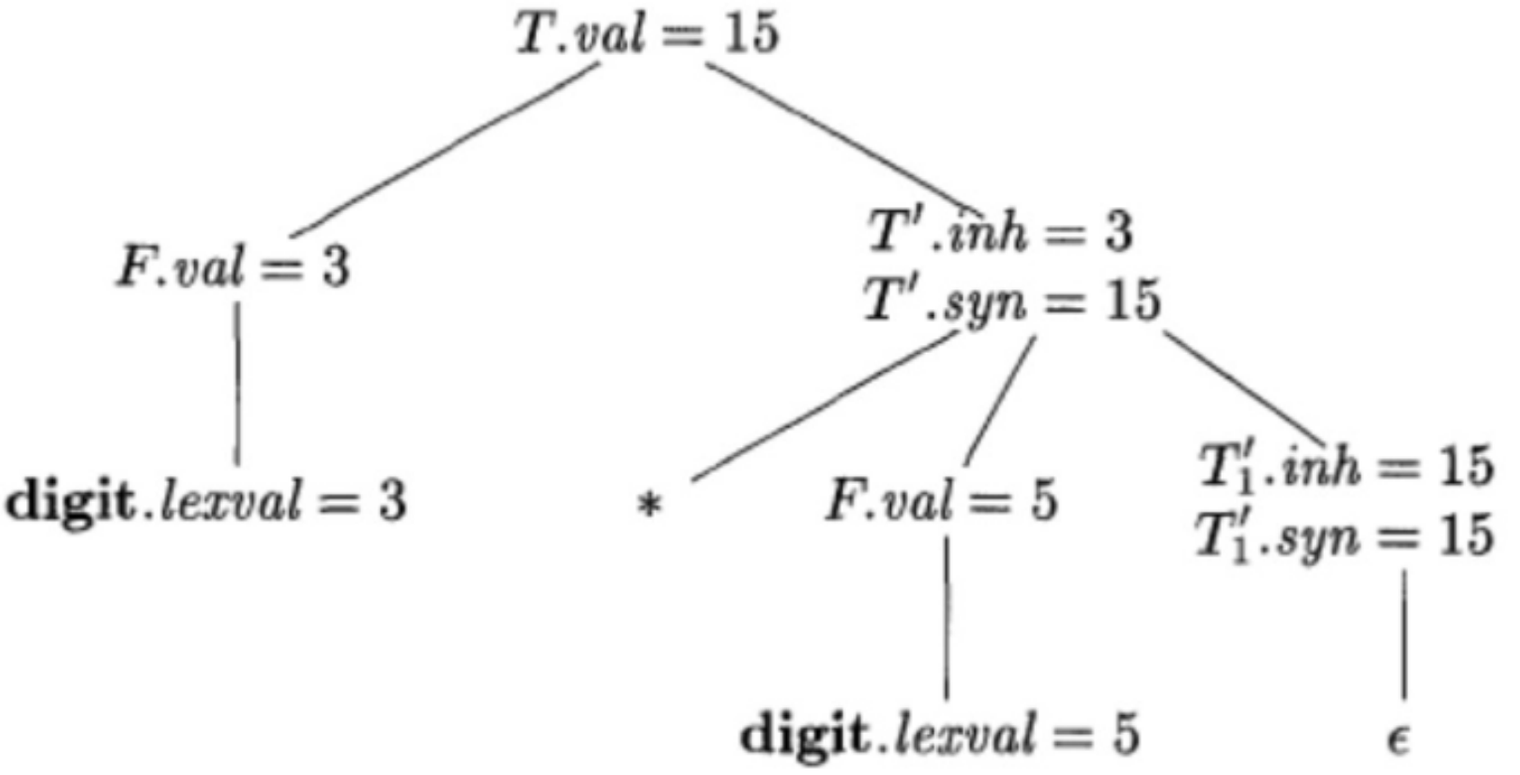


Figure 5.5: Annotated parse tree for $3 * 5$

At the second child of the root, the inherited attribute $T'.inh$ is defined by the semantic rule $T'.inh = F.val$ associated with production 1. Thus, the left operand, 3, for the $*$ operator is passed from left to right across the children of the root.

The production at the node for $T'$ is $T' \rightarrow *FT_1'$. (We retain the subscript 1 in the annotated parse tree to distinguish between the two nodes for $T'$.) The inherited attribute $T_1'.inh$ is defined by the semantic rule $T_1'.inh = T'.inh \times F.val$ associated with production 2.

With $T'.inh = 3$ and $F.val = 5$, we get $T_1'.inh = 15$. At the lower node for $T_1'$, the production is $T' \rightarrow \epsilon$. The semantic rule $T'.syn = T'.inh$ defines $T_1'.syn = 15$. The *syn* attributes at the nodes for $T'$ pass the value 15 up the tree to the node for $T$, where $T.val = 15$. □

### 5.2.1 Dependency Graphs

A *dependency graph* depicts the flow of information among the attribute instances in a particular parse tree; an edge from one attribute instance to another means that the value of the first is needed to compute the second. Edges express constraints implied by the semantic rules. In more detail:

- For each parse-tree node, say a node labeled by grammar symbol $X$, the dependency graph has a node for each attribute associated with $X$.
- Suppose that a semantic rule associated with a production $p$ defines the value of synthesized attribute $A.b$ in terms of the value of $X.c$ (the rule may define $A.b$ in terms of other attributes in addition to $X.c$). Then, the dependency graph has an edge from $X.c$ to $A.b$. More precisely, at every node $N$ labeled $A$ where production $p$ is applied, create an edge to attribute $b$ at $N$, from the attribute $c$ at the child of $N$ corresponding to this instance of the symbol $X$ in the body of the production.[2]
- Suppose that a semantic rule associated with a production $p$ defines the value of inherited attribute $B.c$ in terms of the value of $X.a$. Then, the dependency graph has an edge from $X.a$ to $B.c$. For each node $N$ labeled $B$ that corresponds to an occurrence of this $B$ in the body of production $p$, create an edge to attribute $c$ at $N$ from the attribute $a$ at the node $M$ that corresponds to this occurrence of $X$. Note that $M$ could be either the parent or a sibling of $N$.

**Example 5.4:** Consider the following production and rule:

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

At every node $N$ labeled $E$, with children corresponding to the body of this production, the synthesized attribute *val* at $N$ is computed using the values of *val* at the two children, labeled $E$ and $T$. Thus, a portion of the dependency graph for every parse tree in which this production is used looks like Fig. 5.6. As a convention, we shall show the parse tree edges as dotted lines, while the edges of the dependency graph are solid. □

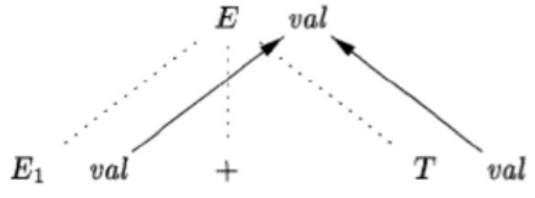


Figure 5.6: $E.val$ is synthesized from $E_1.val$ and $E_2.val$

**Example 5.5:** An example of a complete dependency graph appears in Fig. 5.7. The nodes of the dependency graph, represented by the numbers 1 through 9, correspond to the attributes in the annotated parse tree in Fig. 5.5.

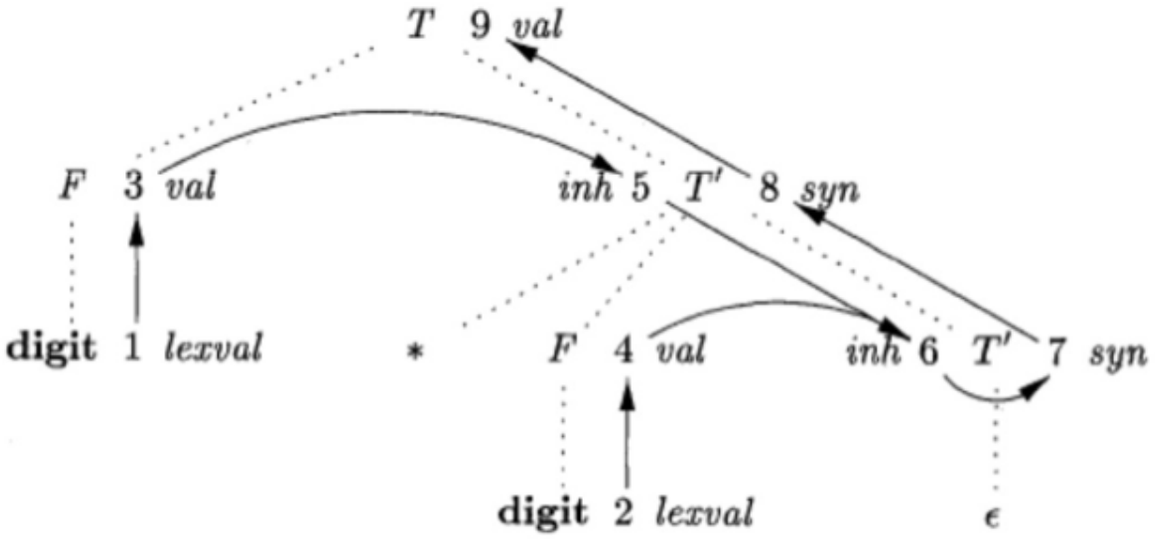


Figure 5.7: Dependency graph for the annotated parse tree of Fig. 5.5

Nodes 1 and 2 represent the attribute *lexval* associated with the two leaves labeled **digit**. Nodes 3 and 4 represent the attribute *val* associated with the two nodes labeled $F$. The edges to node 3 from 1 and to node 4 from 2 result from the semantic rule that defines $F.val$ in terms of **digit**.*lexval*. In fact, $F.val$ equals **digit**.*lexval*, but the edge represents dependence, not equality.

Nodes 5 and 6 represent the inherited attribute $T'.inh$ associated with each of the occurrences of nonterminal $T'$. The edge to 5 from 3 is due to the rule $T'.inh = F.val$, which defines $T'.inh$ at the right child of the root from $F.val$ at the left child. We see edges to 6 from node 5 for $T'.inh$ and from node 4 for $F.val$, because these values are multiplied to evaluate the attribute $inh$ at node 6.

Nodes 7 and 8 represent the synthesized attribute *syn* associated with the occurrences of $T'$. The edge to node 7 from 6 is due to the semantic rule $T'.syn = T'.inh$ associated with production 3 in Fig. 5.4. The edge to node 8 from 7 is due to a semantic rule associated with production 2.

Finally, node 9 represents the attribute $T.val$. The edge to 9 from 8 is due to the semantic rule, $T.val = T'.syn$, associated with production 1. □

Example 1)

Consider the following grammar:

$T \rightarrow FT'$

$T' \rightarrow +FT'$

$T' \rightarrow \epsilon$

$F \rightarrow 1|2|3|\cdots|9$

i) Construct SDD for the grammar

ii) Using SDD, draw annotated parse tree for input "2+3+4"

Solution:

| PRODUCTIONS | SDD |
|---|---|
| $T \rightarrow FT'$ | $T'.inh = F.val$<br>$T.val = T'.syn$ |
| $T' \rightarrow +FT_1'$ | $T_1'.inh = T'.inh + F.val$<br>$T'.syn = T_1'.syn$ |
| $T' \rightarrow \epsilon$ | $T'.syn = T'.inh$ |
| $F \rightarrow 1$ | $F.val = 1$ |
| $F \rightarrow 2$ | $F.val = 2$ |
| $F \rightarrow 3$ | $F.val = 3$ |
| ⋮ | ⋮ |
| $F \rightarrow 9$ | $F.val = 9$ |

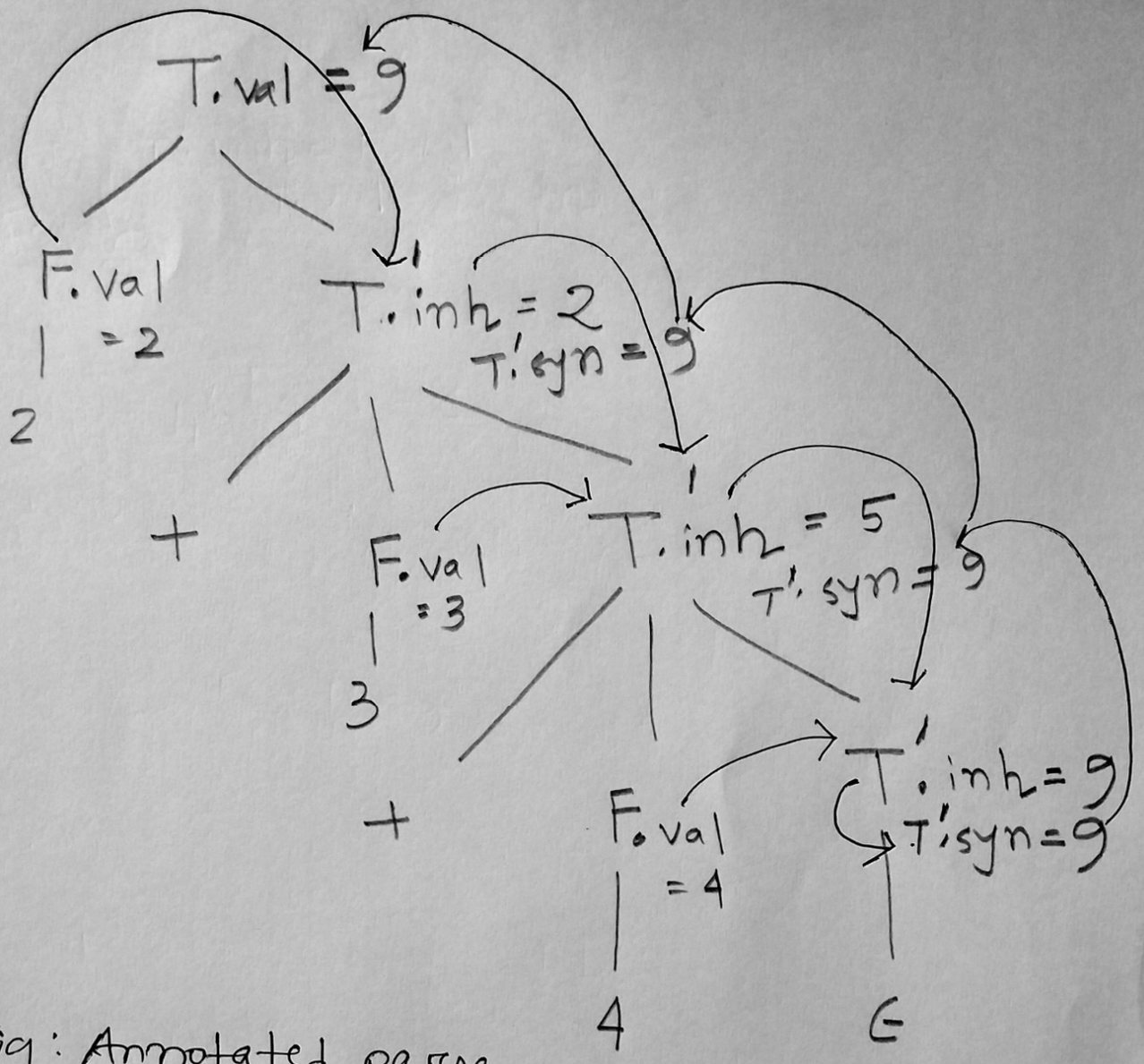


Fig: Annotated parse tree for input 2+3+4

# Introduction

**Syntax Directed Definitions (SDD)** are a generalization of context-free grammars in which:

- Grammar symbols have an associated set of **Attributes**;
- Productions are associated with **Semantic Rules** for computing the values of attributes.

  Example:

  CFG: E → E + T

  SDD: E → E1 + T { E.val = E1.val + T.val}

- Such formalism generates **Annotated Parse-Trees** where each node of the tree is a record with a field for each attribute (e.g., *X.a* indicates the attribute *a* of the grammar symbol *X*).

# SDD Example

Example string:
4 * 5 + 6

| Productions | Semantic Rules |
|---|---|
| S → E | S.val = E.val |
| E → $E_1$ + T | E.val = $E_1$.val + T.val |
| E → T | E.val = T.val |
| T → $T_1$ * F | T.val = $T_1$.val * F.val |
| T→ F | T.val = F.val |
| F→ digit | F.val = digit.lexval |

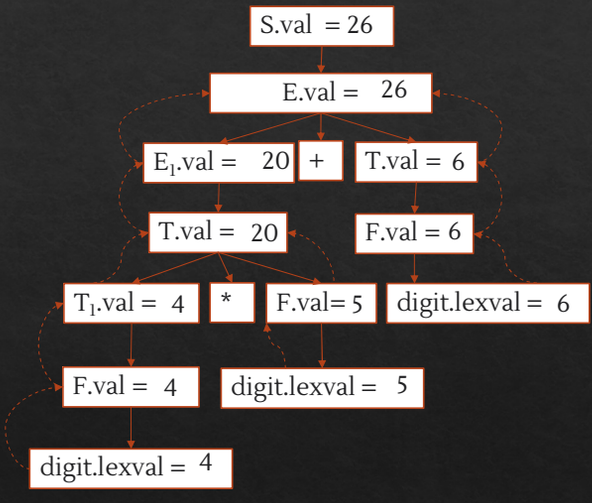


Fig. Annotated Parse Tree 4 * 5 + 6

# Types of Attributes

- Two types of attributes:

1. **Synthesized Attributes:** A synthesized attribute at node N is defined only in terms of attribute values at the children of N and at N itself. For example, let's say A -> BC is a production of a grammar, and A's attribute is dependent on B's attributes or C's attributes then it will be synthesized attribute. (i.e. A.val = B.val + 1, A.val = B.val+C.in)

2. **Inherited Attributes:** An inherited attribute at node N is defined only in terms of attribute values at N's parent, N itself, and N's siblings. For example, let's say A -> BC is a production of a grammar and B's attribute is dependent on A's attributes or C's attributes then it will be inherited attribute. (i.e. C.val = B.val + 1 or C.val = A.val)

# Types of SDD

- Two types of Definitions:

1. **S-Attributed:**

   If an SDD uses only synthesized attributes, it is called as S-attributed SDD.

   S-attributed SDDs are evaluated in bottom-up parsing, as the values of the parent nodes depend upon the values of the child nodes.

2. **L-Attributed:**

   If an SDD uses synthesized attributes and inherited attributes with a restriction that inherited attribute can inherit values from left siblings only, it is called as L-attributed SDD.

   Attributes in L-attributed SDDs are evaluated by depth-first and left-to-right parsing manner.

# SDD Example

Example string:
4 * 5 + 6

| Productions | Semantic Rules |
|---|---|
| S → E | S.val = E.val |
| E → $E_1$ + T | E.val = $E_1$.val + T.val |
| E → T | E.val = T.val |
| T → $T_1$ * F | T.val = $T_1$.val * F.val |
| T→ F | T.val = F.val |
| F→ digit | F.val = digit.lexval |

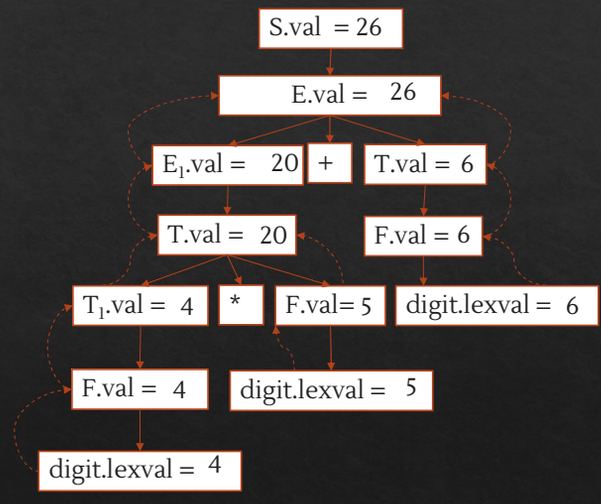


Fig. Annotated Parse Tree 4 * 5 + 6

# L-Attributed SDD

| Productions | Semantic Rules |
|---|---|
| T →FT' | T'.inh = F.val<br>T.val = T'.syn |
| T' →*F$T_1$' | $T_1$'.inh = T'.inh * F.val<br>T'.syn = $T_1$'.syn |
| T' → ε | T'.syn = T'.inh |
| F→ digit | F.val = digit.lexval |

x*y*z

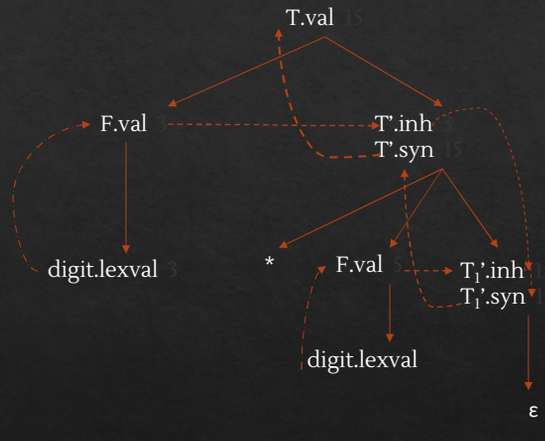


Fig. Dependency Graph for 3*5