Compiler Design

Fatemeh Deldar

Isfahan University of Technology

1402-1403

Syntax-Directed Translation

- **Goal:** The translation of languages guided by context-free grammars
- Syntax-directed translation attaches attributes to the grammar symbol(s) representing the construct
- A syntax-directed definition specifies the values of attributes by **associating semantic rules with the grammar productions**
- **Example:** An infix-to-postfix translator might have a production and rule

PRODUCTION	SEMANTIC RULE
$E \rightarrow E_1 + T$	$E.code = E_1.code \parallel T.code \parallel '+'$

  - **code: A string-valued attribute**

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***
- We shall deal with two kinds of attributes for non-terminals
  - **Synthesized attribute**
    - **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***
    - **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**
  - **Inherited attribute**
    - **An inherited attribute for a non-terminal *B* at a parse-tree node *N* is defined by a semantic rule associated with the production at the parent of *N***
    - **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**

Syntax-Directed Definitions

- While we do not allow an inherited attribute at node N to be defined in terms of attribute values at the children of node N, we do allow a synthesized attribute at node N to be defined in terms of inherited attribute values at node N itself

- **Example**
  - The following SDD evaluates expressions terminated by an endmarker ***n***
  - In the SDD, each of the non-terminals has a single synthesized attribute, called *val*
  - The terminal digit has a synthesized attribute *lexval*

	Production	Semantic Rules
1)	$L \rightarrow E\ \mathbf{n}$	$L.val = E.val$
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)	$F \rightarrow \mathbf{digit}$	$F.val = \mathbf{digit}.\text{lexval}$

Evaluating an SDD at the Nodes of a Parse Tree

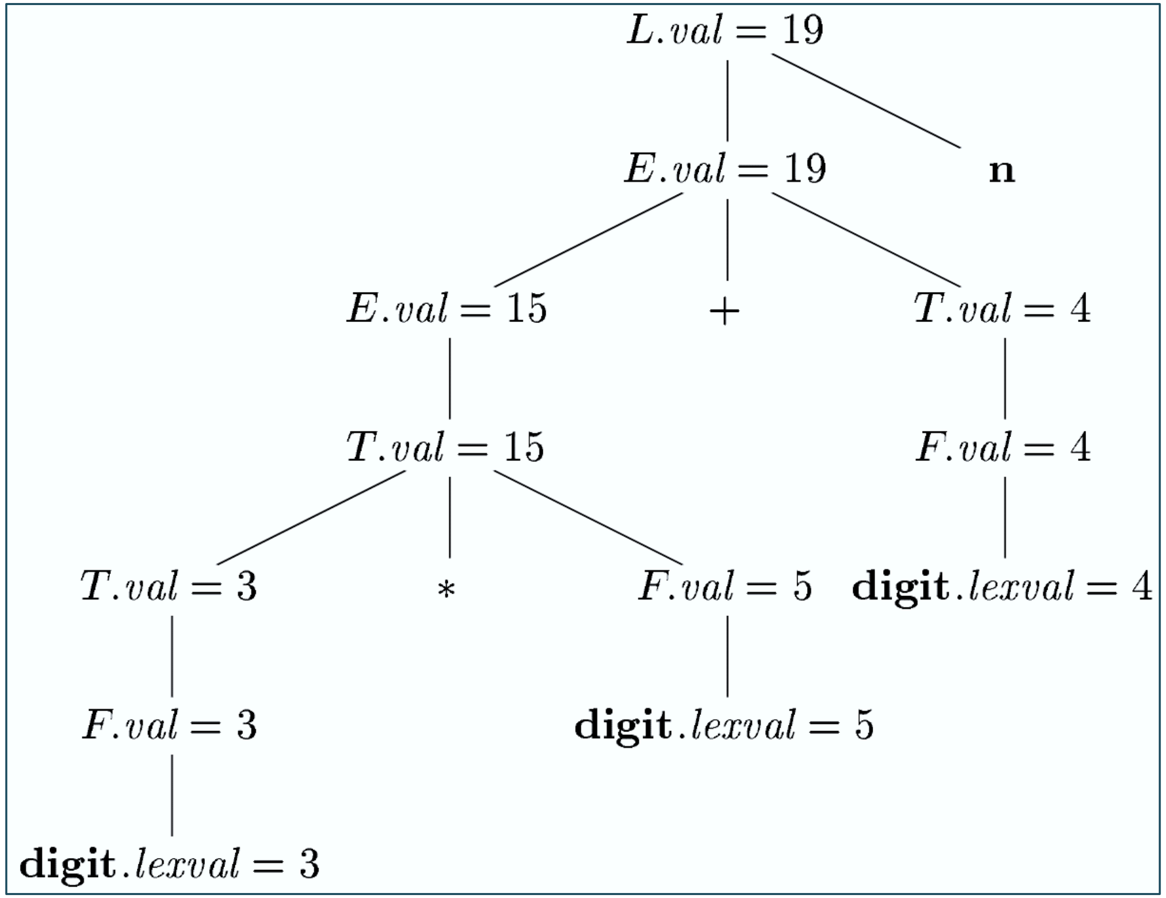
- A parse tree, showing the value(s) of its attribute(s) is called an ***annotated parse tree***
- Before we can evaluate an attribute at a node of a parse tree, we must evaluate all the attributes upon which its value depends
- With synthesized attributes, we can evaluate attributes in any bottom-up order, such as that of a postorder traversal of the parse tree
- For SDD's with both inherited and synthesized attributes, there is no guarantee that there is even one order in which to evaluate attributes at nodes
  - **Example**
    - The following rules are circular; it is impossible to evaluate either $A.s$ at a node N or $B.i$ at the child of N without first evaluating the other

PRODUCTION	SEMANTIC RULES
$A \to B$	$A.s = B.i;$ $B.i = A.s + 1$

Evaluating an SDD at the Nodes of a Parse Tree

- **Example**
  - Annotated parse tree for $3 * 5 + 4\ \boldsymbol{n}$

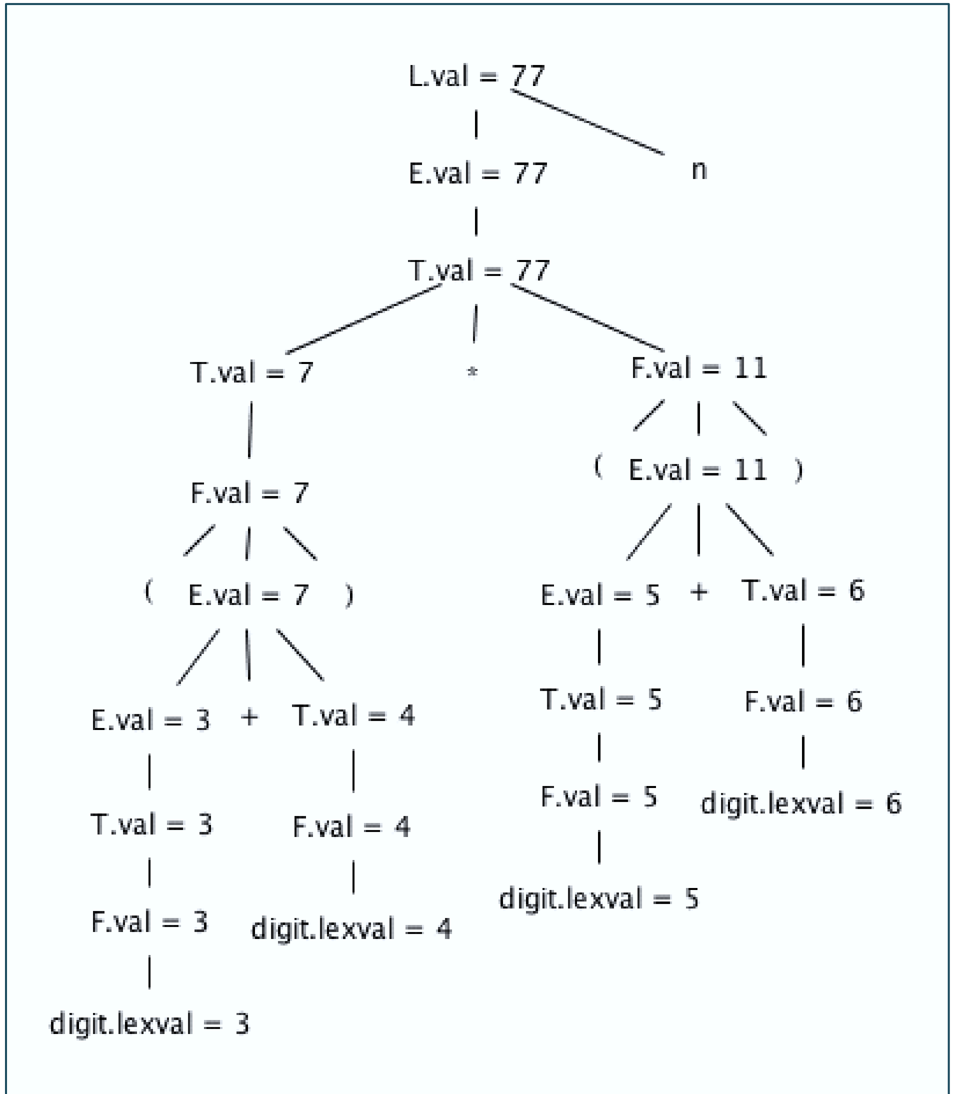
	PRODUCTION	SEMANTIC RULES
1)	$L \to E\ \mathbf{n}$	$L.val = E.val$
2)	$E \to E_1\ +\ T$	$E.val = E_1.val + T.val$
3)	$E \to T$	$E.val = T.val$
4)	$T \to T_1\ *\ F$	$T.val = T_1.val \times F.val$
5)	$T \to F$	$T.val = F.val$
6)	$F \to (\ E\ )$	$F.val = E.val$
7)	$F \to \mathbf{digit}$	$F.val = \mathbf{digit}.\text{lexval}$



Evaluating an SDD at the Nodes of a Parse Tree

- **Example**
  - Annotated parse tree for $(3 + 4) * (5 + 6)\ \boldsymbol{n}$

	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)	$F \to \mathbf{digit}$	$F.val = \mathbf{digit}.\text{lexval}$



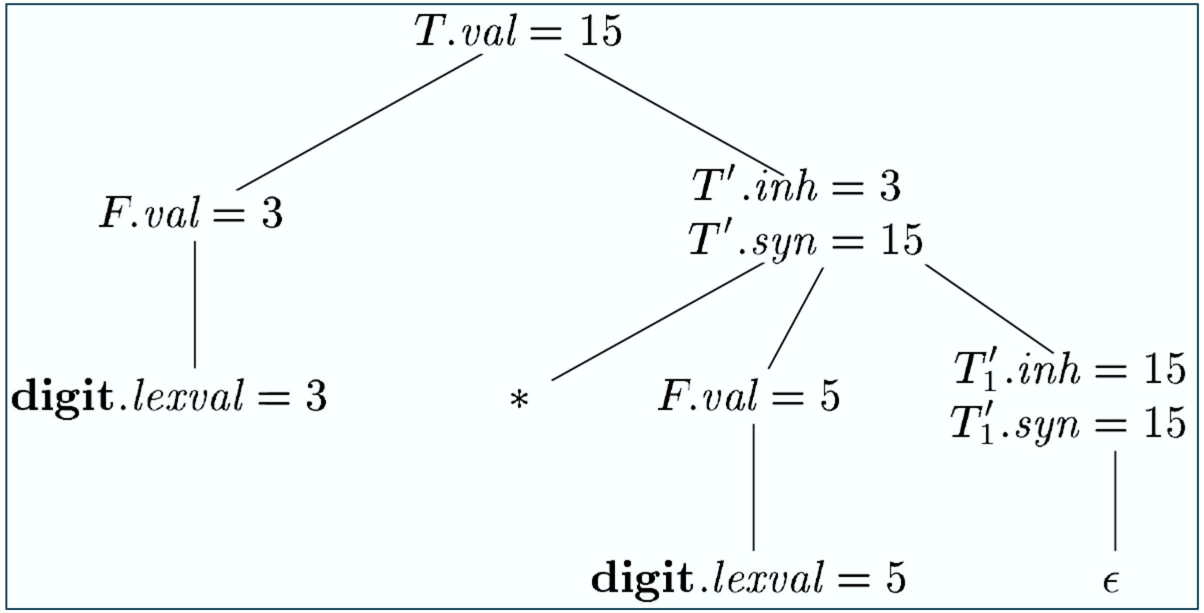
Evaluating an SDD at the Nodes of a Parse Tree

- **Example**
  - In the following SDD, the top-down parse of input $3 * 5$ begins with the production $T \rightarrow FT'$
  - 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

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) $F \rightarrow \textbf{digit}$	$F.val = \textbf{digit}.lexval$

Evaluating an SDD at the Nodes of a Parse Tree

- **Example**
  - The semantic rules are based on the idea that the left operand of the operator * is inherited
  - 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
  - Once all the factors have been accumulated, the result is passed back up the tree using synthesized attributes
  - **Annotated parse tree for $3 * 5$**

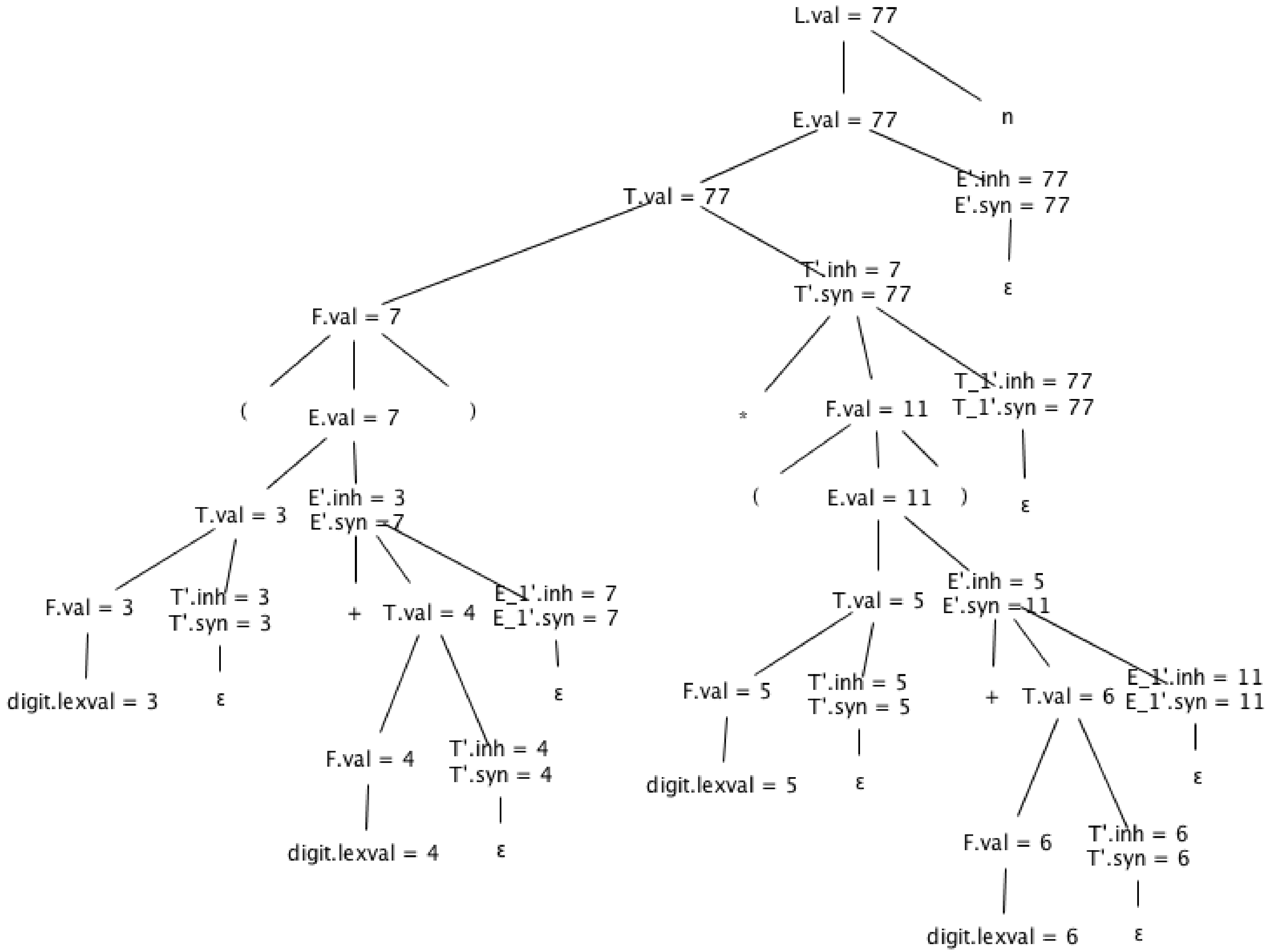


Evaluating an SDD at the Nodes of a Parse Tree

- **Exercise**

	Production	Semantic rules
	Production	**Semantic rules**
1)	$L \rightarrow E\ \boldsymbol{n}$	$L.val\ =\ E.val$
2)	$E -> TE'$	$E'.inh\ =\ T.val$
		$E.val\ =\ E'.syn$
3)	$E' -> +TE_1'$	$E_1'.inh = E'.inh\ +\ T.val$
		$E'.syn = E_1'.syn$
4)	$E' -> \varepsilon$	$E'.syn = E'.inh$
5)	$T -> FT'$	$T'.inh = F.val$
		$T.val\ =\ T'.syn$
6)	$T' -> *\ FT_1'$	$T_1'.inh\ = T'.inh * F.val$
		$T'.syn = T_1'.syn$
7)	$T' -> \varepsilon$	$T'.syn = T'.inh$
8)	$F -> (E)$	$F.val = E.val$
9)	$F -> digit$	$F.val = digit.lexval$

- **Exercise**
  - Annotated parse tree for the following expression, using the previous SDD
    - $(3 + 4) * (5 + 6)\ \boldsymbol{n}$



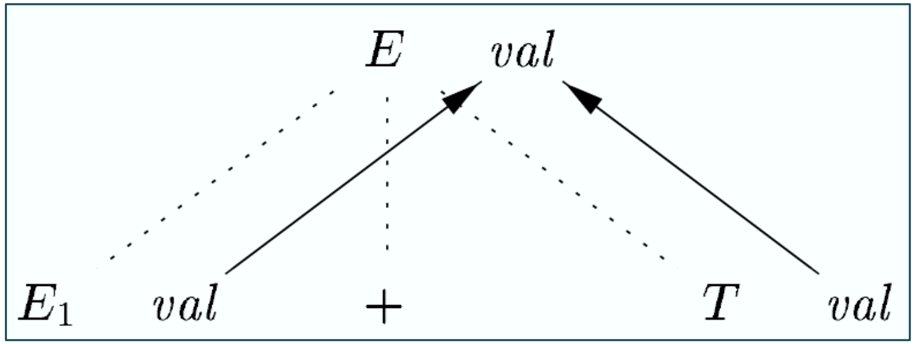
Dependency Graphs

- **Dependency graphs** are a useful tool for determining an evaluation order for the attribute instances in a given parse tree
- ***While an annotated parse tree shows the values of attributes, a dependency graph helps us determine how those values can be computed***
- 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

Dependency Graphs

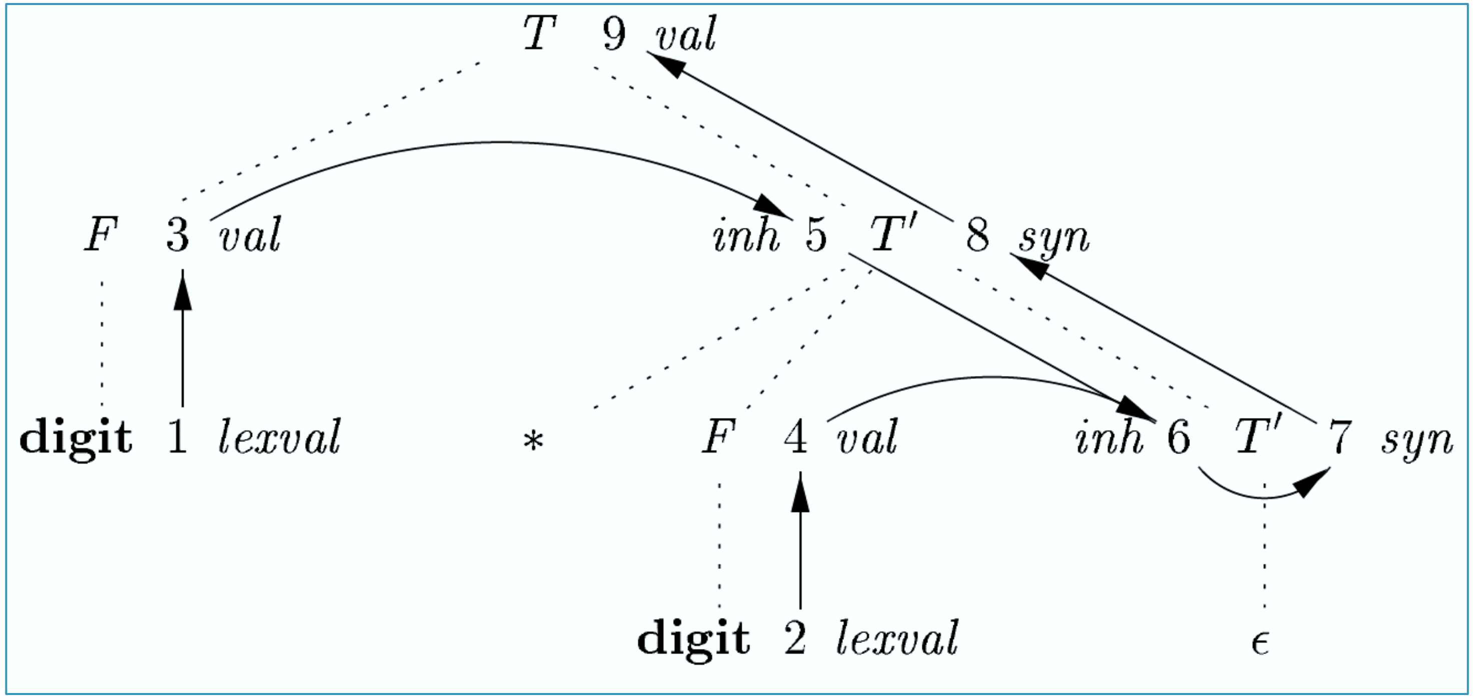
- **Example**

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



- **Example**

	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)	$F \rightarrow \textbf{digit}$	$F.val = \textbf{digit}.lexval$



Ordering the Evaluation of Attributes

- The dependency graph characterizes the possible orders in which we can evaluate the attributes at the various nodes of a parse tree
- Topological sorts
  - ***Sequences of nodes $N_1, N_2, \dots, N_k$ such that if there is an edge of the dependency graph from N_i to N_j, then $i < j$***
- **If there is any cycle in the graph**, then there are no topological sorts; that is, there is no way to evaluate the SDD on this parse tree

Ordering the Evaluation of Attributes

- **Example:** **All topological sorts of the following dependency graph**
  - [1, 2, 3, 4, 5, 6, 7, 8, 9]
  - [1, 2, 3, 5, 4, 6, 7, 8, 9]
  - [1, 2, 4, 3, 5, 6, 7, 8, 9]
  - [1, 3, 2, 4, 5, 6, 7, 8, 9]
  - [1, 3, 2, 5, 4, 6, 7, 8, 9]
  - [1, 3, 5, 2, 4, 6, 7, 8, 9]
  - [2, 1, 3, 4, 5, 6, 7, 8, 9]
  - [2, 1, 3, 5, 4, 6, 7, 8, 9]
  - [2, 1, 4, 3, 5, 6, 7, 8, 9]
  - [2, 4, 1, 3, 5, 6, 7, 8, 9]