CS 4031
Compiler Construction
Lecture 9

Mahzaib Younas

Lecturer, Department of Computer Science

FAST NUCES CFD

Semantic Analysis

- Semantics of a language provide meaning to its constructs like tokens and syntax structure.
- Semantic analysis judge whether the syntax structure constructed in the source program provide any meaning or not.
- For Example:
- Int a = “book”

is syntactically correct but generates a semantic error

- Solution:

CFG + Semantic Rules = Syntax Directed Definition

Semantic Analysis

- Output of syntax analysis which is parse tree is the impact of semantic analysis which is also a parse tree with some additional attributes called annotated parse tree.
- Some semantic rules are associated with the production rules in the semantic analysis like;

Semantic Analysis

1. We associate information with the programming language constructs by attaching attributes to grammar symbols.
2. Values of these attributes are evaluated by the **semantic rules** associated with the production rules.
3. Evaluation of these semantic rules:
   - may generate intermediate codes
   - may put information into the symbol table
   - may perform type checking
   - may issue error messages
   - may perform some other activities
   - in fact, they may perform almost any activities.
4. An attribute may hold almost any thing.
   - a string, a number, a memory location, a complex record.

Syntax Directed Translation (SDT)

- Syntax-directed translation (SDT) refers to a method of compiler implementation where the source language translation is completely driven by the parser, i.e., based on the syntax of the language.
- The parsing process and parse trees are used to direct semantic analysis and the translation of the source program.

SDT can be a separate phase of a compiler or we can augment our conventional grammar with information to control the semantic analysis and translation. This grammars are called attribute grammars.

Attribute Grammar

- We augment a grammar by associating attributes with each grammar symbol that describes its properties.
- With each production in a grammar, we give semantic rules/ actions, which describe how to compute the attribute values associated with each grammar symbol in a production.

Example of Semantic Analysi

$$\boxed{\begin{array}{lcl} E & \longrightarrow & E \mathbin{\texttt{*}} E \\ E & \longrightarrow & E \mathbin{\texttt{+}} E \\ E & \longrightarrow & \texttt{id} \end{array}}$$

$$\begin{array}{lcll} E & \longrightarrow & E_1 \mathbin{\texttt{*}} E_2 & \{E.val := E_1.val * E_2.val\} \\ E & \longrightarrow & E_1 \mathbin{\texttt{+}} E_2 & \{E.val := E_1.val + E_2.val\} \\ E & \longrightarrow & \texttt{int} & \{E.val := \texttt{int}.val\} \end{array}$$

Syntax Directed Translation and Translation Scheme

1. When we associate semantic rules with productions, we use two notations:
    - **Syntax-Directed Definitions**
    - **Translation Schemes**

A. Syntax-Directed Definitions:

- give high-level specifications for translations
- hide many implementation details such as order of evaluation of semantic actions.
- We associate a production rule with a set of semantic actions, and we do not say when they will be evaluated.

B. Syntax-Directed Translation Schemes:

- indicate the order of evaluation of semantic actions associated with a production rule.
- In other words, translation schemes give a little bit information about implementation details.

Syntax Directed Definition

- SDD) is a context-free grammar together with, attributes (values) and rules (Semantic 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.

Syntax Directed Definition

- 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.

Syntax Directed Definitions

Production	**Semantic Rules**
$L \rightarrow E\ \mathbf{n}$	print(E.val)
$E \rightarrow E_1 + T$	$E.val = E_1.val + T.val$
$E \rightarrow T$	$E.val = T.val$
$T \rightarrow T_1 * F$	$T.val = T_1.val * F.val$
$T \rightarrow F$	$T.val = F.val$
$F \rightarrow (E)$	$F.val = E.val$
$F \rightarrow \mathbf{digit}$	$F.val = \mathbf{digit}.lexval$

Attribute Grammar

- So, a semantic rule $b=f(c_1,c_2,\ldots,c_n)$ indicates that the attribute b *depends on* attributes $c_1,c_2,\ldots,c_n$.

- In a **syntax-directed definition**, a semantic rule may just evaluate a value of an attribute or it may have some side effects such as printing values.

- An **attribute grammar** is a syntax-directed definition in which the functions in the semantic rules cannot have side effects (they can only evaluate values of attributes).

Syntax Directed Translation

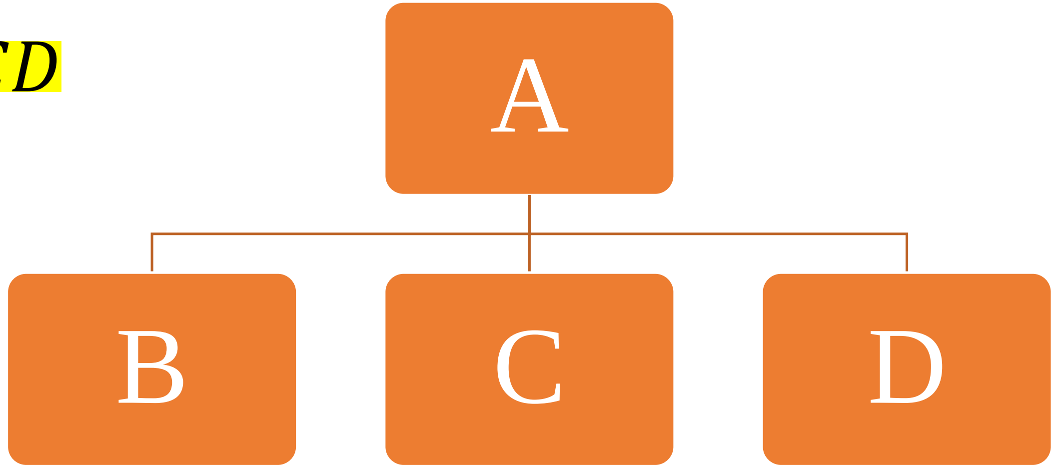
1. A syntax-directed definition is a generalization of a context-free grammar in which:
   - Each grammar symbol is associated with a set of attributes.
   - This set of attributes for a grammar symbol is partitioned into two subsets called
     - **synthesized** and
     - **inherited** attributes of that grammar symbol.
   - Each production rule is associated with a set of semantic rules.

2. The value of an attribute at a parse tree node is defined by the semantic rule associated with a production at that node.
3. The value of a **synthesized attribute** at a node is computed from the values of attributes at the children in that node of the parse tree
4. The value of an **inherited attribute** at a node is computed from the values of attributes at the siblings and parent of that node of the parse tree

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.
- A synthesized attribute at node N is defined only in terms of attribute values at the children of N and at N itself.

$$A \rightarrow BCD$$



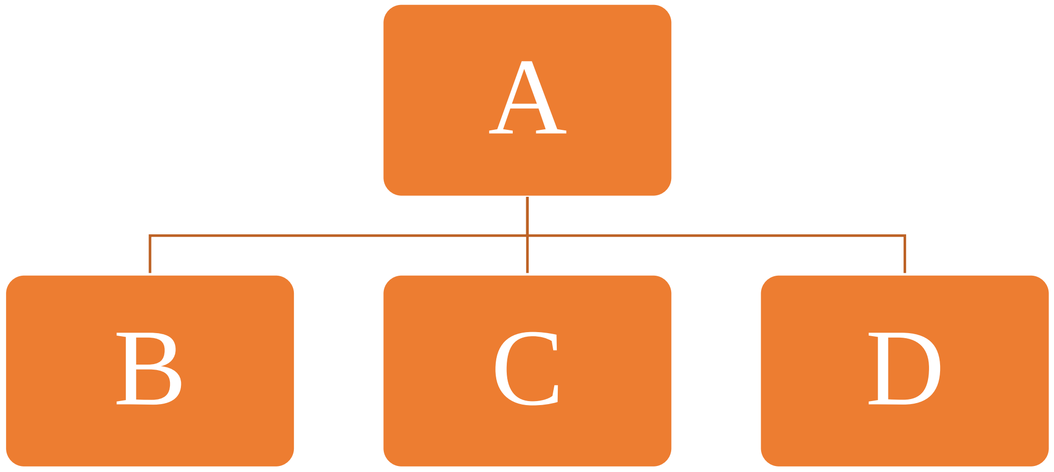
Synthesized Attribute

- The Parent node get the value from the child node

A.val = B.val

A.val = C.val

A.val = D.val



Inherited Attributes

- 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.
- An inherited attribute at node N is defined only in terms of attribute values at N's parent, N itself, and N's siblings

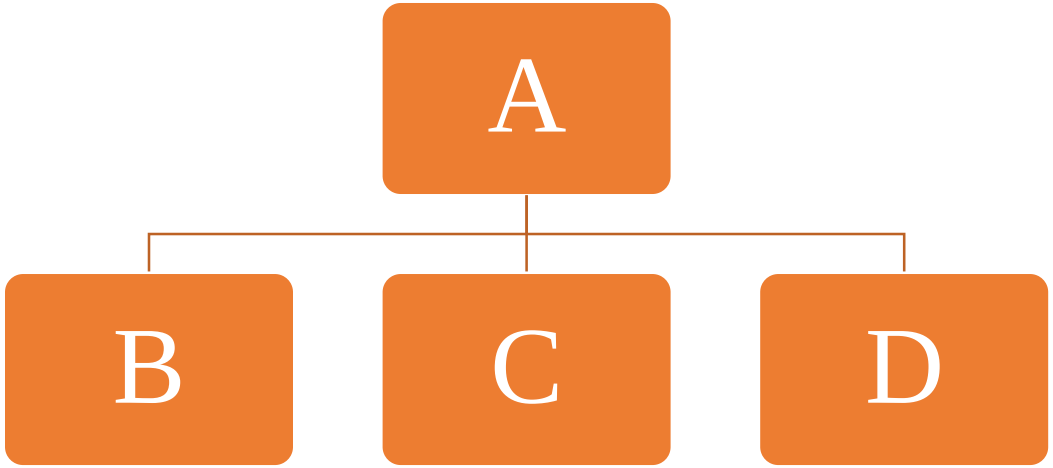
Inherited Attributes

Value of node can be assign by Parent and its siblings both.

C.val = A.val

C.val = B.val

C.val = D.val



Annotated Parse Tree

1. A parse tree showing the values of attributes at each node is called an **annotated parse tree**.
2. Values of Attributes in nodes of annotated parse-tree are either,
   - initialized to constant values or by the lexical analyzer.
   - determined by the semantic-rules.
3. The process of computing the attributes values at the nodes is called **annotating** (or **decorating**) of the parse tree.
4. Of course, the order of these computations depends on the dependency graph induced by the semantic rules.

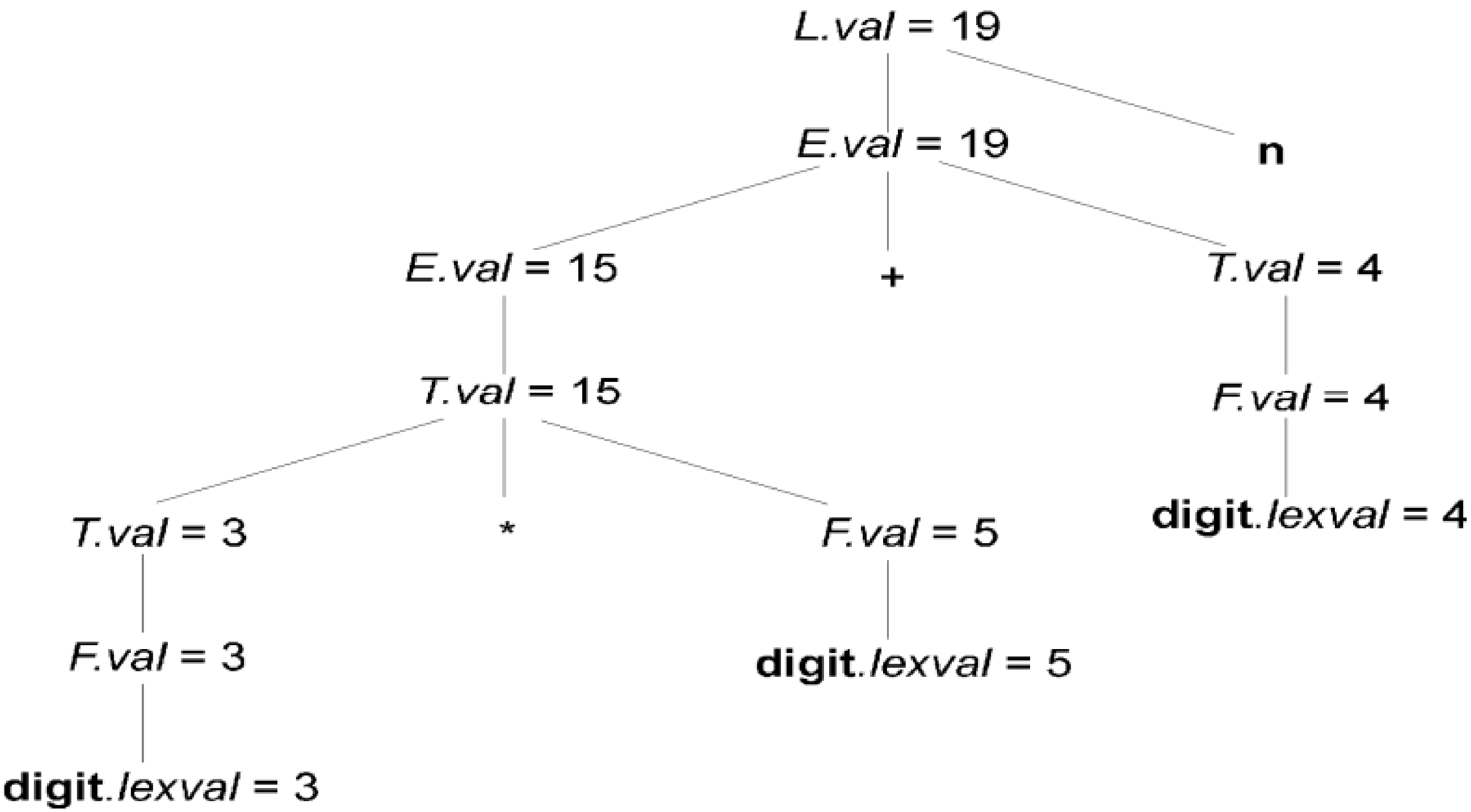
Syntax Directed Definitions

Production	**Semantic Rules**
$L \rightarrow E$ **n**	print(E.val)
$E \rightarrow E_1 + T$	$E.val = E_1.val + T.val$
$E \rightarrow T$	$E.val = T.val$
$T \rightarrow T_1 * F$	$T.val = T_1.val * F.val$
$T \rightarrow F$	$T.val = F.val$
$F \rightarrow (E)$	$F.val = E.val$
$F \rightarrow$ **digit**	F.val = **digit**.lexval

Example: Annotated Parse Tree = 3 * 5 + 4n

- Annotated Parse Tree = 3 * 5 + 4n
- Make the syntax tree by using bottom up approach.

Solution:

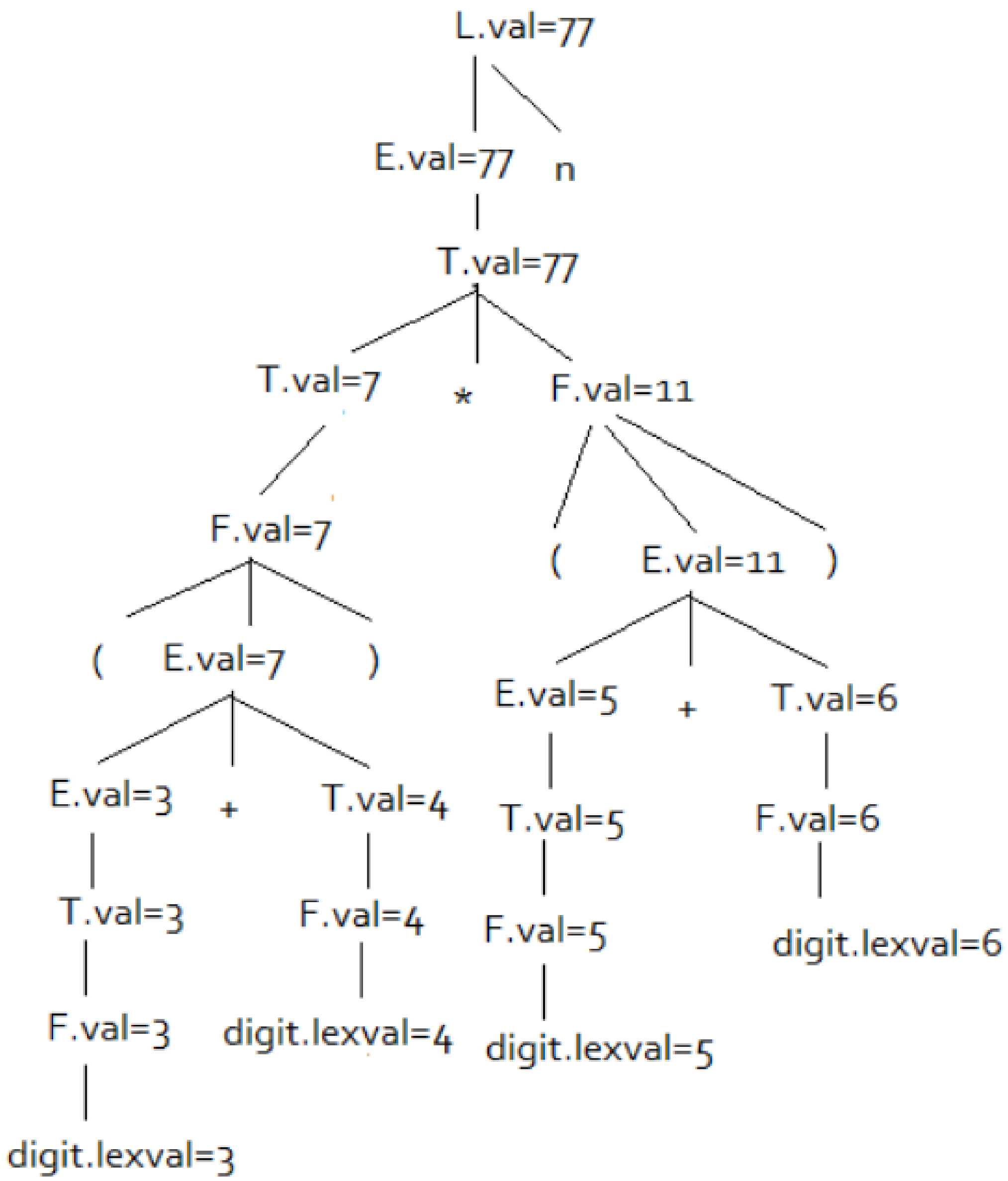


L.val = 19
E.val = 19
n
E.val = 15
+
T.val = 4
T.val = 15
F.val = 4
T.val = 3
*
F.val = 5
digit.lexval = 4
F.val = 3
digit.lexval = 5
digit.lexval = 3

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

- Annotated Parse Tree = (3 + 4) *(5 + 6)n
- Make the syntax tree by using bottom up approach.

Solution:



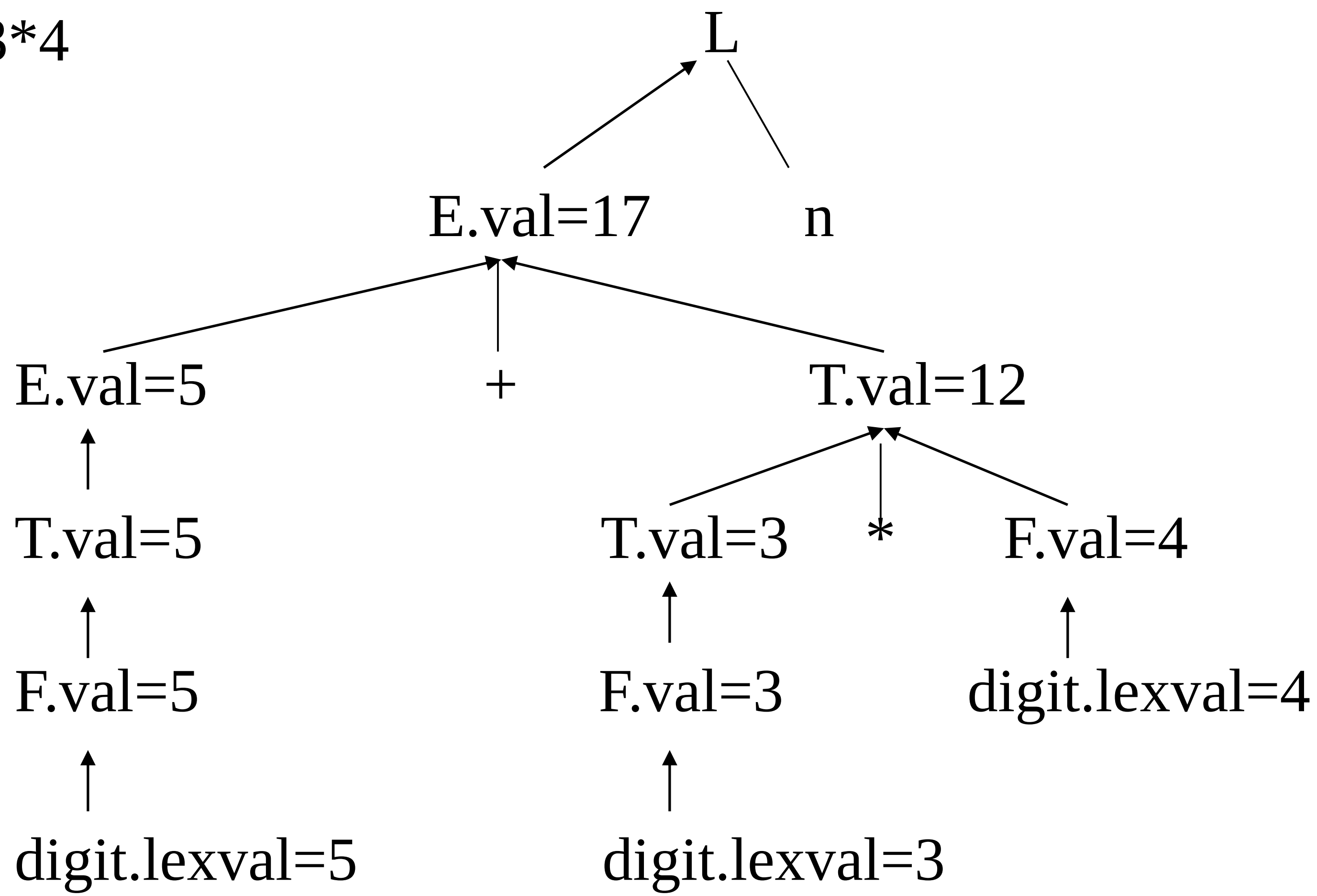
L.val=77
E.val=77
n
T.val=77
T.val=7
*
F.val=11
F.val=7
(
E.val=11
)
(
E.val=7
)
E.val=5
+
T.val=6
E.val=3
+
T.val=4
T.val=5
F.val=6
T.val=3
F.val=4
F.val=5
digit.lexval=6
F.val=3
digit.lexval=4
digit.lexval=5
digit.lexval=3

Dependency Graph

- A dependency graph characterizes the possible order in which we can evaluate the attributes at various nodes of a parse tree.
- If there is an edge from node M to N, then attribute corresponding to M first be evaluated before evaluating N.

Dependency Graph

Input: 5+3*4



Types of SDD

- There are two types of SDD

1. S-Attributed Grammar

   Synthesized Attributes which is also Known as S-Attributed Grammar.

2. L-Attributed Grammar

   A SDD that uses both synthesized and inherited attributes, then such grammar is known as L-attributed Grammar,

   Each inherited attribute is restricted to inherited from parent or left sibling only.

Attribute Flow

S-attributed grammar

- Uses only synthesized types
- Bottom-up attribute flow

L-attributed grammar

- Attributes can be evaluated in a single left-to-right pass over the input
- Each synthesized attribute of LHS depends only on that symbol's own inherited attributes or on attributes (synthesized or inherited) of the production's RHS symbols
- Top down Parser

Inherited Attributes

Production	**Semantic Rules**
$D \rightarrow T\ L$	$L.in = T.type$
$T \rightarrow$ **int**	$T.type = integer$
$T \rightarrow$ **real**	$T.type = real$
$L \rightarrow L_1$ **id**	$L_1.in = L.in$, addtype(**id**.entry,L.in)
$L \rightarrow$ **id**	addtype(**id**.entry,L.in)

1. Symbol T is associated with a synthesized attribute *type*.
2. Symbol L is associated with an inherited attribute *in*.

Example 3:

- Convert the given grammar into the Attributed grammar then make the annotated parse tree for 3*5,
- CFG:
  - $T \ \rightarrow FT$
  - $T` \ \rightarrow * \ FT_1`$
  - $T` \rightarrow \in$
  - $F \ \rightarrow digit$

Solution: Semantic Rules

Production	Semantic Rules
$T \longrightarrow F\ T'$	$T'.inh = F.val$ $T.val = T'.syn$
$T' \longrightarrow {*}F\ T'_1$	$T'_1.inh = T'.inh \times F.val$ $T'.syn = T'_1.syn$
$T' \longrightarrow \varepsilon$	$T'.syn = T'.inh$
$F \longrightarrow$ **digit**	$F.val =$ **digit**.$lexval$

Solution: Annotated Syntax Tree