Syntax Directed Translation

Part I

Syntax Directed Translation

Syntax = form, Semantics = meaning

- Technique used to build semantic information for large structures,
  - based on its syntax

- In other words… Translation of languages guided by the context-free grammars

The Essence of Syntax-Directed Translation

- The semantics (i.e., meaning) of the various constructs in the language is viewed as attributes of the corresponding grammar symbols.

- Example:

  sequence of characters 495

  – grammar symbol TOK_INT

  – meaning $\equiv$ integer 495

  – is an attribute of TOK_INT.

- Attributes are associated with Terminal as well as Nonterminal symbols.
- An attribute may hold almost any thing
  – a string, a number, a memory location, a complex record.

The Essence of Syntax-Directed Translation

- Values of these attributes are evaluated by the **semantic rules** associated with the production rules.

- 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.

The Essence of Syntax-Directed Translation

Production	**Semantic Actions**
$E \rightarrow E_1 + T$	$E.val = E_1.val + T.val$
$E \rightarrow T$	$E.val = T.val$
$T \rightarrow T_1 * F$	$T.val = T_1.val * F.val$
$T \rightarrow F$	$T.val = F.val$
$F \rightarrow num$	$F.val = value(num)$
$F \rightarrow (E)$	$F.val = E.val$

Rule = compute the value of the attribute 'val' at the parent by adding together the value of the attributes at two of the children

Syntax-Directed Definitions and Translation Schemes

- Two notations to associate semantic rules with productions

- **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.
  - More readable.
- **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.
  - More efficient.

Syntax-Directed Definitions

- 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.

Syntax-Directed Definition -- Example

Production	**Semantic Rules**
$L \rightarrow E$ **n**	L.val = 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

- Symbols E, T, and F are associated with a synthesized attribute *val*.
- The token **digit** has a synthesized attribute *lexval* (it is assumed that it is evaluated by the lexical analyzer).

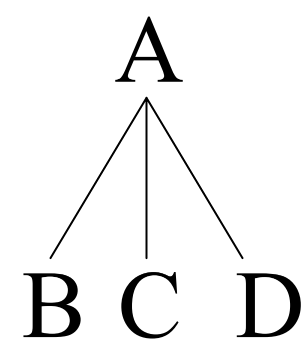
Synthesized Attributes

A synthesized attribute for a non-terminal 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.

OR

The value of a synthesized attribute for a node is computed using only information associated with the node and the node's children (or the lexical analyzer for leaf nodes).



Example:

Production	**Semantic Rules**
A → B C D	A.a := B.b + C.e

Example Problems for Synthesized

- Expression grammar – given a valid expression (ex: 1 * 2 + 3), determine the associated value while parsing.

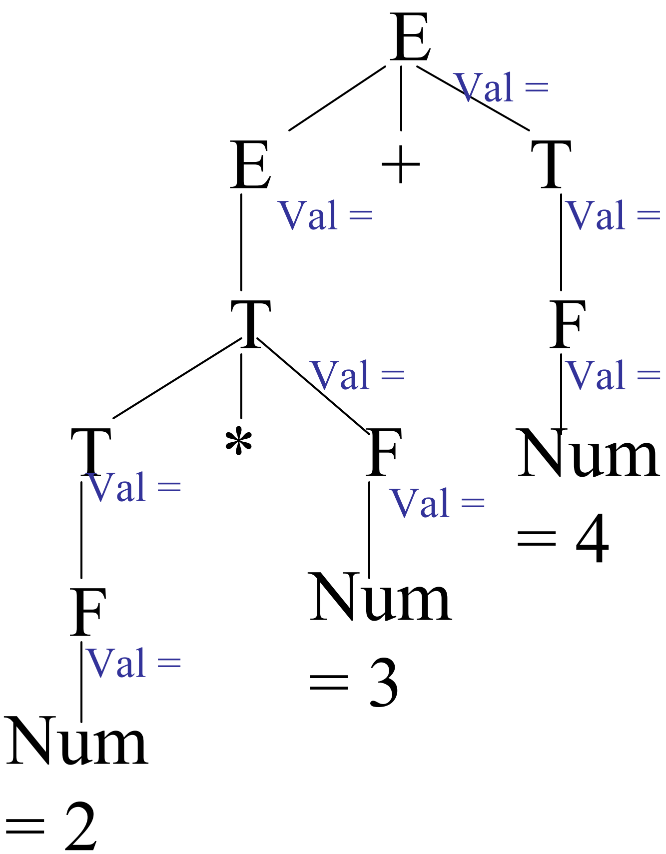
- Grid – Given a starting location of 0,0 and a sequence of north, south, east, west moves (ex: NESNNE), find the final position on a unit grid.

Synthesized Attributes – Expression Grammar

Production	Semantic Actions
$E \rightarrow E_1 + T$	$E.val = E_1.val + T.val$
$E \rightarrow T$	$E.val = T.val$
$T \rightarrow T_1 * F$	$T.val = T_1.val * F.val$
$T \rightarrow F$	$T.val = F.val$
$F \rightarrow num$	$F.val = value(num)$
$F \rightarrow (E)$	$F.val = E.val$

Synthesized Attributes –Annotating the parse tree

Production	Semantic Actions
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 → num	F.val = value(num)
F → (E)	F.val = E.val

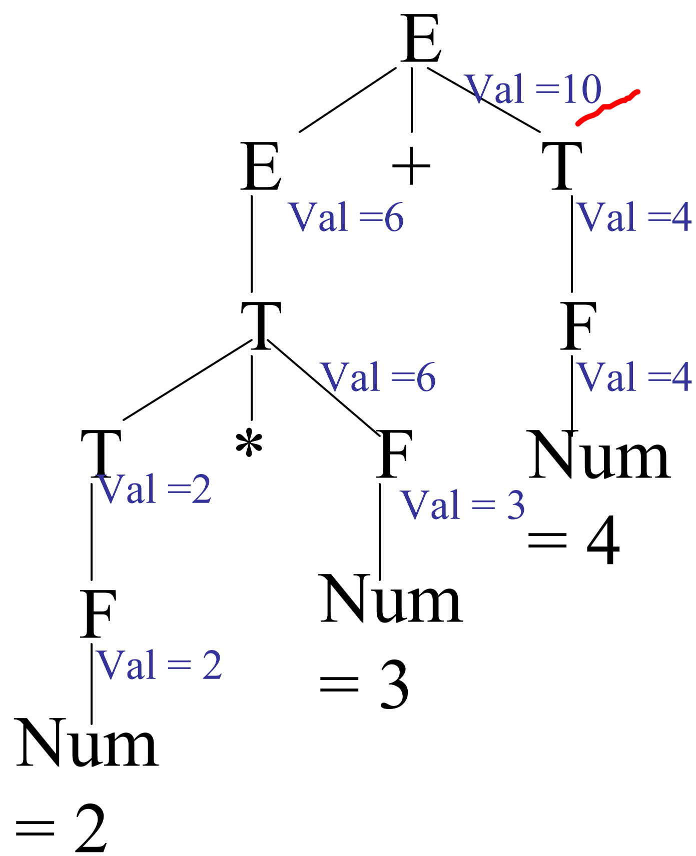


Input: 2 * 3 + 4

Synthesized Attributes –Annotating the parse tree

Production	Semantic Actions
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 → num	F.val = value(num)
F → (E)	F.val = E.val

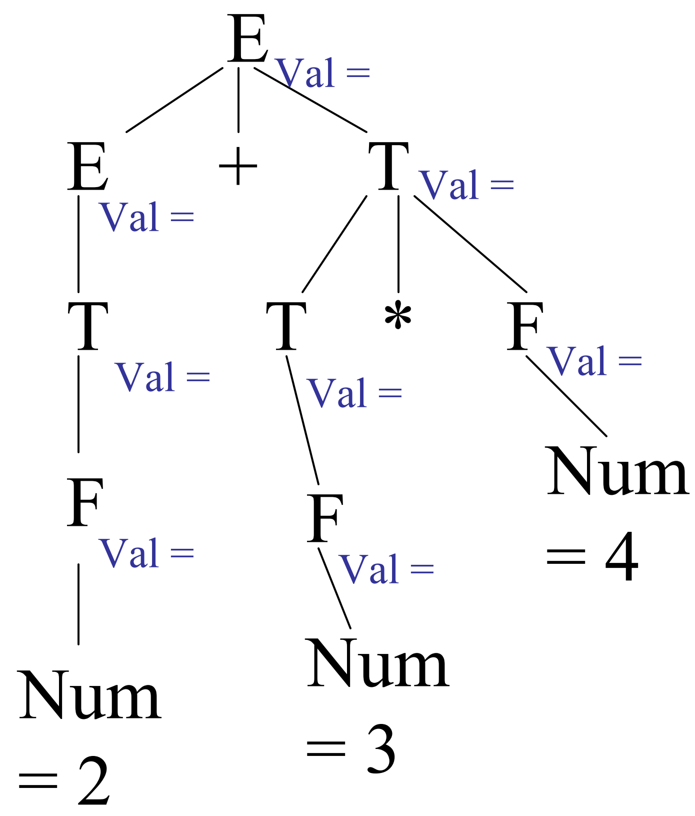
Input: 2 * 3 + 4



Synthesized Attributes –Annotating the parse tree

Production	Semantic Actions
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 → num	F.val = value(num)
F → (E)	F.val = E.val

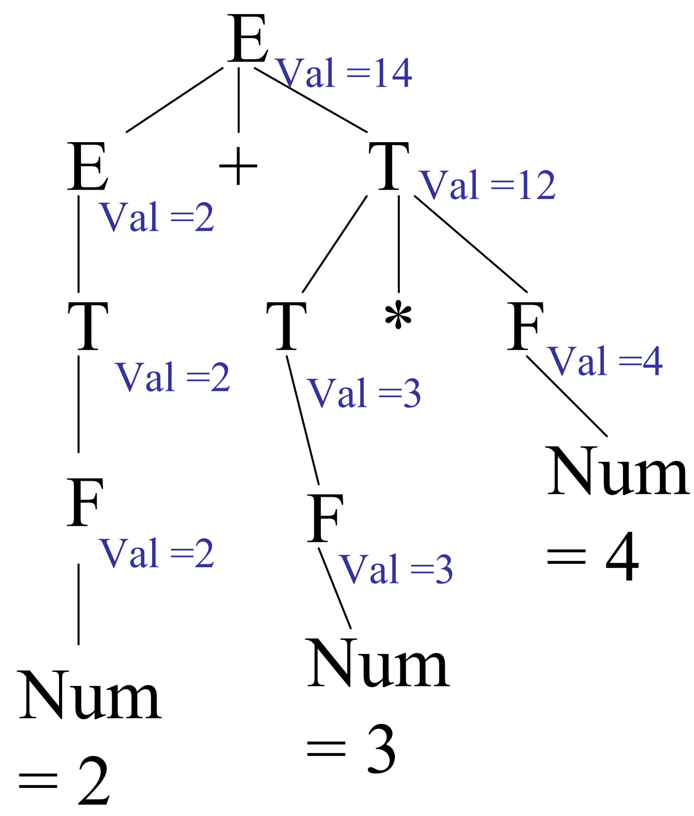
Input: 2 + 4 * 3



Synthesized Attributes –Annotating the parse tree

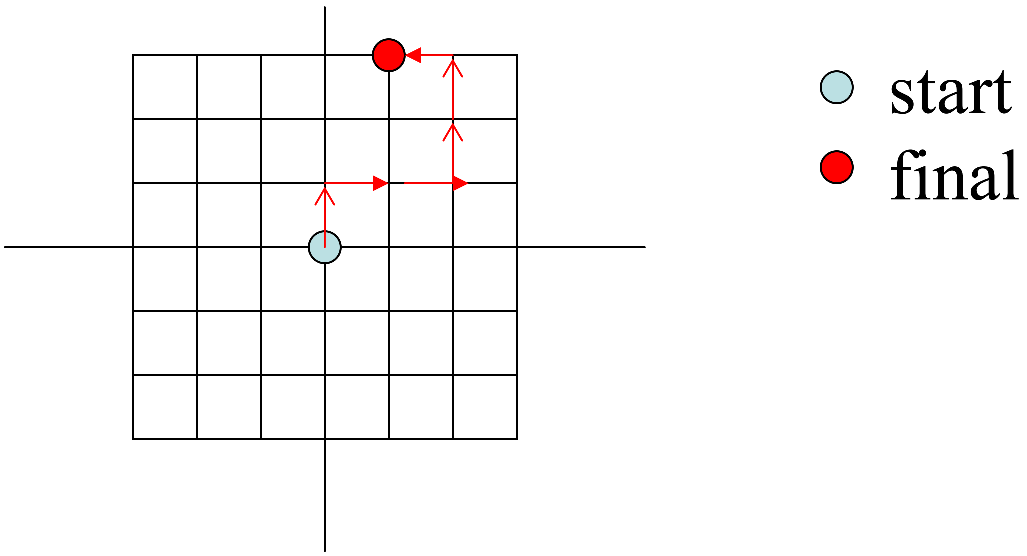
Production	Semantic Actions
E → E_1 + T	E.val = E_1.val + T.val
E → T	E.val = T.val
T → T_1 * F	T.val = T_1.val * F.val
T → F	T.val = F.val
F → num	F.val = value(num)
F → (E)	F.val = E.val

Input: 2 + 4 * 3



Grid Example

- Given a starting location of 0,0 and a sequence of north, south, east, west moves (ex: NEENNW), find the final position on a unit grid.



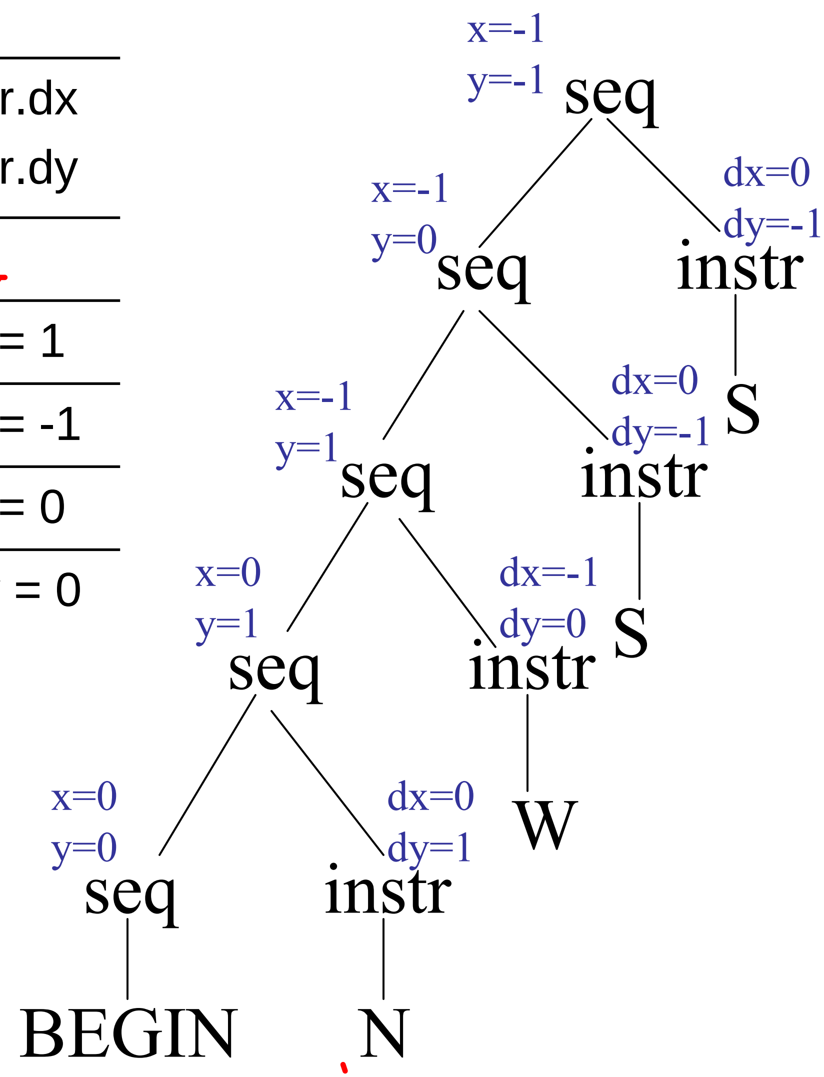
Synthesized Attributes – Grid Positions

Production	**Semantic Actions**
seq → seq_1 instr	$seq.x = seq_1.x + instr.dx$ $seq.y = seq_1.y + instr.dy$
seq → BEGIN	$seq.x = 0$, $seq.y = 0$
instr → NORTH	$instr.dx = 0$, $instr.dy = 1$
instr → SOUTH	$instr.dx = 0$, $instr.dy = -1$
instr → EAST	$instr.dx = 1$, $instr.dy = 0$
instr → WEST	$instr.dx = -1$, $instr.dy = 0$

Synthesized Attributes –Annotating the parse tree

Production	**Semantic Actions**
seq → seq_1 instr	$seq.x = seq_1.x + instr.dx$ $seq.y = seq_1.y + instr.dy$
seq → BEGIN	seq.x = 0, seq.y = 0
instr → NORTH	instr.dx = 0, instr.dy = 1
instr → SOUTH	instr.dx = 0, instr.dy = -1
instr → EAST	instr.dx = 1, instr.dy = 0
instr → WEST	instr.dx = -1, instr.dy = 0

Input: BEGIN N W S S



Inherited Attributes

if an attribute is not synthesized, it is inherited.

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

Example:

Production	**Semantic Rules**
A → B C D	B.b := A.a + C.b

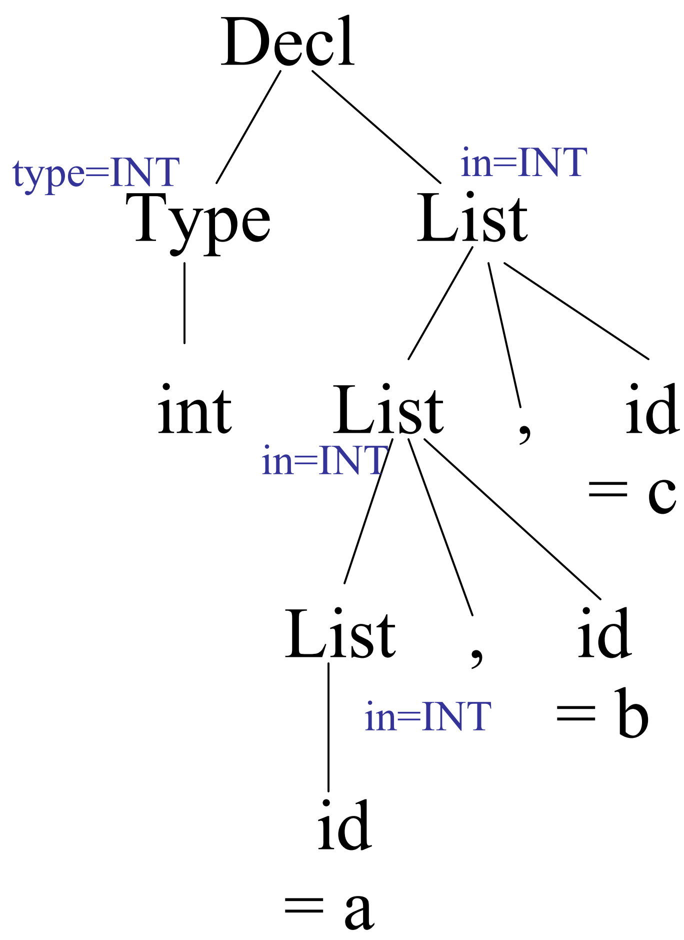
Inherited Attributes – Determining types

Productions	**Semantic Actions**
Decl → Type List	List.in = Type.type
Type → int	Type.type = INT
Type → real	T.type = REAL
List → $List_1$, id	$List_1$.in = List.in, addtype(id.entry.List.in)
List → id	addtype(id.entry,List.in)

Inherited Attributes – Example

Productions	**Semantic Actions**
Decl → Type List	List.in = Type.type
Type → int	Type.type = INT
Type → real	T.type = REAL
List → $List_1$, id	$List_1$.in = List.in, addtype(id.entry.List.in)
List → id	addtype(id.entry,List.in)

Input: int a,b,c



Syntax-Directed Definitions

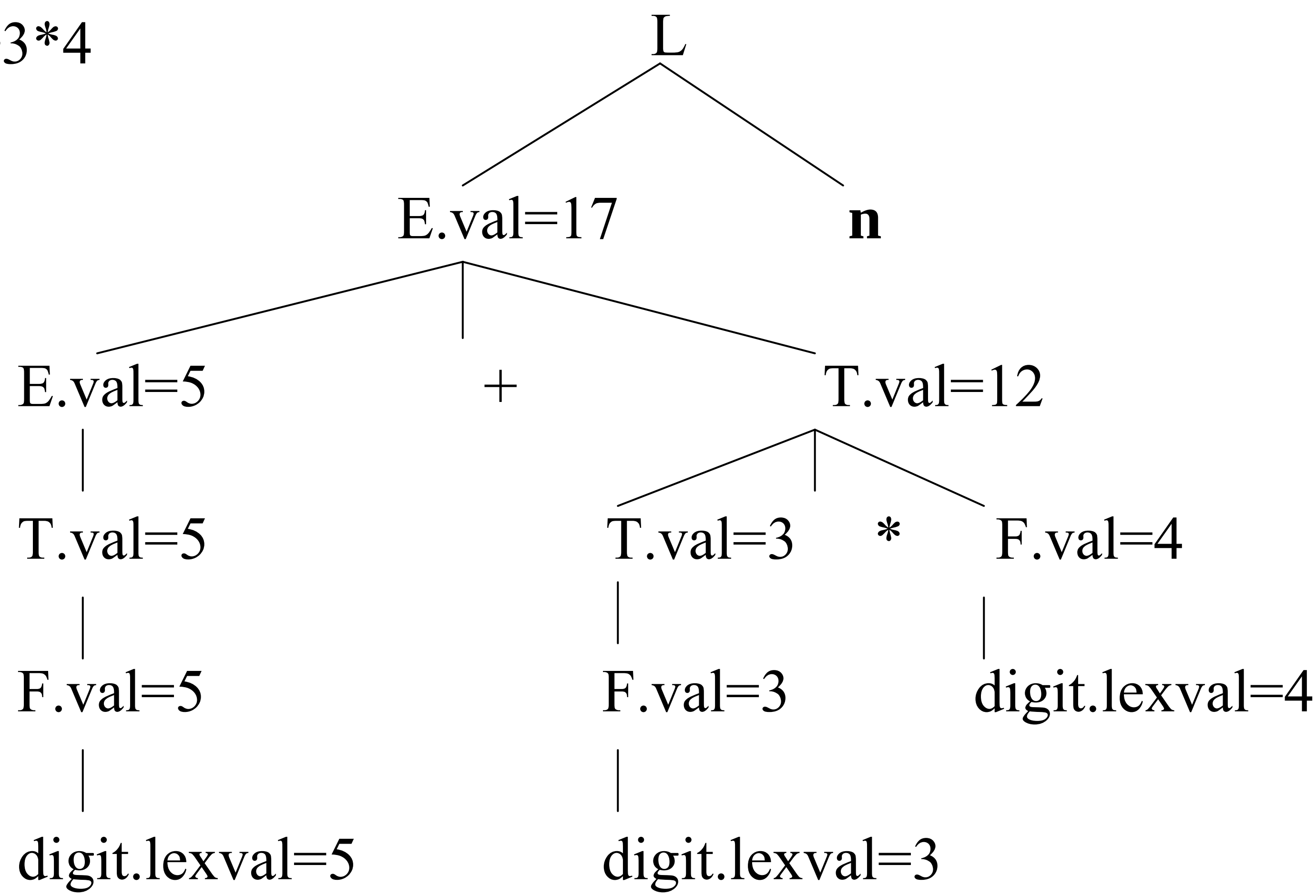
- *Semantic rules* set up dependencies between attributes which can be represented by a *dependency graph*.
- This *dependency graph* determines the evaluation order of these semantic rules.
- Evaluation of a semantic rule defines the value of an attribute. But a semantic rule may also have some side effects such as printing a value.

Annotated Parse Tree

- A parse tree showing the values of attributes at each node is called an **annotated parse tree**.
- The process of computing the attributes values at the nodes is called **annotating** (or **decorating**) of the parse tree.
- Of course, the order of these computations depends on the dependency graph induced by the semantic rules.

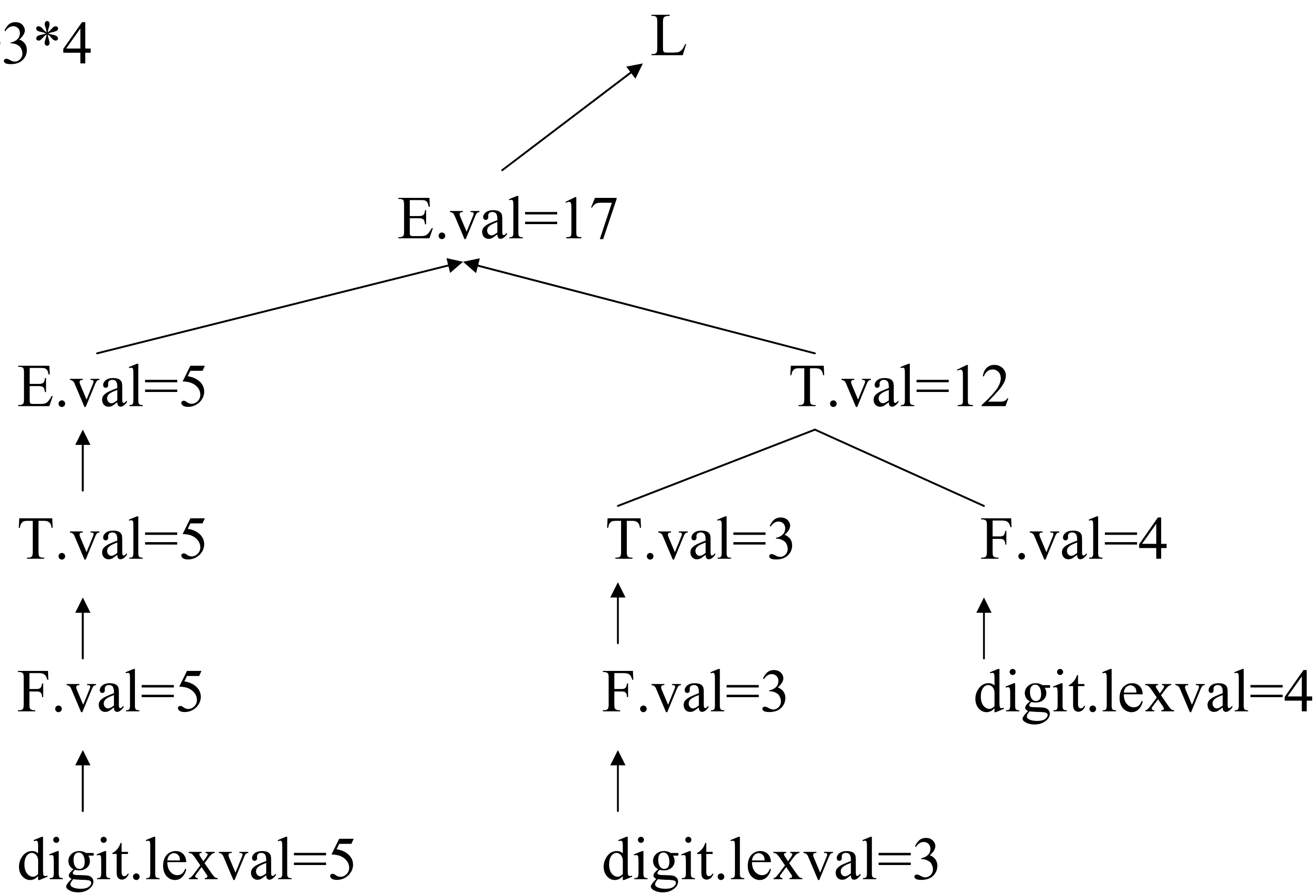
Annotated Parse Tree -- Example

Input: 5+3*4



Dependency Graph

Input: 5+3*4



Syntax-Directed Definition (SDD)

- In a syntax-directed definition, each production $A \rightarrow \alpha$ is associated with a set of semantic rules of the form:

  $b=f(c_1,c_2,\ldots,c_n)$ where f is a function,

  and b can be one of the followings:

  ➔ b is a synthesized attribute of A and $c_1,c_2,\ldots,c_n$ are attributes of the grammar symbols in the production ($A \rightarrow \alpha$).

  OR

  ➔ b is an inherited attribute one of the grammar symbols in α (on the right side of the production), and $c_1,c_2,\ldots,c_n$ are attributes of the grammar symbols in the production ($A \rightarrow \alpha$).

Attribute Grammar

- So, a semantic rule $b=f(c_1,c_2,\ldots,c_n)$ indicates that the attribute b *depends* o*n* 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 Definition – Example2

Production	**Semantic Rules**
$E \rightarrow E_1 + T$	E.loc=newtemp(), E.code = E_1.code || T.code || add E_1.loc,T.loc,E.loc
$E \rightarrow T$	E.loc = T.loc, E.code=T.code
$T \rightarrow T_1 * F$	T.loc=newtemp(), T.code = T_1.code || F.code || mult T_1.loc,F.loc,T.loc
$T \rightarrow F$	T.loc = F.loc, T.code=F.code
$F \rightarrow (E)$	F.loc = E.loc, F.code=E.code
$F \rightarrow$ **id**	F.loc = **id**.name, F.code=“”

- Symbols E, T, and F are associated with synthesized attributes *loc* and *code*.
- The token **id** has a synthesized attribute *name* (it is assumed that it is evaluated by the lexical analyzer).
- It is assumed that || is the string concatenation operator.

Syntax-Directed Definition – 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)

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

A Dependency Graph – Inherited Attributes