Syntax-Directed Translation

Muhammad Haggag, Ph.D.
Computer Science Department
Faculty of Computers and Information
Mansoura University
dr_m.haggag@yahoo.com

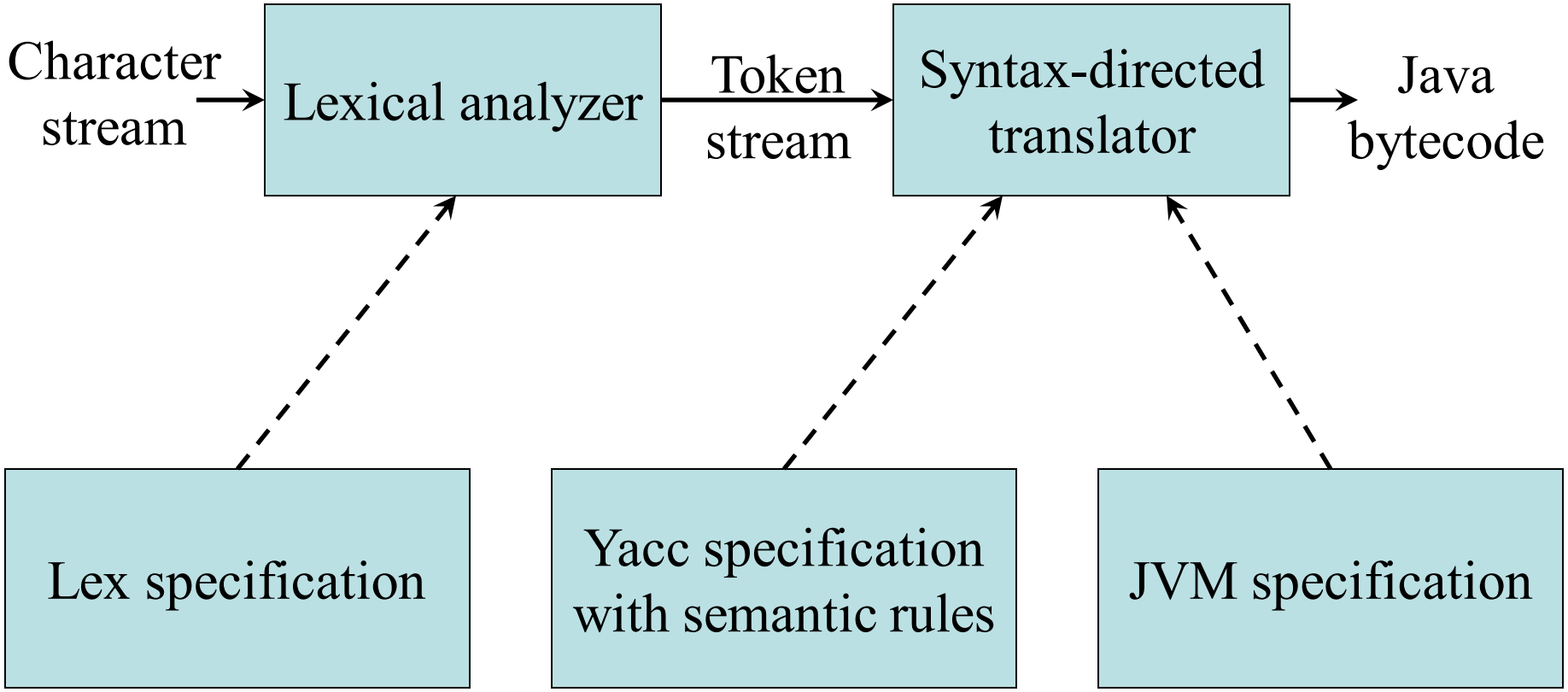
SDT

- Syntax-directed translation refers to a method of compiler implementation where the source language translation is completely driven by the parser.

- A common method of syntax-directed translation is translating a string into a sequence of actions by attaching one such **action** to each **rule of a grammar**. Thus, parsing a string of the grammar produces a sequence of rule applications. **SDT** provides a simple way to attach semantics to any such syntax.

- Interleave semantic analysis with syntax analysis phase of the compiler, we use **Syntax Directed Translation.**

The Structure of our Compiler Revisited



Syntax-Directed Definitions

- A *syntax-directed definition* (or *attribute grammar*) binds a set of *semantic rules* to productions (grammar)
- Actions (rules) are steps or procedures that will be carried out when that production is used in a derivation (usually evaluate attributes)
- Terminals and nonterminals have *attributes* holding values set by the semantic rules
- A *depth-first traversal* algorithm traverses the parse tree thereby executing semantic rules to assign attribute values

Attribute Grammars

- An **attribute** can represent anything we choose
  - The value of an expression when literal constants are used
  - The data type of a constant, variable, or expression
  - The location (or offset) of a variable in memory
  - The translated code of an expression, statement, or function

- Given a symbol *X*, with an attribute *t*, that attribute is referred to as *X.t*

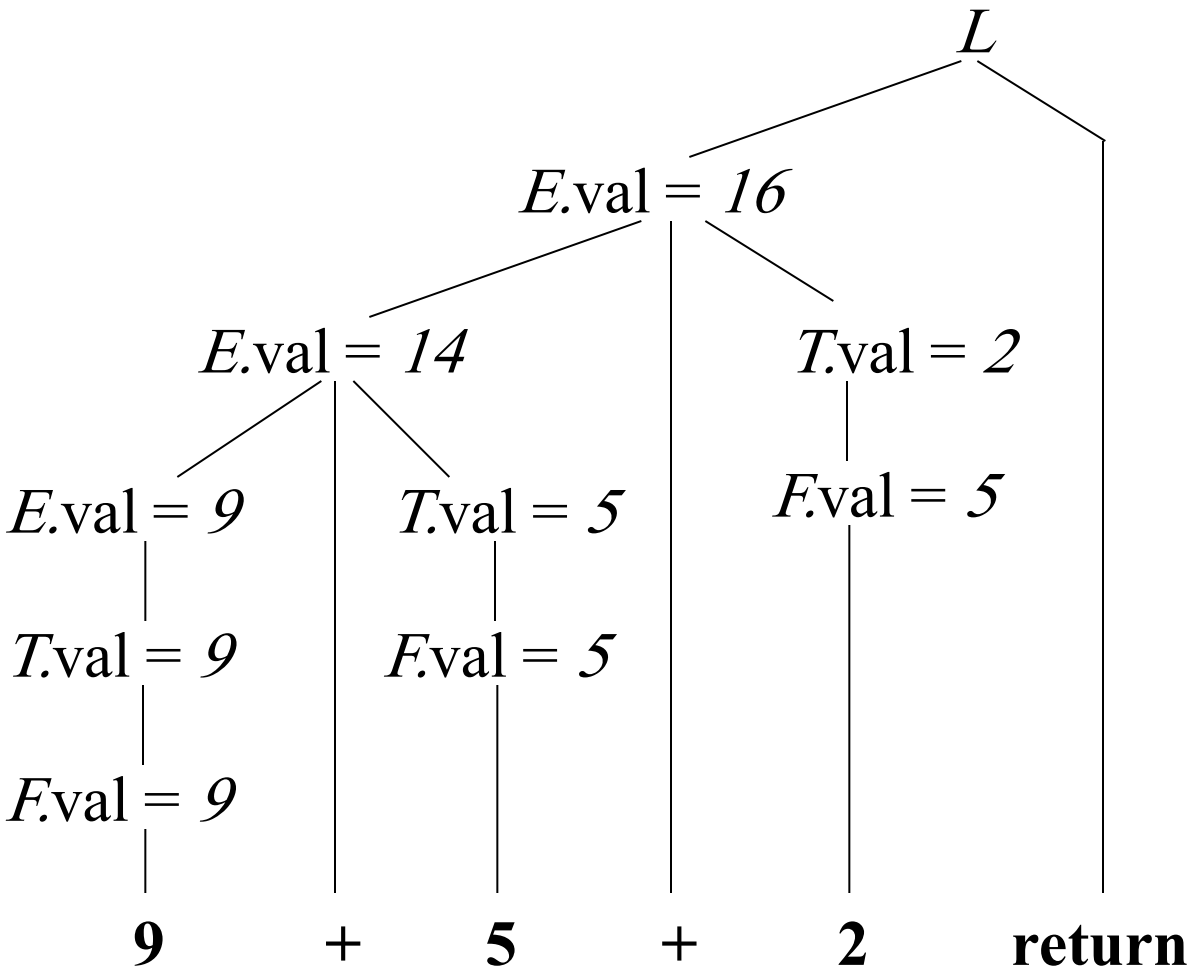
- An **annotated** or **attributed** parse tree is a
  - Parse tree showing the values of attributes at each node
  - Attributes may be evaluated on the fly as an input is parsed
  - Alternatively, attributes may be also evaluated after parsing

Example Attribute Grammar

Production	Semantic Rule
$L \rightarrow E$ **return**	*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

Note: all attributes in this example are of the **synthesized** type

Example Annotated Parse Tree

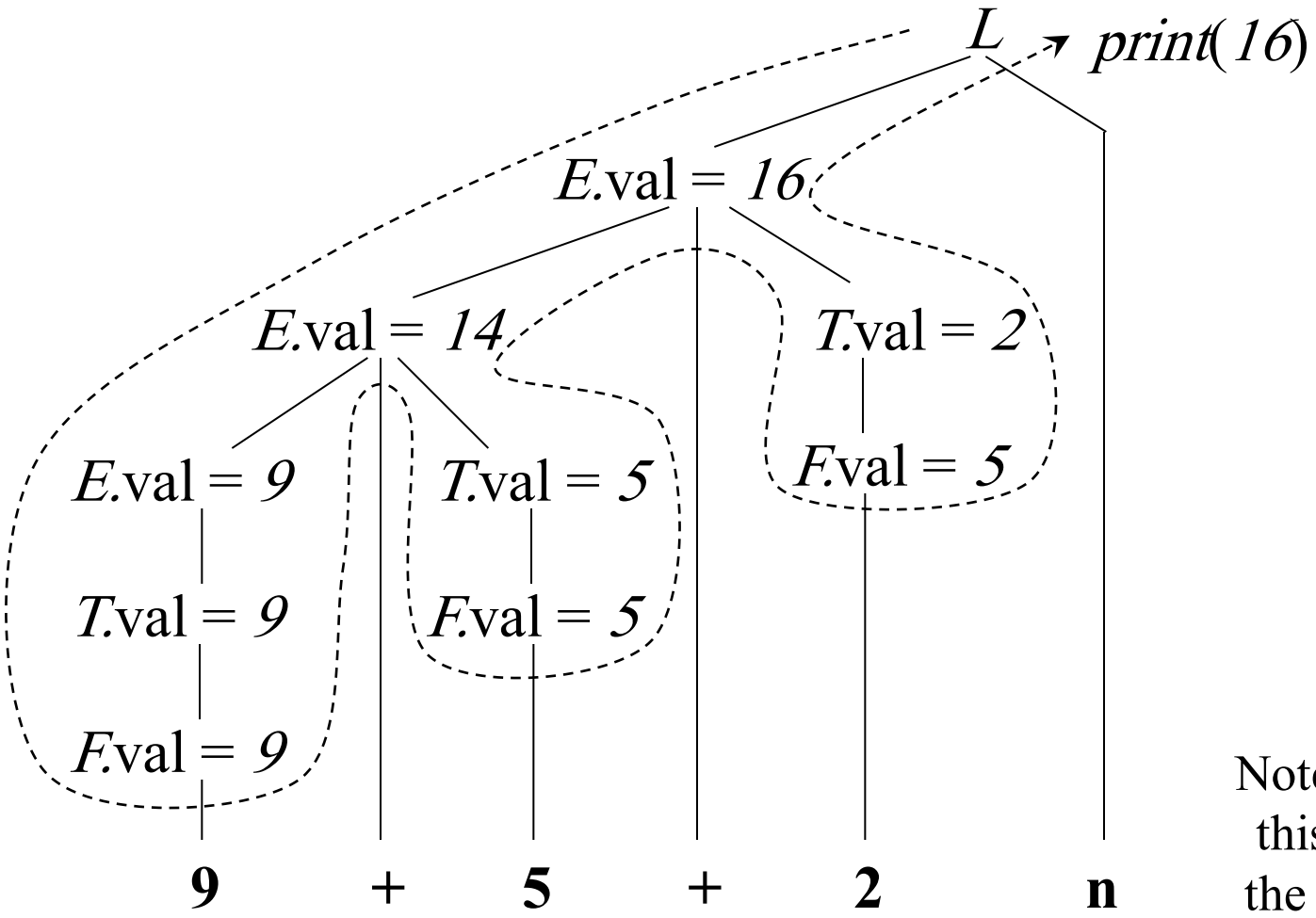


Note: all attributes in this example are of the synthesized type

Annotating a Parse Tree With Depth-First Traversals

```
procedure visit(n : node);
begin
    for each child m of n, from left to right do
        visit(m);
    evaluate semantic rules at node n
end
```

Depth-First Traversals (Example)



Note: all attributes in this example are of the synthesized type

Attributes (revisit)

- Attribute values may represent
  - Numbers (literal constants)
  - Strings (literal constants)
  - Memory locations, such as a frame index of a local variable or function argument
  - A data type for type checking of expressions
  - Scoping information for local declarations

Synthesized and Inherited Attributes

- The attributes are divided into two classes:
  - **Synthesized** Attributes
  - **Inherited** Attributes
- A **synthesized attribute** of a parse tree node is computed from
  - Attribute values of the **children nodes**
- An **inherited attribute** of a parse tree node is computed from
  - Attribute values of the **parent node**
  - Attribute values of the **sibling nodes**
- Tokens may have only synthesized attributes
  - Token attributes are supplied by the scanner
- Nonterminals may have synthesized and/or inherited attributes
- Attributes are evaluated according to **Semantic rules**
  - Semantic rules are associated with production rules

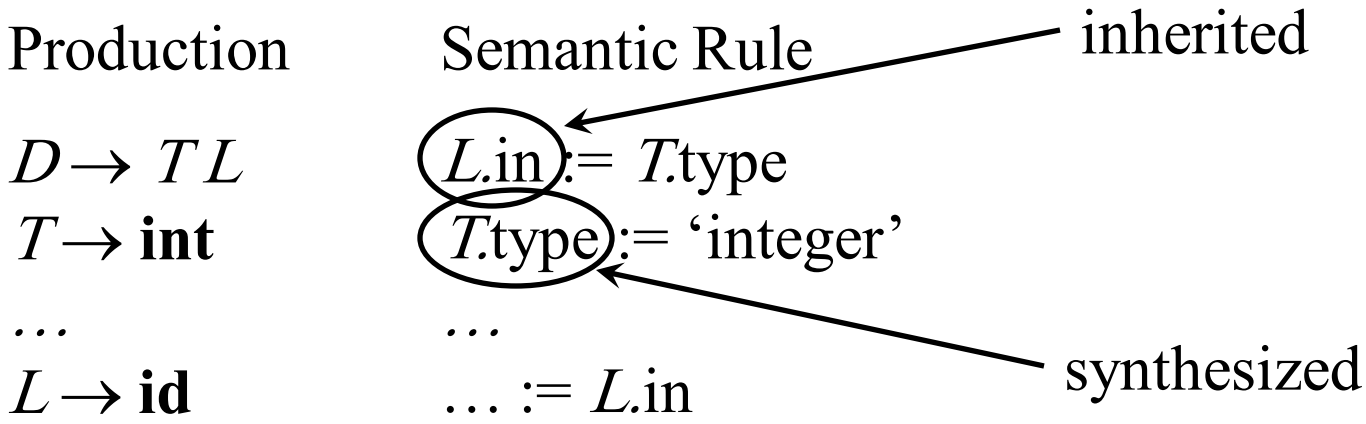
Synthesized Versus Inherited Attributes

- Given a production
  $A \rightarrow \alpha$
  then each semantic rule is of the form
  $b := f(c_1, c_2, \ldots, c_k)$

  where f is a function and c_i are attributes of A and α, and either
  - b is a *synthesized* attribute of A
  - b is an *inherited* attribute of one of the grammar symbols in α

- Attribute **b** depends on attributes $\boldsymbol{c_1, c_2, \ldots, c_k}$

Synthesized Versus Inherited Attributes (cont'd)

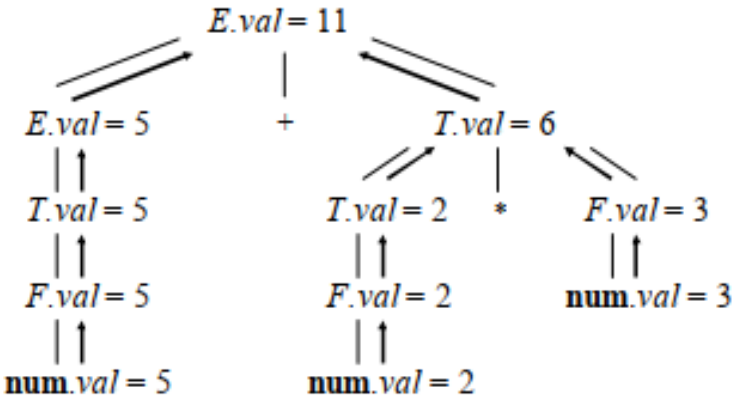


Symbol T is associated with a synthesized attribute **type**
Symbol L is associated with an inherited attribute **in**

S-Attributed Grammars

- S-Attributed grammars allow only synthesized attributes
- Synthesized attributes are evaluated bottom up
- S-Attributed grammars work perfectly with LR parsers
- Consider an S-Attributed grammar for constant expressions:
  - Each nonterminal has a single synthetic attribute: *val*
  - The annotated parse tree for **5 + 2 * 3** is shown below

Production	Semantic Rules
$E \rightarrow E^2 + T$	$E.val := E^2.val + T.val$
$E \rightarrow T$	$E.val := T.val$
$T \rightarrow T^2 * F$	$T.val := T^2.val * F.val$
$T \rightarrow F$	$T.val := F.val$
$F \rightarrow (E)$	$F.val := E.val$
$F \rightarrow$ **num**	$F.val :=$ **num**.*val*

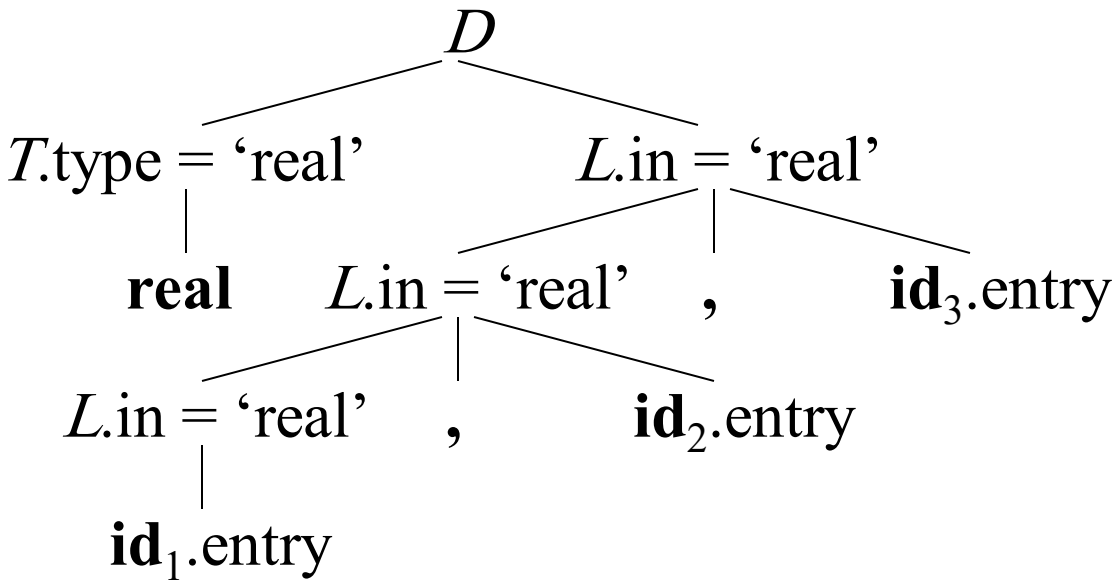


Example Attribute Grammar with Synthesized+Inherited Attributes

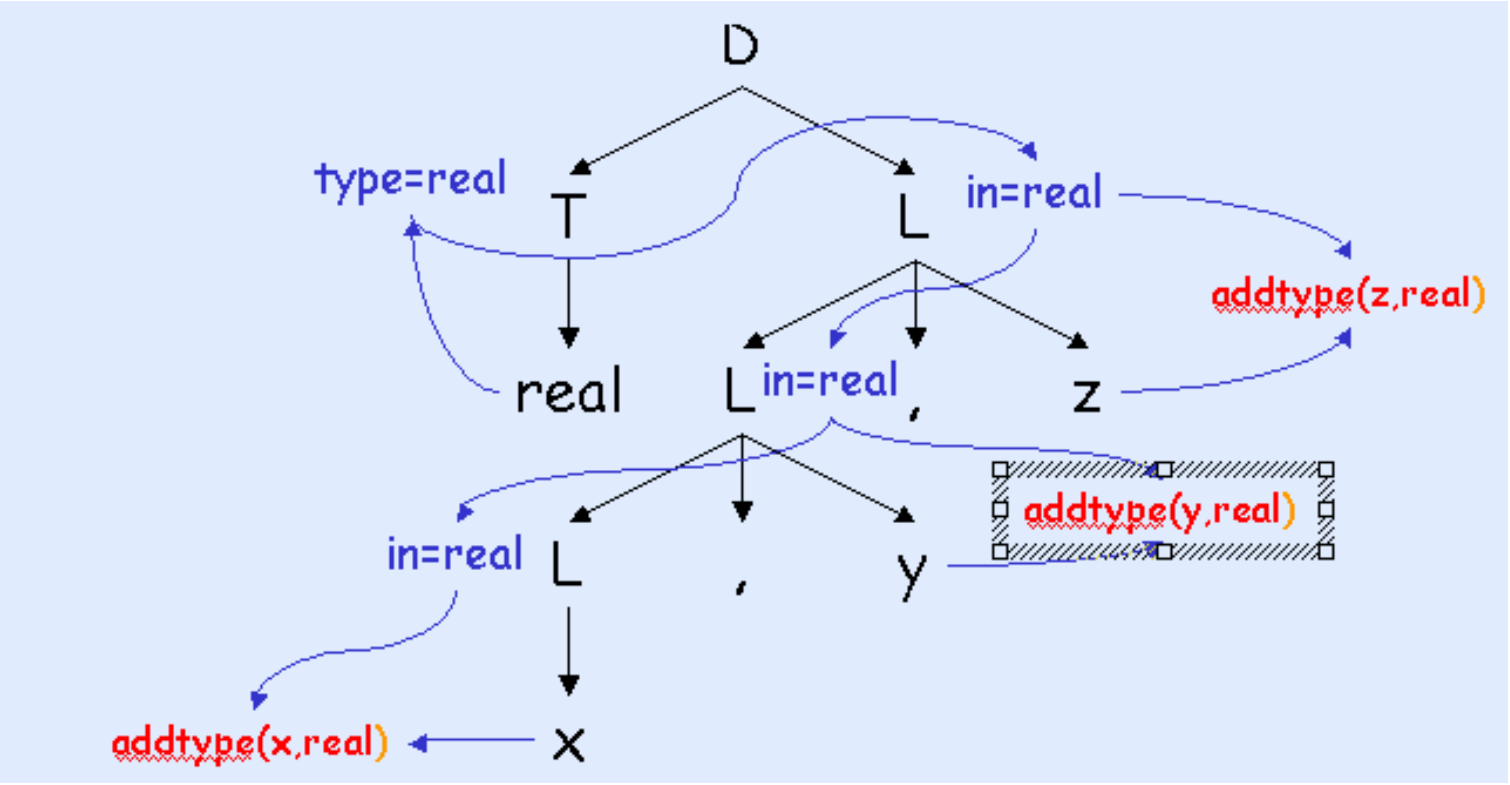
Production	Semantic Rule
$D \rightarrow T\ L$	L.in := T.type
$T \rightarrow$ **int**	T.type := 'integer'
$T \rightarrow$ **real**	T.type := 'real'
$L \rightarrow L_1$ **, id**	L_1.in := L.in; *addtype*(**id**.entry, L.in)
$L \rightarrow$ **id**	*addtype*(**id**.entry, L.in)

Synthesized: T.type, **id**.entry
Inherited: L.in

Example Annotated Parse Tree

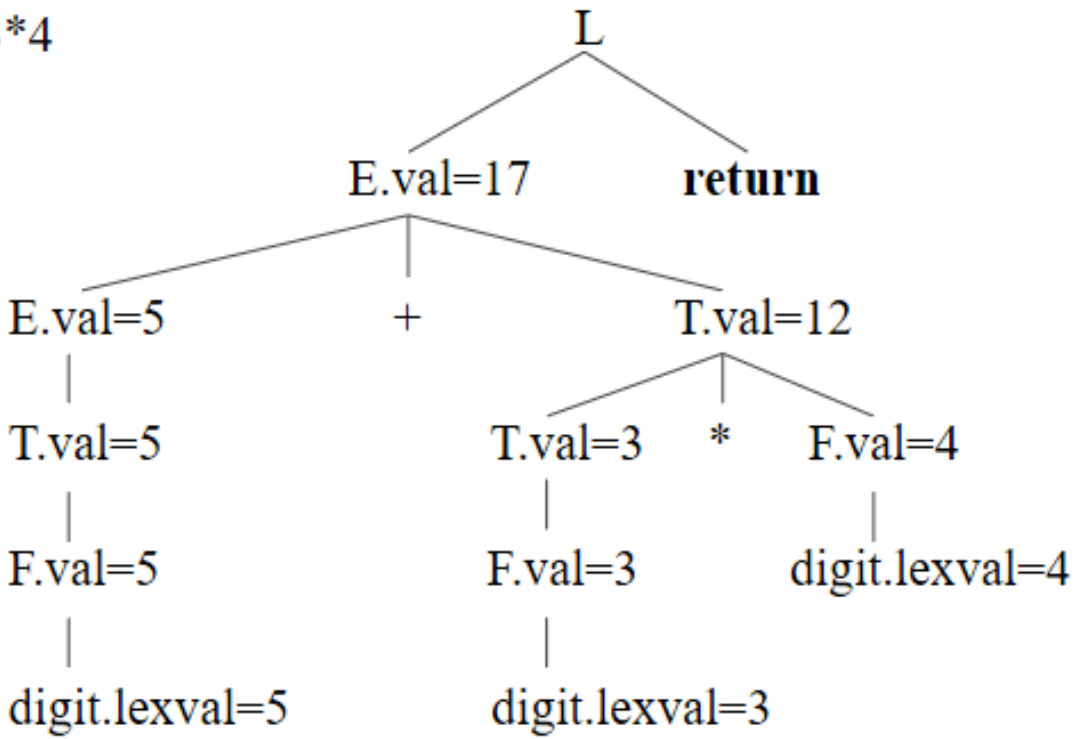


Evaluation of input: **real x,y,z**

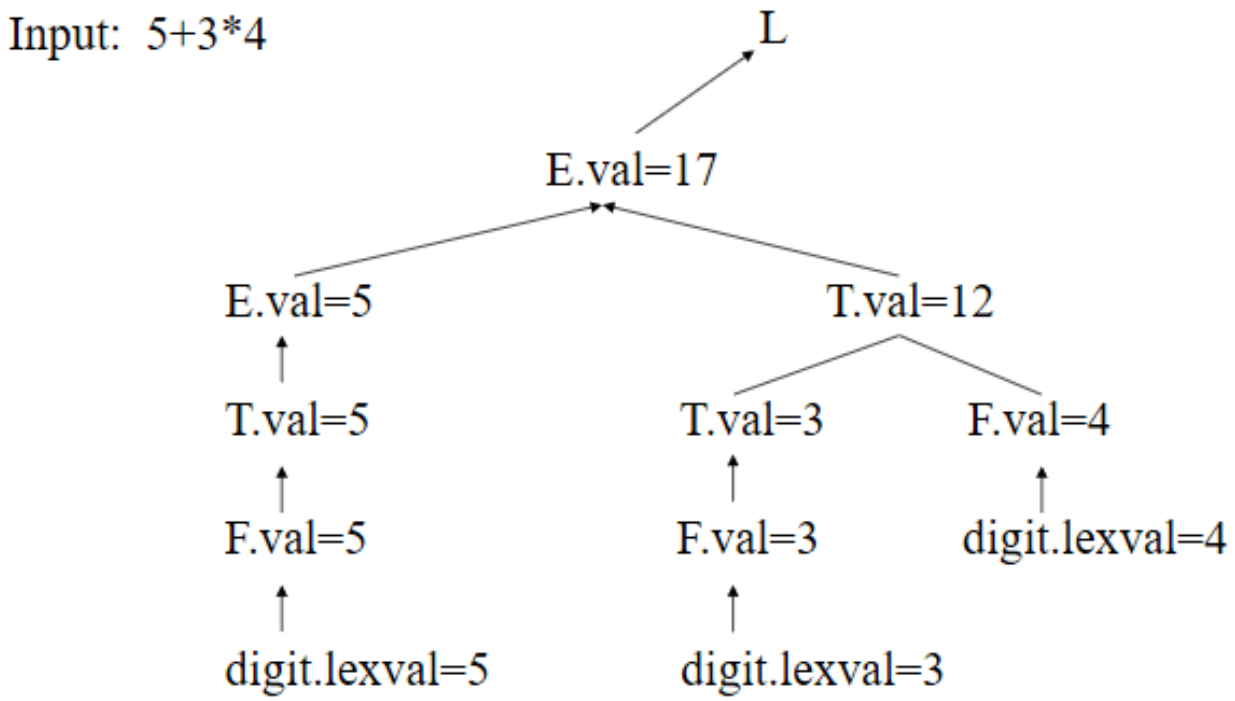


Annotated Parse Tree -- Example

Input: 5+3*4



Dependency Graph



DG: Edges in the dependency graph determine the evaluation order for attribute values

Thanks