Syntax Directed Translation

Syntax-Directed Translation

- Grammar symbols are associated with **attributes** to associate information with the programming language constructs that they represent.
- 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
- An attribute may hold almost any thing.
  - a string, a number, a memory location, a complex record.

Syntax-Directed Translation Cont…

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

Syntax-Directed Definition

- 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 on* attributes $c_1,c_2,\ldots,c_n$.
- 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).

Synthesized Attribute Ex1

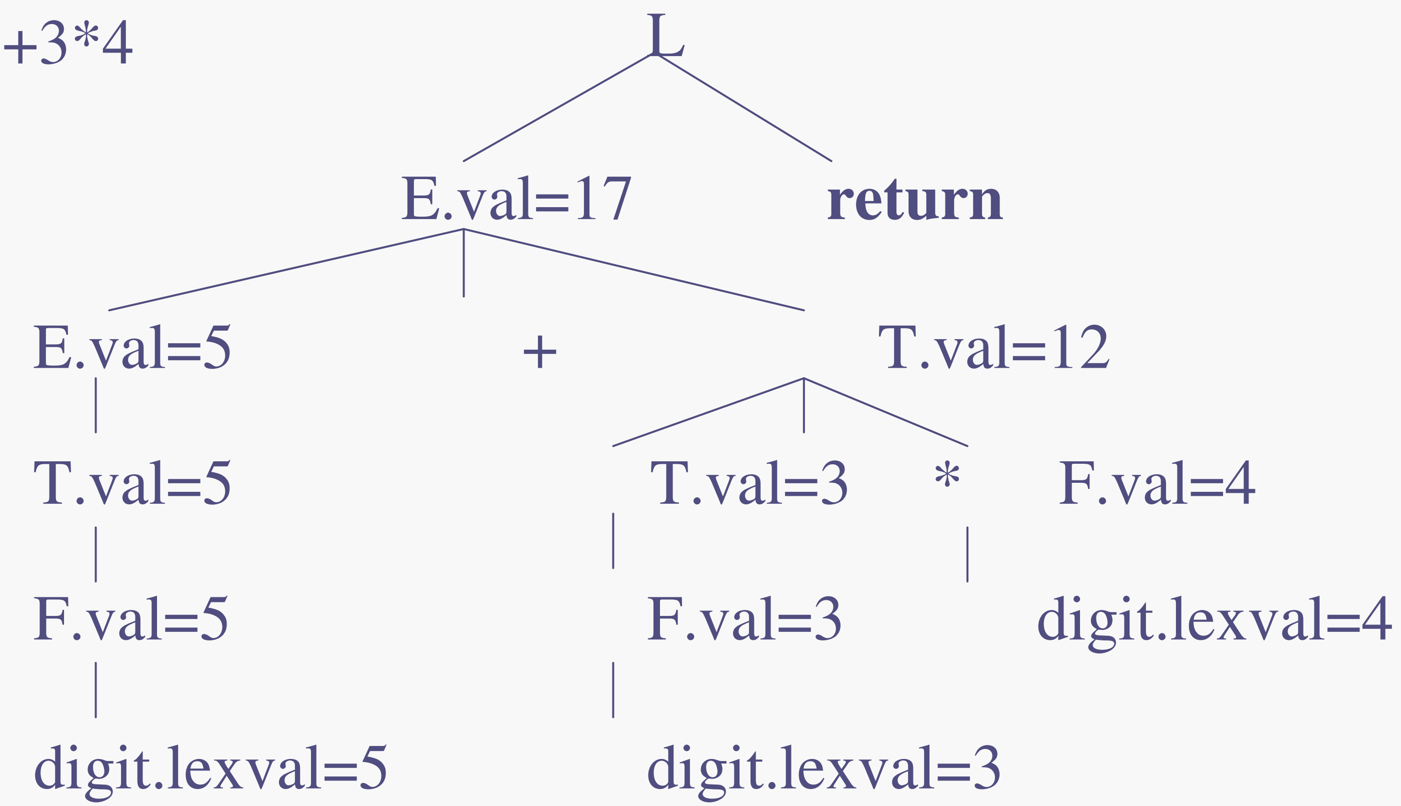
Production	**Semantic Rules**
$L \rightarrow E$ **return**	print(E.val)
$E \rightarrow E_1 + T$	$E.val = E_1.val + T.val$
$E \rightarrow T$	$E.val = T.val$
$T \rightarrow T_1 * F$	$T.val = T_1.val * F.val$
$T \rightarrow F$	$T.val = F.val$
$F \rightarrow (E)$	$F.val = E.val$
$F \rightarrow$ **digit**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

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).

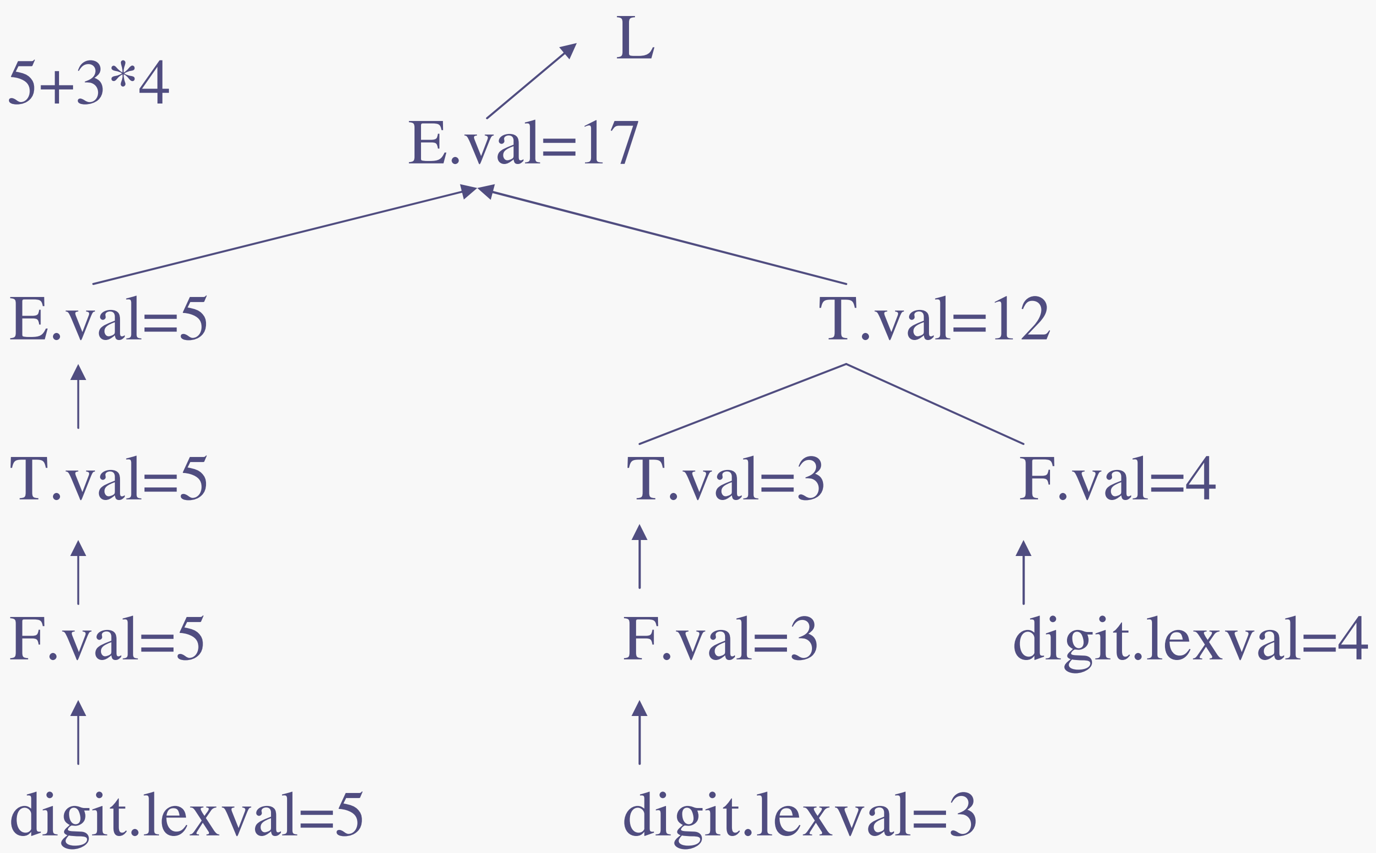
Annotated Parse Tree Example

Input: 5+3*4



Dependency Graph

Input: 5+3*4



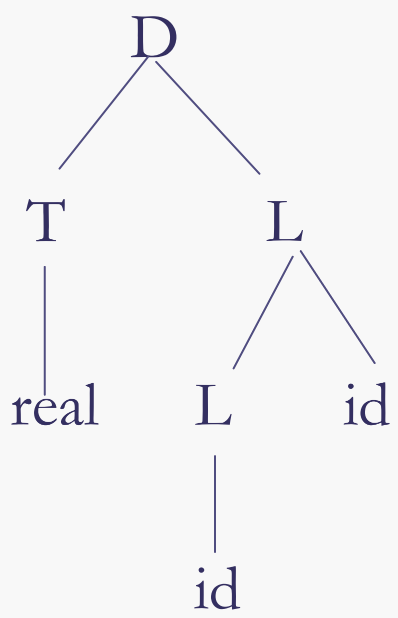
Inherited Attribute

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)

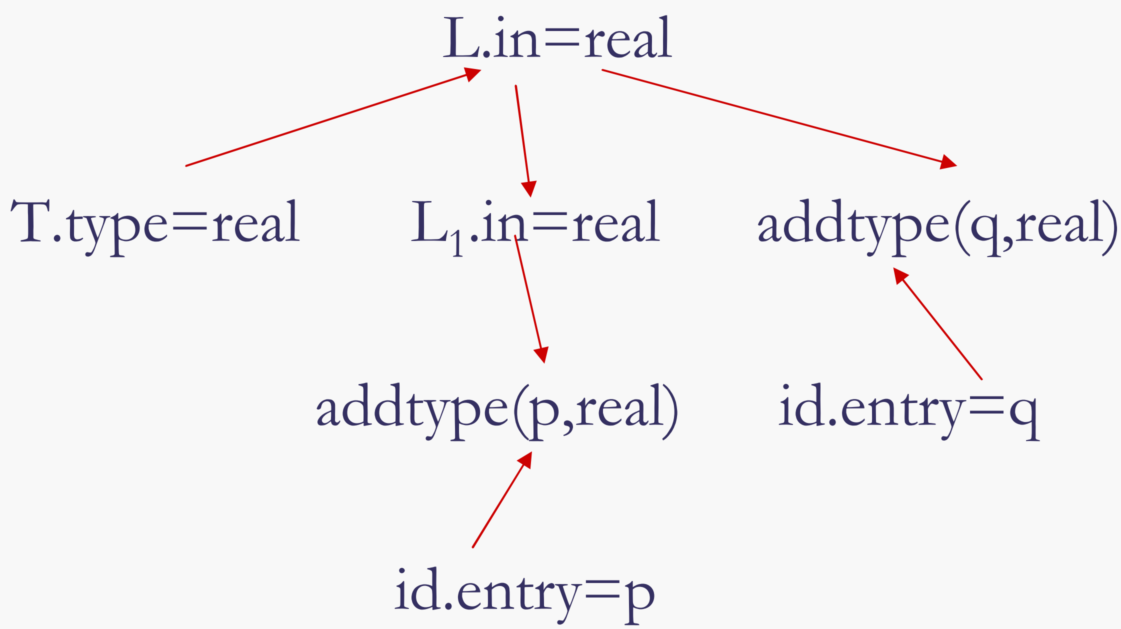
- Symbol T is associated with a synthesized attribute *type*.
- Symbol L is associated with an inherited attribute *in*.

Inherited Attribute Cont…

Input: `real p q`



parse tree



dependency graph

Translation Schemes

- In a syntax-directed definition, we do not say anything about the evaluation times of the semantic rules (when the semantic rules associated with a production should be evaluated?).

- A **translation scheme** is a context-free grammar in which:
  – attributes are associated with the grammar symbols and
  – semantic actions enclosed between braces {} are inserted within the right sides of productions.

- *Ex:* A → { ... } X { ... } Y { ... }

Semantic Actions

Translation Schemes

- When designing a translation scheme, some restrictions should be observed to ensure that an attribute value is available when a semantic action refers to that attribute.
- These restrictions (motivated by L-attributed definitions) ensure that a semantic action does not refer to an attribute that has not yet computed.
- In translation schemes, we use *semantic action* terminology instead of *semantic rule* terminology used in syntax-directed definitions.
- The position of the semantic action on the right side indicates when that semantic action will be evaluated.

Translation Schemes

Production	Semantic Rule
$E \to E_1 + T$	$E.val = E_1.val + T.val$

➔ a production of a syntax directed definition

$$\Downarrow$$

$E \to E_1 + T \; \{ \; E.val = E_1.val + T.val \; \}$

➔ the production of the corresponding translation scheme

Translation Scheme Example

- A simple translation scheme that converts infix expressions to the corresponding postfix expressions.

$E \rightarrow T\ R$

$R \rightarrow +\ T\ \{\ \text{print}(\text{"+"})\ \}\ R_1$

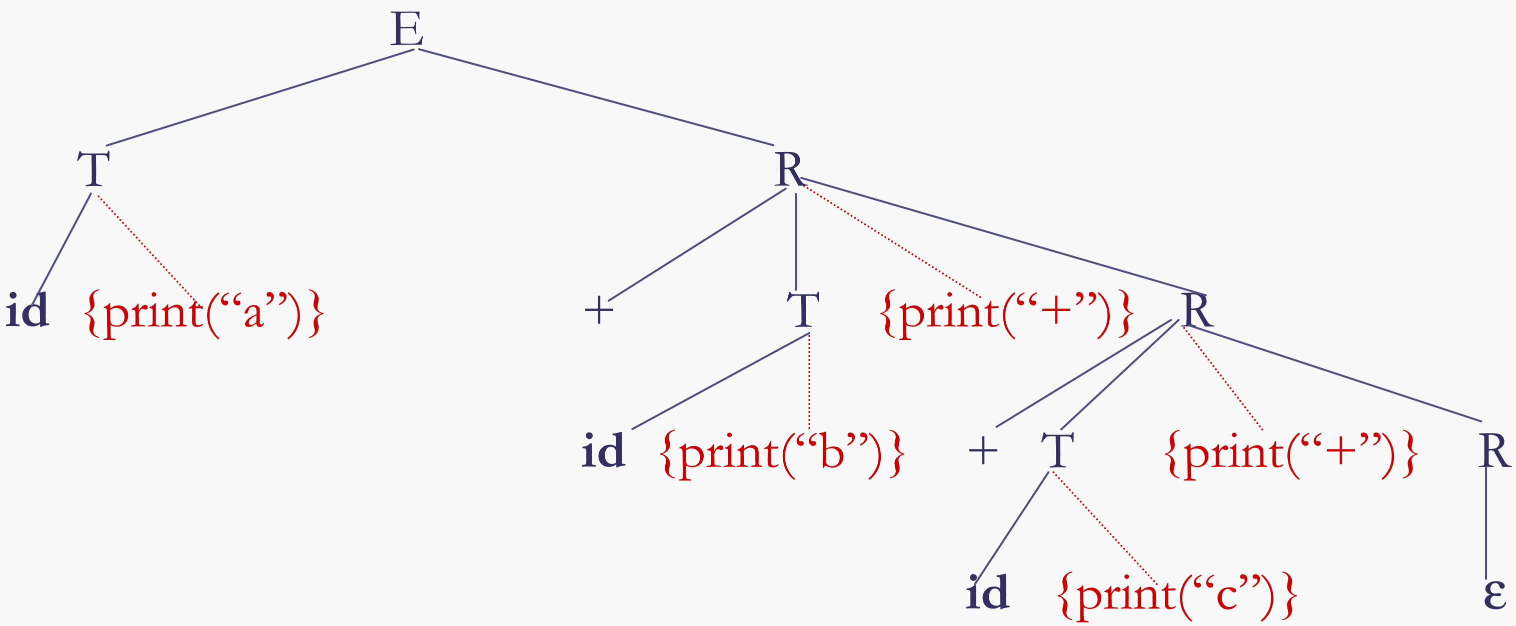
$R \rightarrow \varepsilon$

$T \rightarrow \textbf{id}\ \{\ \text{print}(\textbf{id}.\text{name})\ \}$

a+b+c ➔ ab+c+

infix expression postfix expression

Translation Scheme Example



The depth first traversal of the parse tree (executing the semantic actions in that order)

will produce the postfix representation of the infix expression.