Unit-4

Semantic Analysis:
Syntax Directed Translation

Mrs.Soma Ghosh
gsn.comp@coeptech.ac.in

- **Syntax Directed Translations**
- Syntax Directed Definitions
- Implementing Syntax Directed Definitions
  - Dependency Graphs
  - S-Attributed Definitions
  - L-Attributed Definitions
- Translation Schemes

Semantic Analysis

- **Semantic Analysis** computes additional information related to the meaning of the program once the syntactic structure is known.
- In typed languages as C, semantic analysis involves adding information to the symbol table and performing type checking.
- The information to be computed is beyond the capabilities of standard parsing techniques, therefore it is not regarded as syntax.
- As for Lexical and Syntax analysis, also for Semantic Analysis we need both a *Representation Formalism* and an *Implementation Mechanism*.

Syntax Directed Translation: Intro

- The **Principle of Syntax Directed Translation** states that the meaning of an input sentence is related to its syntactic structure, i.e., to its Parse-Tree.
- By **Syntax Directed Translations** we indicate those formalisms for specifying translations for programming language constructs guided by context-free grammars.
  - We associate **Attributes** to the grammar symbols representing the language constructs.
  - Values for attributes are computed by **Semantic Rules** associated with grammar productions.

Syntax Directed Translation: Intro (Cont.)

- Evaluation of Semantic Rules may:
  - Generate Code;
  - Insert information into the Symbol Table;
  - Perform Semantic Check;
  - Issue error messages;
  - etc.
- There are two notations for attaching semantic rules:
  1. **Syntax Directed Definitions.** High-level specification hiding many implementation details (also called **Attribute Grammars**).
  2. **Translation Schemes.** More implementation oriented: Indicate the order in which semantic rules are to be evaluated.

- Syntax Directed Translations
- **Syntax Directed Definitions**
- Implementing Syntax Directed Definitions
  - Dependency Graphs
  - S-Attributed Definitions
  - L-Attributed Definitions
- Translation Schemes

Syntax Directed Definitions

- **Syntax Directed Definitions** are a generalization of context-free grammars in which:
  1. Grammar symbols have an associated set of **Attributes**;
  2. Productions are associated with **Semantic Rules** for computing the values of attributes.
- Such formalism generates **Annotated Parse-Trees** where each node of the tree is a record with a field for each attribute (e.g., $X.a$ indicates the attribute a of the grammar symbol X).

Syntax Directed Definitions (Cont.)

- The value of an attribute of a grammar symbol at a given parse-tree node is defined by a semantic rule associated with the production used at that node.
- We distinguish between two kinds of attributes:

1. **Synthesized Attributes.** They are computed from the values of the attributes of the children nodes.
2. **Inherited Attributes.** They are computed from the values of the attributes of both the siblings and the parent nodes.

Form of Syntax Directed Definitions

- Each production, $A \rightarrow \alpha$, is associated with a set of semantic rules: $b := f(c_1, c_2, \ldots, c_k)$, where f is a function and either
  1. b is a **synthesized** attribute of A, and $c_1, c_2, \ldots, c_k$ are attributes of the grammar symbols of the production, or
  2. b is an **inherited** attribute of a grammar symbol in α, and $c_1, c_2, \ldots, c_k$ are attributes of grammar symbols in α or attributes of A.
- **Note.** Terminal symbols are assumed to have synthesized attributes supplied by the lexical analyzer.
- Procedure calls (e.g. *print* in the next slide) define values of *Dummy* synthesized attributes of the non terminal on the left-hand side of the production.

Syntax Directed Definitions: An Example

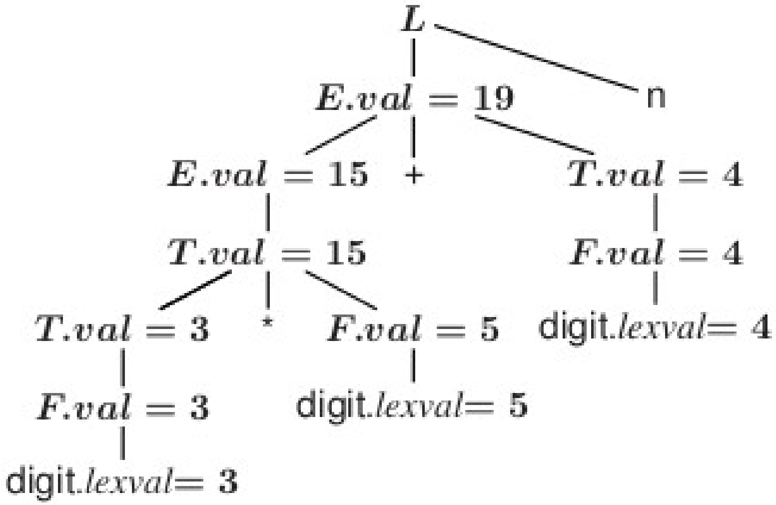
- **Example.** Let us consider the Grammar for arithmetic expressions. The Syntax Directed Definition associates to each non terminal a synthesized attribute called *val*.

PRODUCTION	SEMANTIC RULE
$L \to E\mathsf{n}$	$print(E.val)$
$E \to E_1 + T$	$E.val := E_1.val + T.val$
$E \to T$	$E.val := T.val$
$T \to T_1 * F$	$T.val := T_1.val * F.val$
$T \to F$	$T.val := F.val$
$F \to (E)$	$F.val := E.val$
$F \to \mathsf{digit}$	$F.val := \mathsf{digit}.lexval$

S-Attributed Definitions

Definition. An **S-Attributed Definition** is a Syntax Directed Definition that uses only synthesized attributes.

- **Evaluation Order.** Semantic rules in a S-Attributed Definition can be evaluated by a bottom-up, or PostOrder, traversal of the parse-tree.
- **Example.** The above arithmetic grammar is an example of an S-Attributed Definition. The annotated parse-tree for the input 3*5+4n is:



Inherited Attributes

- **Inherited Attributes** are useful for expressing the dependence of a construct on the context in which it appears.
- It is always possible to rewrite a syntax directed definition to use only synthesized attributes, but it is often more natural to use both synthesized and inherited attributes.
- **Evaluation Order.** Inherited attributes cannot be evaluated by a simple PreOrder traversal of the parse-tree:
  - Unlike synthesized attributes, the order in which the inherited attributes of the children are computed is important!!! Indeed:
    - Inherited attributes of the children can depend from both left and right siblings!

Inherited Attributes: An Example

- **Example.** Let us consider the syntax directed definition with both inherited and synthesized attributes for the grammar for "type declarations":

PRODUCTION	SEMANTIC RULE
$D \to TL$	$L.in := T.type$
$T \to \mathsf{int}$	$T.type := integer$
$T \to \mathsf{real}$	$T.type := real$
$L \to L_1, \mathsf{id}$	$L_1.in := L.in$; *addtype*(id.*entry*, *L.in*)
$L \to \mathsf{id}$	*addtype*(id.*entry*, *L.in*)

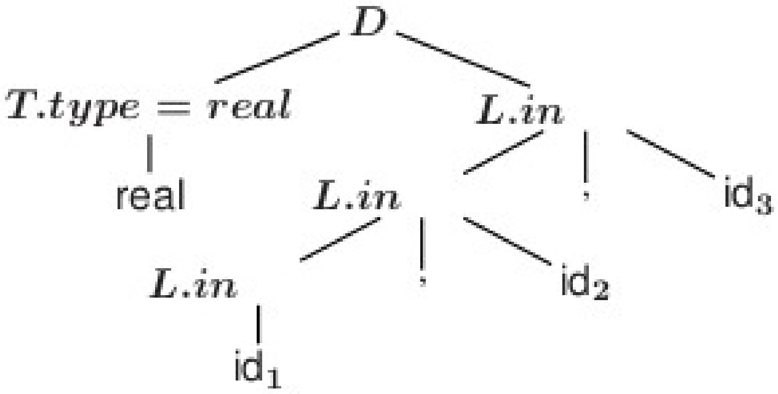
- The non terminal T has a synthesized attribute, *type*, determined by the keyword in the declaration.

- The production $D \to TL$ is associated with the semantic rule $L.in := T.type$ which set the *inherited* attribute $L.in$.

- Note: The production $L \to L_1, \mathsf{id}$ distinguishes the two occurrences of L.

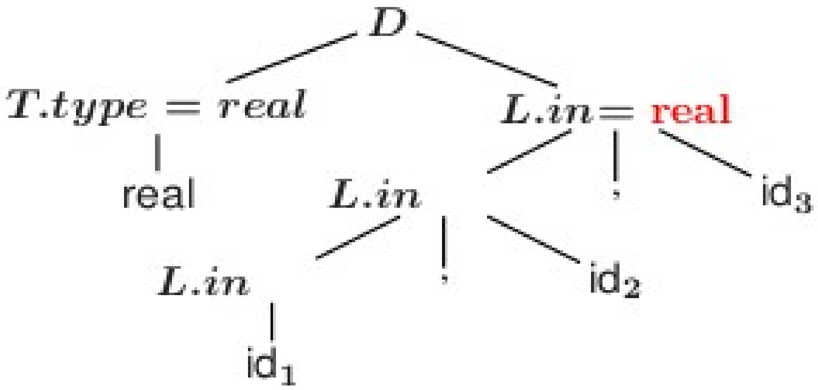
Inherited Attributes: An Example (Cont.)

- Synthesized attributes can be evaluated by a PostOrder traversal.
- Inherited attributes that *do not depend from right children* can be evaluated by a classical PreOrder traversal.
- The annotated parse-tree for the input real id_1, id_2, id_3 is:



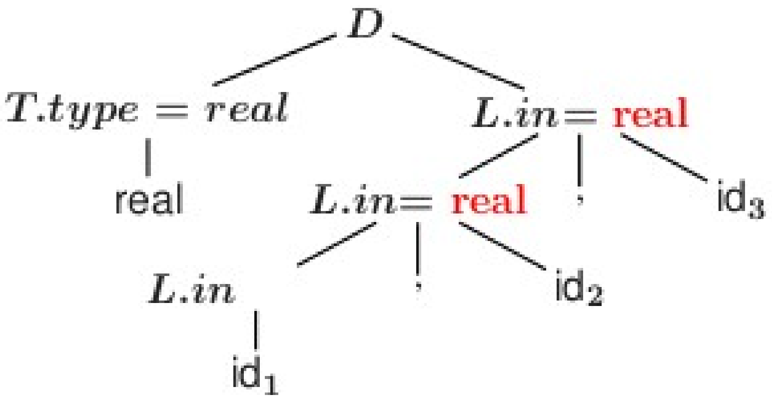
Inherited Attributes: An Example (Cont.)

- Synthesized attributes can be evaluated by a PostOrder traversal.
- Inherited attributes that *do not depend from right children* can be evaluated by a classical PreOrder traversal.
- The annotated parse-tree for the input real id_1, id_2, id_3 is:



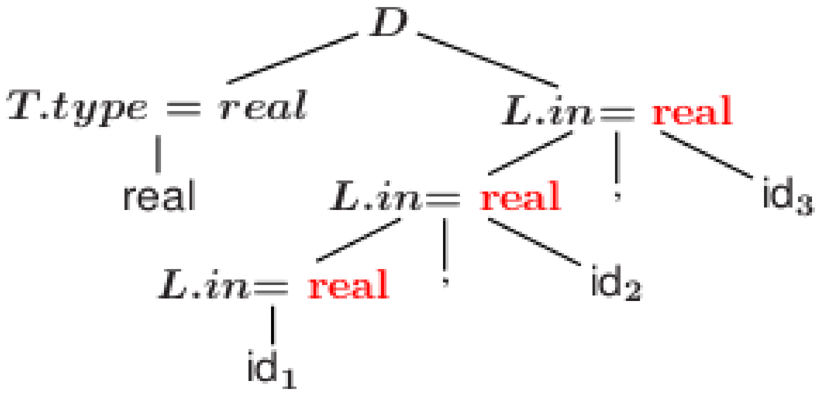
Inherited Attributes: An Example (Cont.)

- Synthesized attributes can be evaluated by a PostOrder traversal.
- Inherited attributes that *do not depend from right children* can be evaluated by a classical PreOrder traversal.
- The annotated parse-tree for the input real id_1, id_2, id_3 is:



Inherited Attributes: An Example (Cont.)

- Synthesized attributes can be evaluated by a PostOrder traversal.
- Inherited attributes that *do not depend from right children* can be evaluated by a classical PreOrder traversal.
- The annotated parse-tree for the input real id_1, id_2, id_3 is:



- $L.in$ is then inherited top-down the tree by the other L-nodes.
- At each L-node the procedure *addtype* inserts into the symbol table the type of the identifier.

- Syntax Directed Translations
- Syntax Directed Definitions
- **Implementing Syntax Directed Definitions**
  - **Dependency Graphs**
  - S-Attributed Definitions
  - L-Attributed Definitions
- Translation Schemes

Dependency Graphs

- Implementing a Syntax Directed Definition consists primarily in finding an order for the evaluation of attributes
  - Each attribute value must be available when a computation is performed.
- **Dependency Graphs** are the most general technique used to evaluate syntax directed definitions with both synthesized and inherited attributes.
- A Dependency Graph shows the interdependencies among the attributes of the various nodes of a parse-tree.
  - There is a node for each attribute;
  - If attribute b depends on an attribute c there is a link from the node for c to the node for b ($b \leftarrow c$).
- **Dependency Rule:** If an attribute b depends from an attribute c, then we need to fire the semantic rule for c first and then the semantic rule for b.

Evaluation Order

- The evaluation order of semantic rules depends from a *Topological Sort* derived from the dependency graph.
- **Topological Sort:** Any ordering $m_1, m_2, \ldots, m_k$ such that if $m_i \rightarrow m_j$ is a link in the dependency graph then $m_i < m_j$.
- Any topological sort of a dependency graph gives a valid order to evaluate the semantic rules.

Evaluation of S-Attributed Definitions

- **Example.** Consider the S-attributed definitions for the arithmetic expressions. To evaluate attributes the parser executes the following code

PRODUCTION	CODE
$L \to E\mathsf{n}$	$print(val[top-1])$
$E \to E_1 + T$	$val[ntop] := val[top] + val[top-2]$
$E \to T$	
$T \to T_1 * F$	$val[ntop] := val[top] * val[top-2]$
$T \to F$	
$F \to (E)$	$val[ntop] := val[top-1]$
$F \to \mathsf{digit}$	

- The variable $ntop$ is set to the *new top of the stack*. After a reduction is done top is set to $ntop$: When a reduction $A \to \alpha$ is done with $|\alpha| = r$, then $ntop = top - r + 1$.

- During a shift action both the token and its value are pushed into the stack.

L-Attributed Definitions

- **L-Attributed Definitions** contain both synthesized and inherited attributes but do not need to build a dependency graph to evaluate them.

- **Definition.** A syntax directed definition is *L-Attributed* if each *inherited attribute* of X_j in a production $A \to X_1 \ldots X_j \ldots X_n$, depends only on:

  1. The attributes of the symbols to the **left** (this is what L in *L-Attributed* stands for) of X_j, i.e., $X_1 X_2 \ldots X_{j-1}$, and
  2. The *inherited* attributes of A.

- **Theorem.** Inherited attributes in L-Attributed Definitions can be computed by a PreOrder traversal of the parse-tree.

- Syntax Directed Translations
- Syntax Directed Definitions
- Implementing Syntax Directed Definitions
  - Dependency Graphs
  - S-Attributed Definitions
  - L-Attributed Definitions
- **Translation Schemes**

Translation Schemes

- **Translation Schemes** are more implementation oriented than syntax directed definitions since they **indicate the order** in which semantic rules and attributes are to be evaluated.
- **Definition.** A Translation Scheme is a context-free grammar in which
  1. Attributes are associated with grammar symbols;
  2. Semantic Actions are enclosed between braces {} and **are inserted within the right-hand side of productions.**
- Yacc uses Translation Schemes.

Translation Schemes (Cont.)

- Translation Schemes deal with both synthesized and inherited attributes.
- Semantic Actions are treated as terminal symbols: Annotated parse-trees contain semantic actions as children of the node standing for the corresponding production.
- Translation Schemes are useful to evaluate L-Attributed definitions at parsing time (even if they are a general mechanism).
  - **An L-Attributed Syntax-Directed Definition can be turned into a Translation Scheme.**

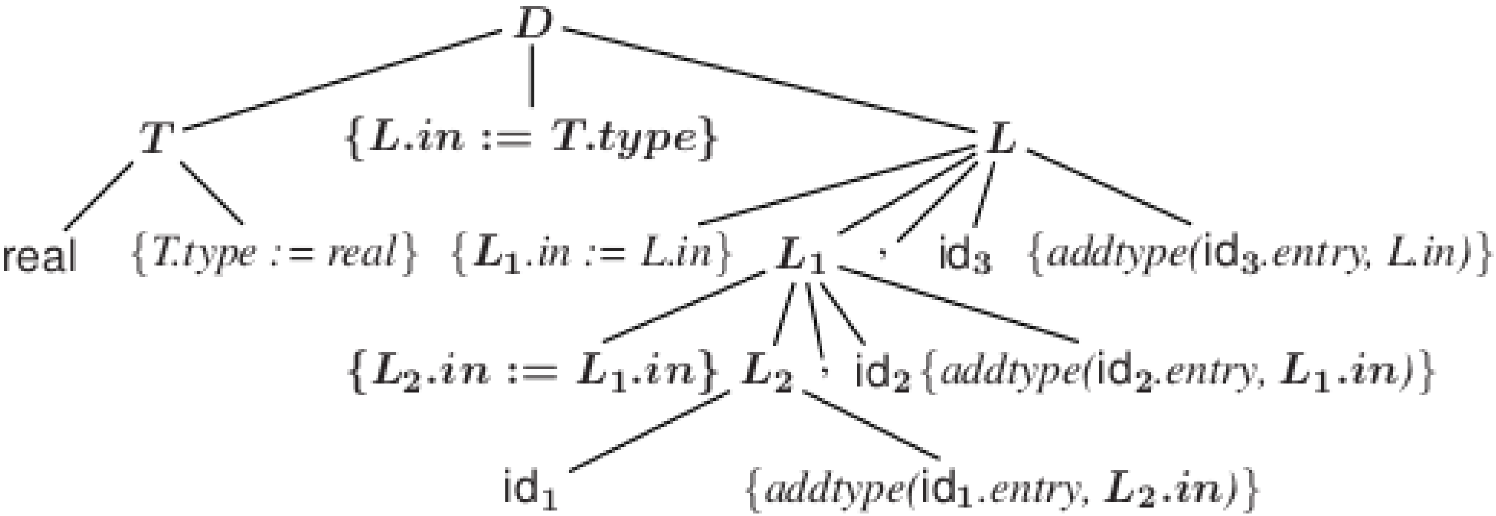
Translation Schemes: An Example

- Consider the Translation Scheme for the L-Attributed Definition for "type declarations":

$$
\begin{aligned}
D &\rightarrow \quad T \; \{L.in := T.type\} \; L \\
T &\rightarrow \quad \mathsf{int} \; \{T.type := integer\} \\
T &\rightarrow \quad \mathsf{real} \; \{T.type := real\} \\
L &\rightarrow \quad \{L_1.in := L.in\} \; L_1, \mathsf{id} \; \{addtype(\mathsf{id}.entry, L.in)\} \\
L &\rightarrow \quad \mathsf{id} \; \{addtype(\mathsf{id}.entry, L.in)\}
\end{aligned}
$$

Translation Schemes: An Example (Cont.)

- **Example (Cont).** The parse-tree with semantic actions for the input real id_1, id_2, id_3 is:



- **Traversing the Parse-Tree in depth-first order (PostOrder) we can evaluate the attributes.**