Unit-2.2 Semantic Analysis
Syntax Directed Translation

Prashant Gautam

2021

Introductions

- 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

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

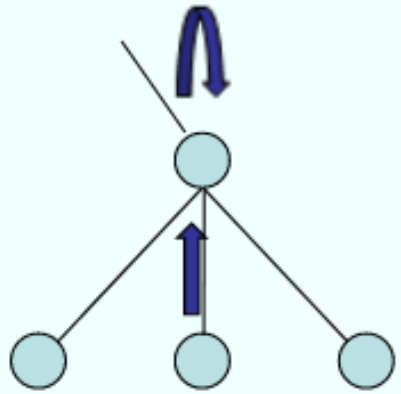
- Syntax-directed definition (Attribute Grammar)
  - Productions with semantic rules
    - Ex: E→E1+T E.code=E1.code ||T.code||'+'
  - More readable
  - Useful for specification
- Syntax-directed translation (Translation Scheme)
  - Productions with semantic actions
    - Ex: E→E1+T { print '+' }
  - More efficient
  - Useful for implementation

Syntax-Directed Definitions

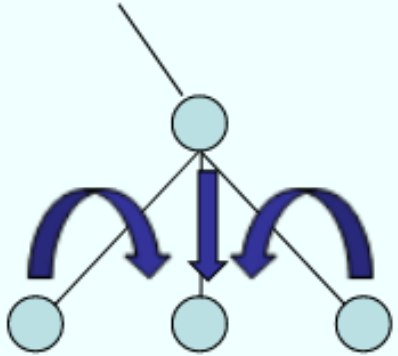
- A syntax-directed definition is a generalization of a context-free grammar in which each grammar symbol is associated with a set of attributes. This set of attributes for a grammar symbol is partitioned into two subsets **synthesized** and **inherited** attributes of that grammar.
- *Semantic rules* set up dependencies between attributes which can be represented by a dependency graph.
- 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.

Synthesized and Inherited Attributes

- For a grammar symbol X and its attribute a, use notation "X.a"
- Each attribute X.x at a parse tree node for symbol X may be
  - **synthesized** , meaning that its value is obtained
    - from attributes of child nodes in the parse tree,
    - or from the lexical analyzer,
    - or from other attributes of the same node in the parse tree



  - **inherited** , meaning that its value is obtained
    - from the parent node in the parse tree,
    - or from sibling nodes in the parse tree
    - v. useful when there is mismatch between grammar for parsing, and "abstract syntax" for translation



EXAMPLE

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

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

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).

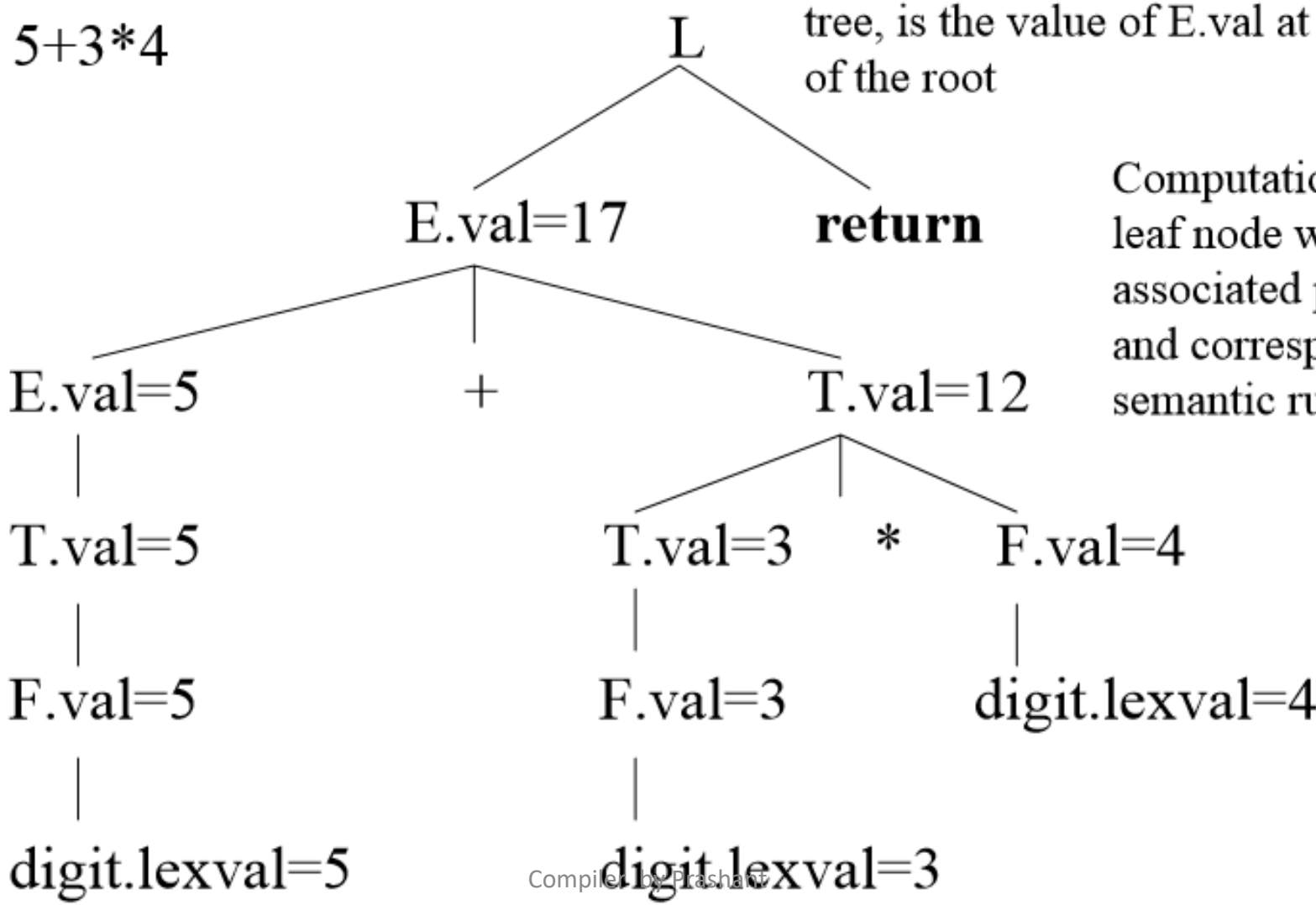
Annotated Parse Tree

Example

Input: 5+3*4

Output: The value, printed at the root of tree, is the value of E.val at the first child of the root

Computation start from leaf node with associated production and corresponding semantic rule



Inherited Attributes

An inherited attribute at any node is defined based on the attributes at the parent and/or siblings of the node.

Useful for describing context sensitive behavior of grammar symbols.

For example, an inherited attribute can be used to keep track of whether an identifier appears at the left or right side of an assignment operator. This can be used to decide whether the address or the value of the identifier needed.

Inherited Attributes

Example

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)

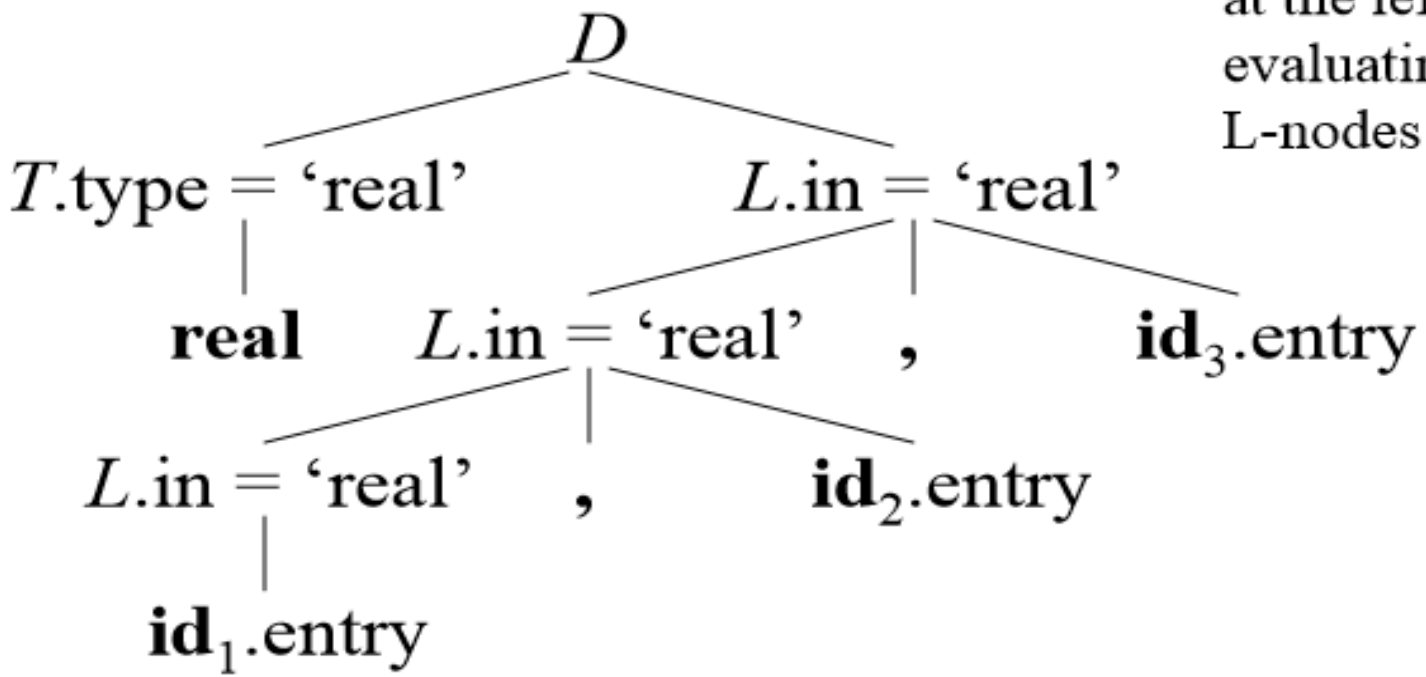
inherited

synthesized

Inherited Attributes

Parse Tree

Input : real id1 , id2 , id3

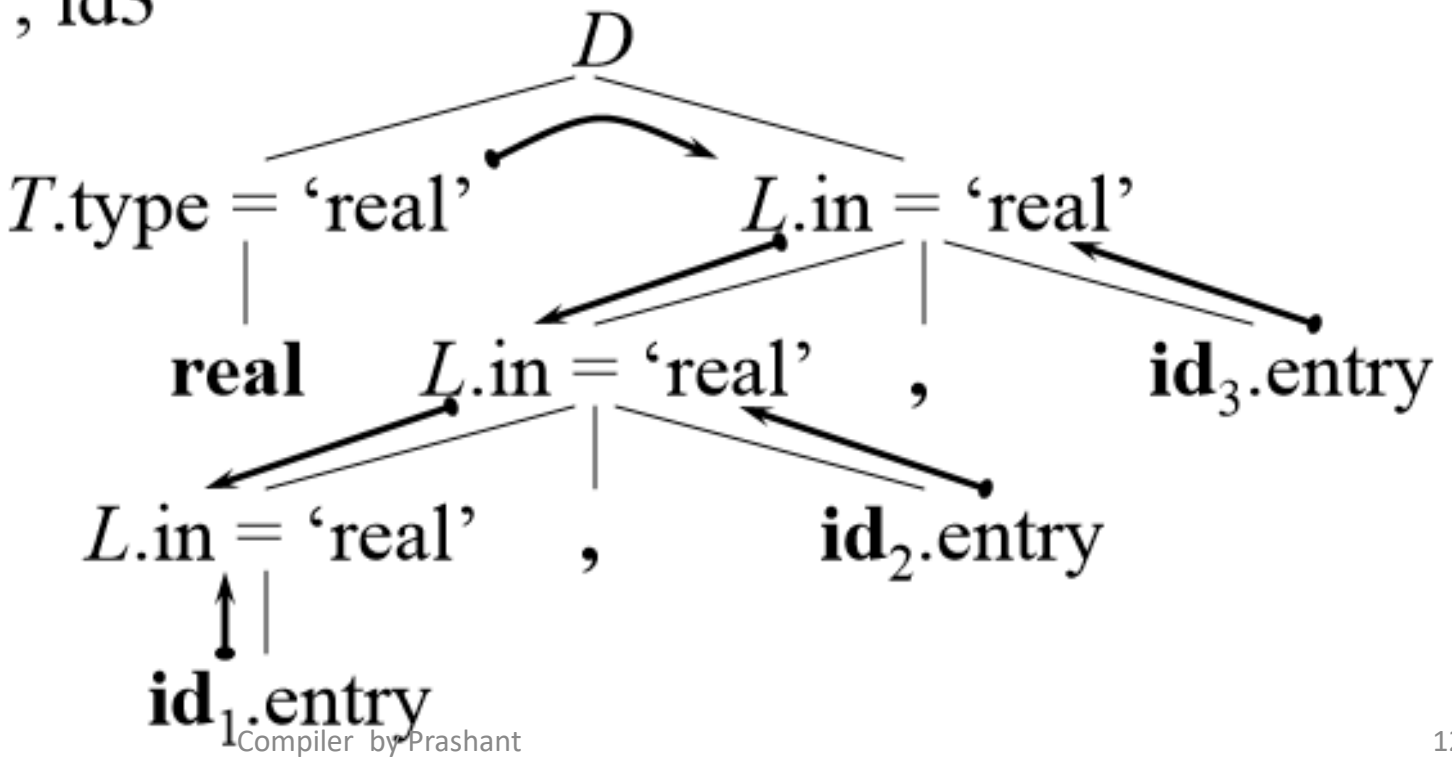


The value of L.in at the three L-nodes gives the type of identifiers id1, id2 & id3. These values are determine by computing the value of attribute T.type at the left child of the root and then evaluating L.in in top-down at the three L-nodes in the right sub tree of the root

Dependency Graph

- In order to correctly evaluate attributes of syntax tree nodes, a dependency graph is useful. A dependency graph is a directed graph that contains attributes as nodes and dependencies across attributes as edge

Input : real id1 , id2 , id3



S-Attributed Definitions

- A syntax-directed definition that uses synthesized attributes exclusively is called an S-attributed definition(or S-attributed grammar)
- A parse tree of an S-attributed definition/grammar is annotated by evaluating the semantic rules for the attribute at each node bottom-up approach.

L-Attributed Definitions

An L-attribute is an inherited attribute which can be evaluated in a left-to-right fashion. L-attributed definitions can be evaluated using a *depth-first* evaluation order. (L –> attributes flow from left to right)

More precisely, a syntax-directed definition is *L-attributed* if each inherited attribute of X_j on the right side of $A \rightarrow X_1 X_2 \ldots X_n$ depends only on

1. the attributes of the symbols $X_1, X_2, \ldots, X_{j-1}$ to the left of X_j in the production and
2. the inherited attributes of A

S-Attributed Vs. L-attributed Definitions

S-attributed Definitions	L-Attributed Definitions
1. Uses only synthesized attributes.	1. Uses both synthesized and inherited attributes. Each inherited attribute is restricted to inherit either from parent or left sibling only. E.g. A→ XYZ {Y.S = A.S ,Y.S = X.S , Z.S = Y.S }
2. Semantic actions are placed at right end of the production.	2. Semantic actions are placed anywhere on RHS. A→ { }BC / D{ }E / FG{ }
3. Attributes are evaluated during bottom – up Parsing.	3. Attributes are evaluated by traversing parse tree dept first, left to right order.

Translation Scheme

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: $A \rightarrow \{ \ldots \}\ X\ \{ \ldots \}\ Y\ \{ \ldots \}$

Semantic Actions

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.

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+

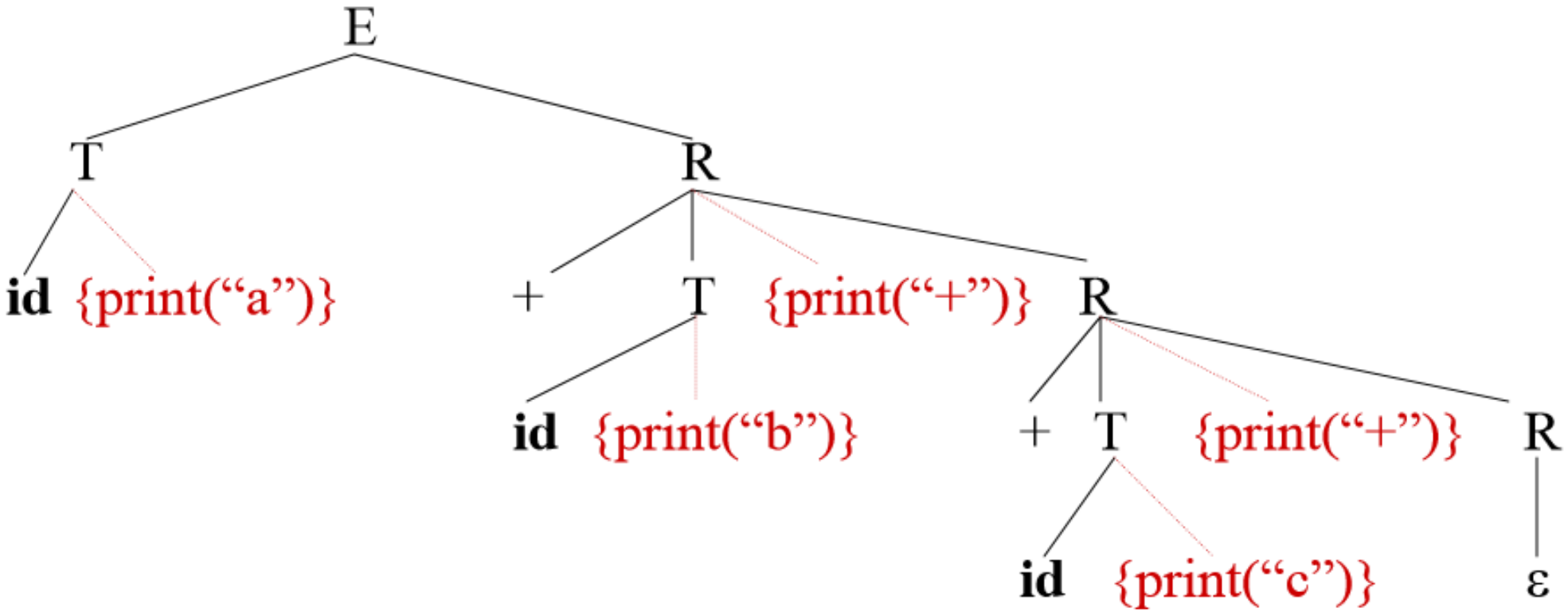
infix expression postfix expression

$E \rightarrow T\ R$

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

$R \rightarrow \varepsilon$

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



The depth first traversal of the parse tree (executing the semantic actions in that order) will produce the postfix representation of the infix expression.

Question

- Write the translation scheme that converts infix expression given below to corresponding postfix expression

2 + 3 * 4

E➔ E+T {print(“+”);}

E→ T {}

T→ T*F {print(“*”);}

T→ F {}

F→ num {print(num.lexval);}