# Compiler Design

**Preet Kanwal**
Department of Computer Science & Engineering

Teaching Assistant : Kavya P K

# Compiler Design

## Unit 3: L-Attributed SDD

**Preet Kanwal**

Department of Computer Science & Engineering

## Lecture Overview

**In this lecture, you will learn about -**

- What is an L-attributed SDD?
- L–Attributed SDD Examples
    - Simple Type declaration
    - Array type Variable Declaration
    - Variable declaration verification
    - Desk calculator

## Recap

- Syntax Directed Definitions (SDD) are a generalization of context-free grammars in which -
  - Grammar symbols have an associated set of attributes
  - Productions are associated with Semantic Rules for computing the values of attributes.
- Two kinds of attributes
  - Synthesized Attributes - computed from the values of the attributes of the children nodes.
  - Inherited Attributes - computed from the values of the attributes of both the siblings and the parent nodes.
- An SDD with only synthesized attributes is called an S-attributed definition.

## What is an L-attributed SDD?

A Syntax Directed Definition is **L-attributed** if all attributes are either -

1. Synthesized
2. Extended synthesized attributes, which can depend not only on attributes at the children, but on inherited attributes at the node itself.
3. Inherited, but depending only on inherited attributes at the parent and any attributes at siblings to the left.

# L-attributed SDD

- The formal definition of an L-Attributed SDD is as follows -

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

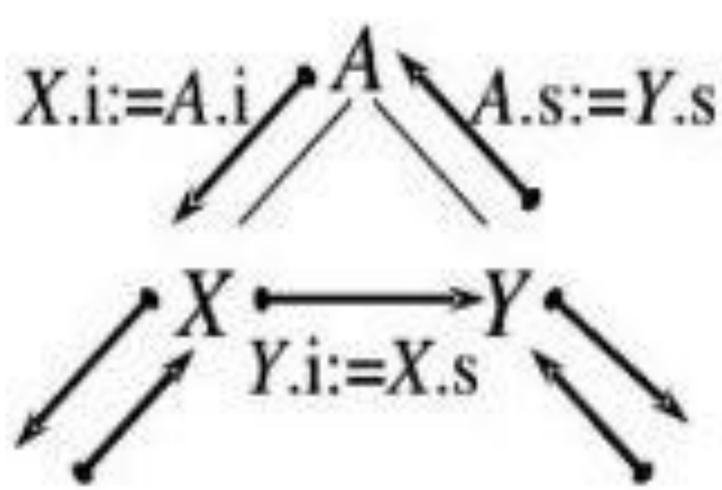
1. The attributes of the symbols to the left (this is what L in L-Attributed stands for) of $X_j$, i.e., $X_1 X_2 \ldots X_{j-1}$
2. The inherited attributes of $A$.

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

## L-attributed SDD

- L-attributed definitions allow for the natural order of evaluating attributes, i.e, depth first, left to right.
- For example -

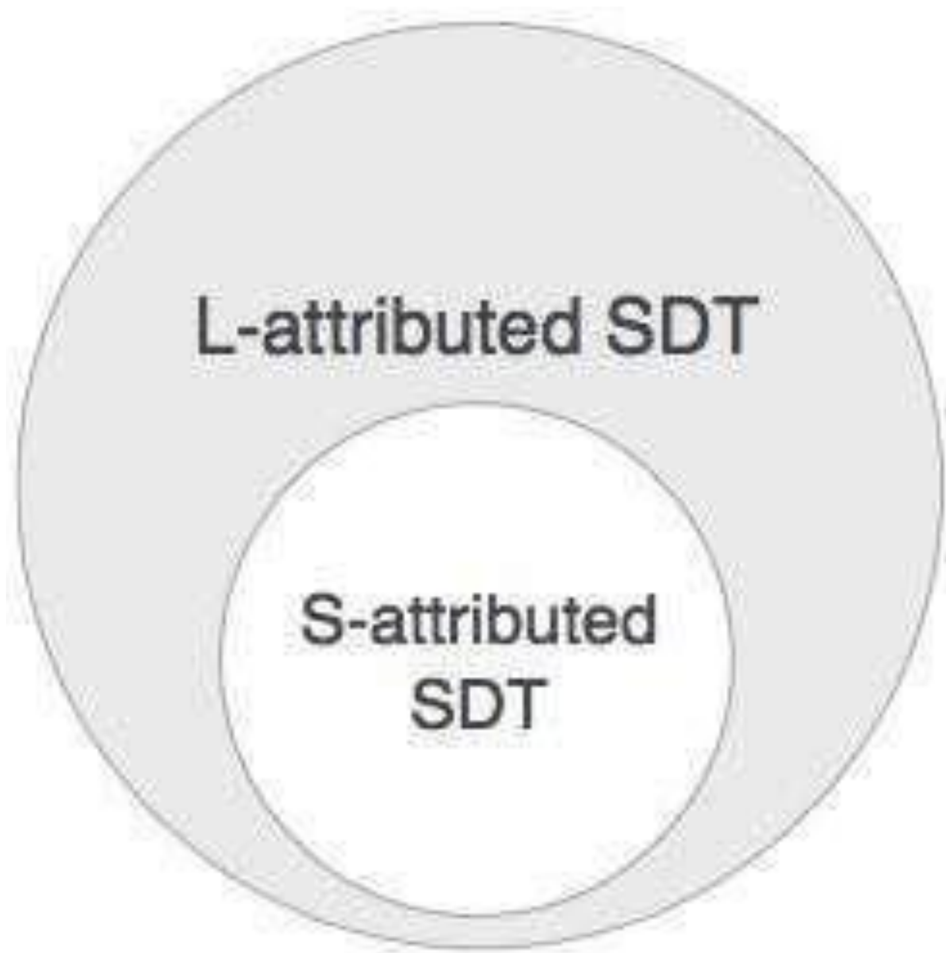
  A → X Y




$$X.i := A.i$$
$$Y.i := X.s$$
$$A.s := Y.s$$

# L-attributed SDD

**Every S-attributed Syntax-Directed Definition is also L-attributed.**



## L-Attributed SDD to implement a Simple Desk Calculator

**Complete the semantic rules for the following L-attributed SDD.**

**Evaluate the SDD for the input 3 + 5**

| Production | Semantic Rule |
|---|---|
| $E \rightarrow T\ E'$ | |
| $E' \rightarrow +\ T\ E'_1$ | |
| $E \rightarrow \lambda$ | |
| $T \rightarrow F\ T'$ | |
| $T' \rightarrow *\ F\ T'_1$ | |
| $T' \rightarrow \lambda$ | |
| $F \rightarrow \text{num}$ | |

## L-Attributed SDD to implement a Simple Desk Calculator

| Production | Semantic Rule |
|---|---|
| $E \rightarrow T\ E'$ | { $E'.inhval = T.val$; $E.val = E'.val$; } |
| $E' \rightarrow +\ T\ E'_1$ | { $E'_1.inhval = E'.inhval + T.val$; $E'.val = E'_1.val$; } |
| $E' \rightarrow \lambda$ | { $E'.val = E'_1.inhval$; } |
| $T \rightarrow F\ T'$ | { $T'.inhval = F.val$; $T.val = T'.val$; } |
| $T' \rightarrow *\ F\ T'_1$ | { $T'_1.inhval = T'.inhval + F.val$; $T'.val = T'_1.val$; } |
| $T' \rightarrow \lambda$ | { $T'.val = T'.inhval$ } |

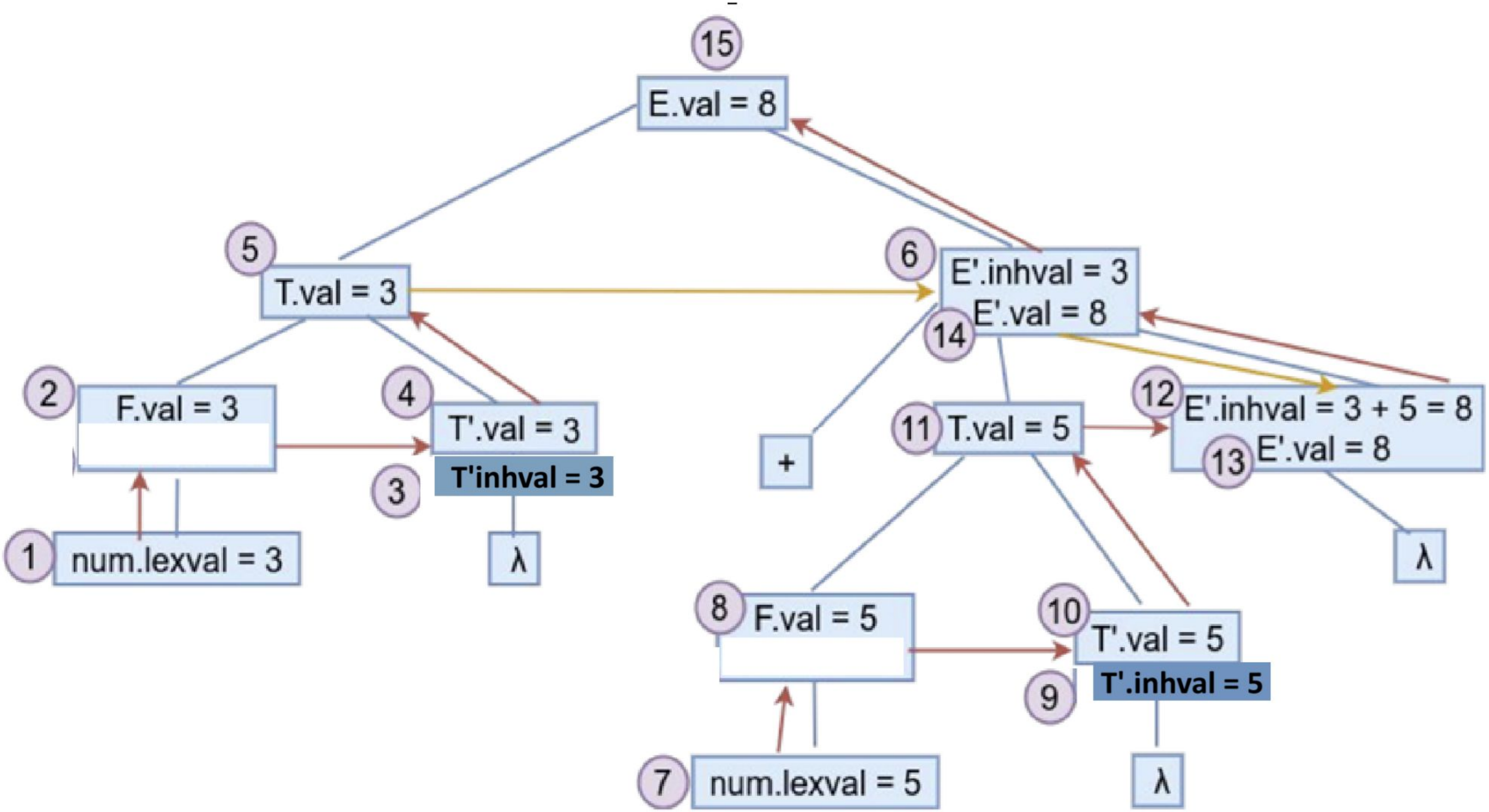
This is an LDD -

In $E \rightarrow T\ E'$,

the attribute for E' is inherited from T, which is the left-sibling.

## L-Attributed SDD to implement a Simple Desk Calculator

### Evaluate the SDD for the input 3 + 5



## L-attributed SDD for Simple Type declaration

Complete the semantic rules for the following L-attributed SDD.

Evaluate the SDD for the input int a,b

| Production | Semantic Rule |
|---|---|
| $D \longrightarrow T\ L$ | |
| $T \longrightarrow$ int | |
| $T \longrightarrow$ float | |
| $L \longrightarrow L_1$ , id | |
| $L \longrightarrow$ id | |

## L-attributed SDD for Simple Type declaration

| Production | Semantic Rule |
|---|---|
| D → T L | *{ L.inhType = T.type; L.inhWidth = T.width; }* |
| T → int | *{ T.type = integer; T.width = 4;}* |
| T → float | *{ T.type = float; T.width = 8; }* |
| L → $L_1$ , id | *{ $L_1$.inhType = L.inhType; $L_1$.inhWidth = L.inhWidth; update(id.entry, L.inhType,L.inhWidth); }* |
| L → id | *{update(id.entry, L.inhType,L.inhWidth); }* |

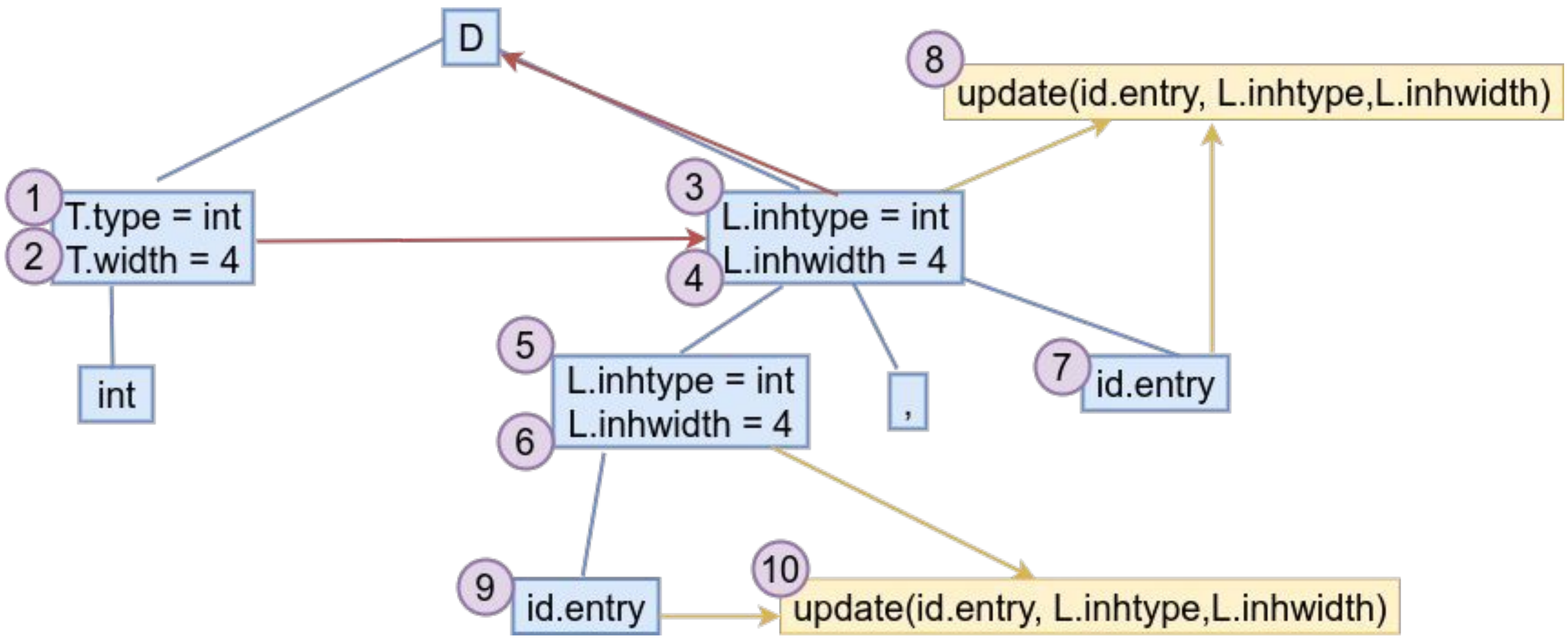
This is an LDD -

In D →T L, the attribute for L is inherited from T, which is the left-sibling.

update() is used to update type and storage in the symbol table.

## L-attributed SDD for Simple Type declaration

**Evaluate the SDD for the input int a,b;**



## L-attributed SDD to identify Array Type

Complete the semantic rules for the following L-attributed SDD.

Evaluate the SDD for the input int [2][3]

| Production | Semantic Rule |
|---|---|
| T → B C | |
| B → int | |
| B → float | |
| C → [num] $C_1$ | |
| C → λ | |

## L-attributed SDD to identify Array Type

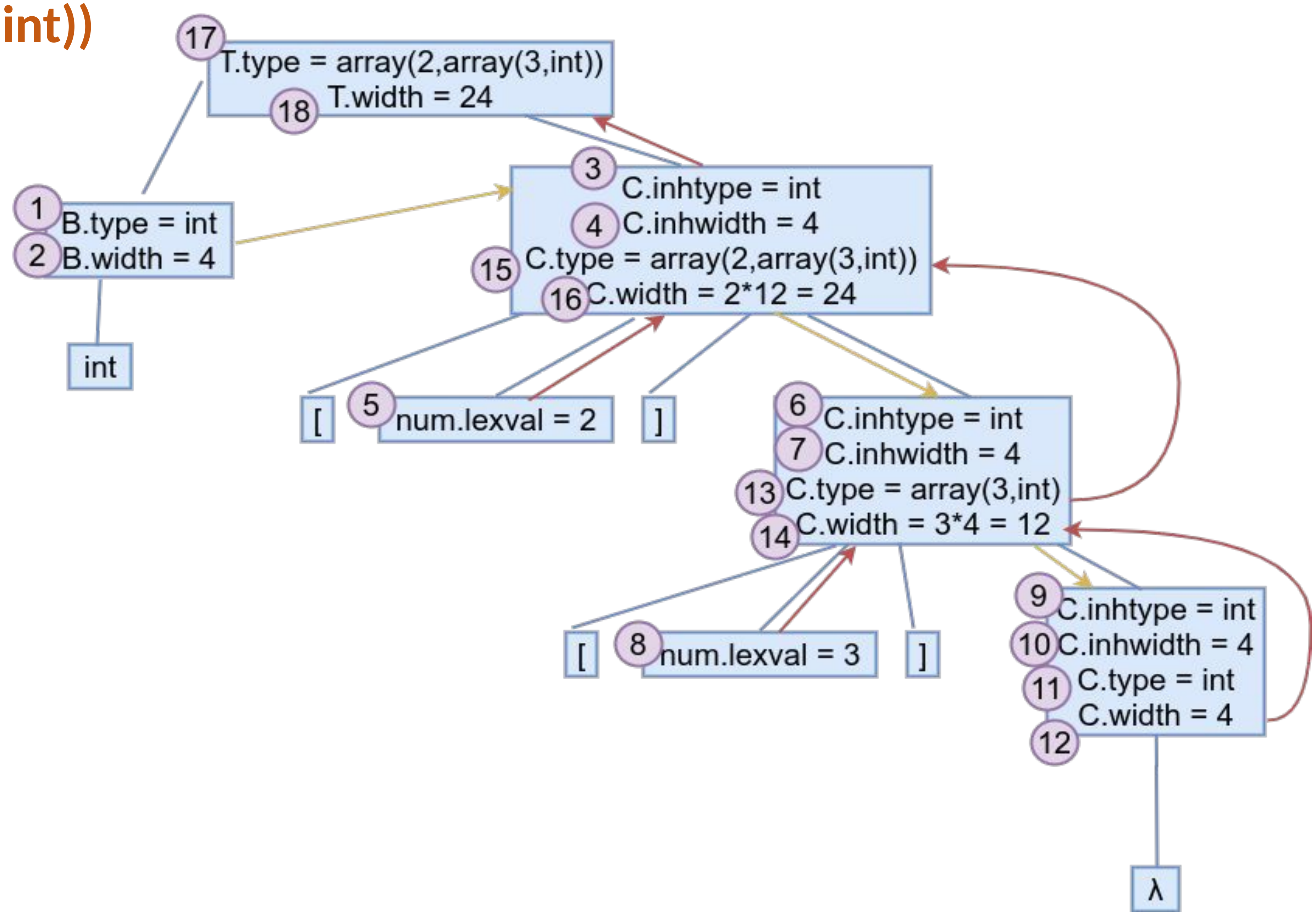
| Production | Semantic Rule |
|---|---|
| $T \longrightarrow B\ C$ | *{ C.inhType = B.type; C.inhWidth = B.width; T.type = C.type; T.width = C.width; }* |
| $B \longrightarrow$ int | *{ B.type = integer;B.width = 4;}* |
| $B \longrightarrow$ float | *{ B.type = float; B.width = 8; }* |
| $C \longrightarrow$ [num] $C_1$ | *{ $C_1$.inhType = C.inhType; $C_1$.inhWidth = C.inhWidth;*<br>*C.type = array(num.lexval , $C_1$.type);*<br>*C.width = num.lexval * $C_1$.width ;);*<br>*addType(id.entry, L.inhType);*<br>*addWidth(id.entry, L.inhWidth; }* |
| $C \longrightarrow \lambda$ | *{ C.type = C.inhType; C.width = C.inhWidth; }* |

## L-attributed SDD for Simple Type declaration

### Evaluate the SDD for the input int a[2][3]

### int a[2][3] translates to

### array(2,array(3,int))



## L-attributed SDD for Basic vs Array Type declaration in C Semantics

Construct an SDD to identify an array of the following format - float[4] x, y

Follow C Semantics - i.e, x is an array type, y is a basic type

| Production | Semantic Rule |
|---|---|
| $D \rightarrow T\ L$ | |
| $T \rightarrow B\ C$ | |
| $B \rightarrow$ int | |
| $B \rightarrow$ float | |
| $C \rightarrow$ [num] $C_1$ | |
| $C \rightarrow \lambda$ | |
| $L \rightarrow L_1$ , id | |
| $L \rightarrow$ id | |

## L-attributed SDD for Basic vs Array Type declaration in C Semantics

| Production | Semantic Rule |
|---|---|
| D→ T L | *{ L.inhType = T.type; L.inhWidth = T.width; L.inhbasicType = T.basicType; L.inhbasicWidth = T.basicWidth; }* |
| T → B C | *{ C.inhType = B.type; C.inhWidth = B.width; T.type = C.type; T.width = C.width; T.basicType = B.type; T.basicWidth = B.width;}* |
| B → int | *{ B.type = integer;B.width = 4;}* |
| B→ float | *{ B.type = float; B.width = 8; }* |
| C → [num] $C_1$ | *{* $C_1$*.inhType = C.inhType;* $C_1$*.inhWidth = C.inhWidth; C.type = array(num.lexval ,* $C_1$*.type); C.width = num.lexval ** $C_1$*.width; }* |
| C → λ | *{* $C_1$*.inhType = C.inhType;* $C_1$*.inhWidth = C.inhWidth; }* |

## L-attributed SDD for Basic vs Array Type declaration in C Semantics

| Production | Semantic Rule |
|---|---|
| … | … |
| $L \rightarrow L_1$ , id | $L_1$.inhType = L.inhType;<br>$L_1$.inhWidth = L.inhWidth;<br>$L_1$.inhbasicType = L.inhbasicType;<br>$L_1$.inhbasicWidth = L.inhbasicWidth;<br>addType(id.entry, L.inhbasicType);<br>addWidth(id.entry, L.inhbasicWidth; |
| $L \rightarrow$ id | { addType(id.entry, L.inhType);<br>addWidth(id.entry, L.inhWidth; } |

# THANK YOU

**Preet Kanwal**

Department of Computer Science & Engineering

**preetkanwal@pes.edu**