

# Syntax Directed Translation

## Lecture 10

### Section 5.1 - 5.3

ROKAN UDDIN FARUQUI

Associate Professor

Dept of Computer Science and Engineering

University of Chittagong, Bangladesh

Email: *rokan@cu.ac.bd*

- 1 Abstract Syntax Trees
- 2 Syntax-Directed Definitions
- 3 Synthesized Attributes
- 4 Inherited Attributes
- 5 Examples
- 6 Assignment

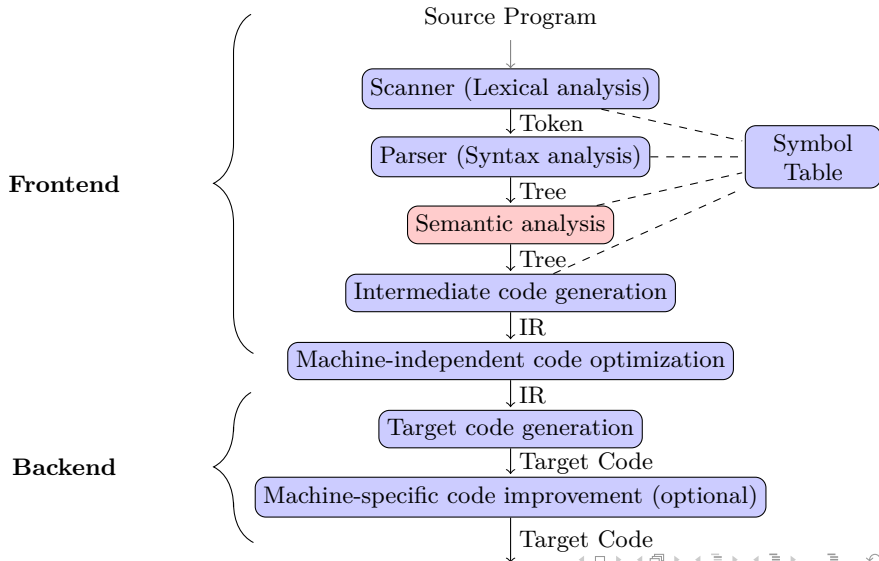


# Outline

- 1 Abstract Syntax Trees
- 2 Syntax-Directed Definitions
- 3 Synthesized Attributes
- 4 Inherited Attributes
- 5 Examples
- 6 Assignment

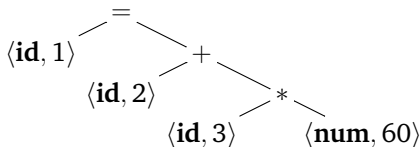


# The Phases of Compilation



# From Parse Tree/ Abstract Syntax Tree (AST)

`position = initial + rate * 60`



id	lexeme
1	position
2	initial
3	rate



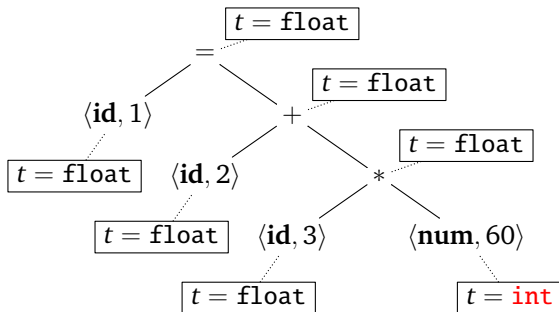
static type checking

# Type Checking, semantic analysis that results in, b annotated tree

```

add x, b
iadd
addf
int a, b;
float c;
b = 2;
c = 4.5;
a = b * c

```



id	lexeme	t
1	position	float
2	initial	float
3	rate	float



# Parse Trees

$$E \Rightarrow E + T$$

- A parse tree shows the *grammatical* structure of a statement.
- It includes all of the grammar symbols (terminals and nonterminals) that were encountered during parsing.



# Abstract Syntax Trees

- An abstract syntax tree (AST) shows the logical structure of the statement.
- Each node represents an action to be taken by the program or an object to be acted upon.
- The syntax tree may introduce operations that were not in the source code or the grammar.
  - ✓ • Dereferencing operations.
  - ✓ • Type-casting operations.
  - ✓ • Jump statements.

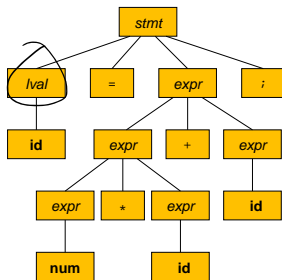




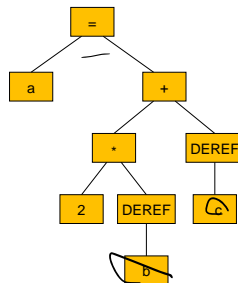
# Syntax Trees vs. Parse Trees

Example (Syntax Trees and Parse Trees)

- Consider the statement  $a = 2*b + c;$



Parse Tree



Syntax Tree



# Syntax Trees vs. Parse Trees

- The parse tree never really exists, except insofar as the parser follows its logical order.
- The AST builder builds the syntax tree from the information obtained by the parser.
- Then the code generator writes the assembly code from the syntax tree.



# Abstract Syntax Trees

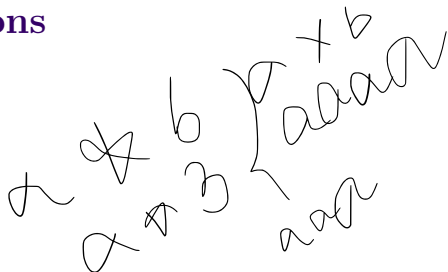
- Recursive descent parsers generally create a single AST for the entire program.
- They build the tree from root to leaf.
- In bottom-up parsing, the AST will be built from the bottom up.



# Outline

- 1 Abstract Syntax Trees
- 2 Syntax-Directed Definitions**
- 3 Synthesized Attributes
- 4 Inherited Attributes
- 5 Examples
- 6 Assignment





A way to systematically associate annotations, e.g. types, to nodes in the grammar/parse tree, calculated by a semantic specification.



# Syntax-Directed Definitions

## Definition

A **syntax-directed definition (SDD)** is a context-free grammar with attributes added to the grammar symbols.

- attributes are stored in the nodes of the syntax tree.
- each production has a set of semantic rules associated for computing the attributes.


If  $X$  is a grammar symbol,  $a$  is an attribute, then  $X.a$  denotes the value of  $a$  at node  $X$



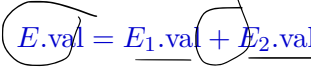
# Example

## Example (Syntax-Directed Definitions)

- Let the grammar be

$$E \rightarrow E + E \mid \text{num}$$


- Then  $E$  derives its value from the **num** tokens in the expression.
- This is expressed formally by the rules

$$\begin{array}{l} E.\text{val} = E_1.\text{val} + E_2.\text{val} \\ \hline E.\text{val} = \text{num.lexval} \end{array}$$


# Syntax-Directed Definitions

- In a syntax-directed definition, each node has
  - A set of synthesized attributes, and
  - A set of inherited attributes.





# Outline

- 1 Abstract Syntax Trees
- 2 Syntax-Directed Definitions
- 3 Synthesized Attributes**
- 4 Inherited Attributes
- 5 Examples
- 6 Assignment



# Synthesized Attributes

## Definition

A **synthesized attribute** of a grammar symbol is a property that is determined by the attributes of the symbols below it in the parse tree.

- In other words, if  $A \rightarrow \alpha$  is a production, then  $A$ 's synthesized attributes are determined by the attributes of the symbols in  $\alpha$ .



# Synthesized Attributes

- If the AST represents a numerical expression, then the value of the root node is determined by the values of the nodes below it in the tree.
- Thus, the value of the root node is a synthesized attribute.



# Example

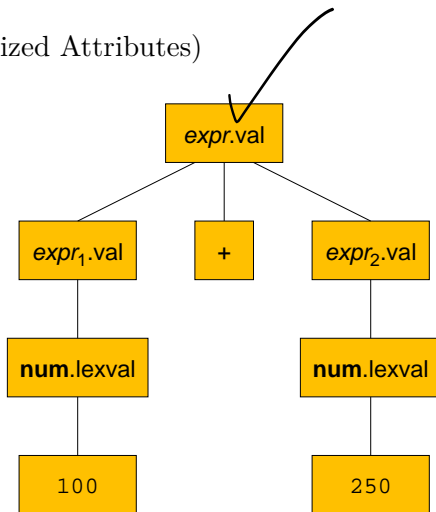
## Example (Synthesized Attributes)

- The terminals get their values directly from the lexical analyzer.
- For example, a **num** token's value attribute would be the numerical value of the string of digits in the token.



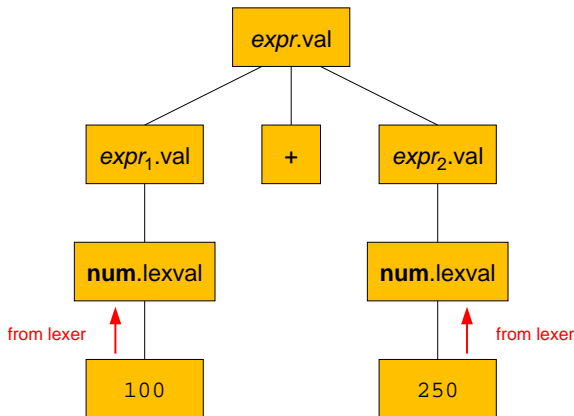
# Example

Example (Synthesized Attributes)



# Example

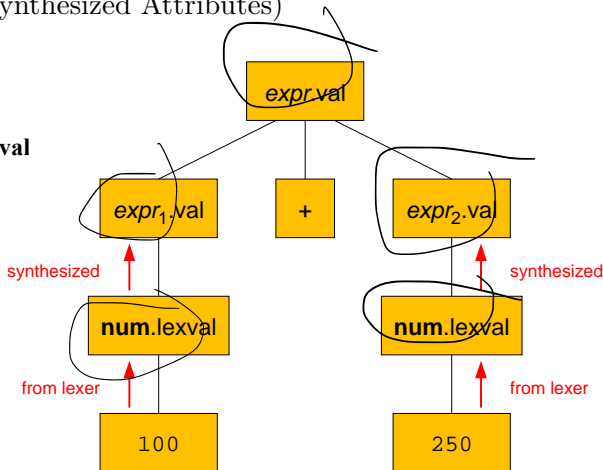
Example (Synthesized Attributes)



# Example

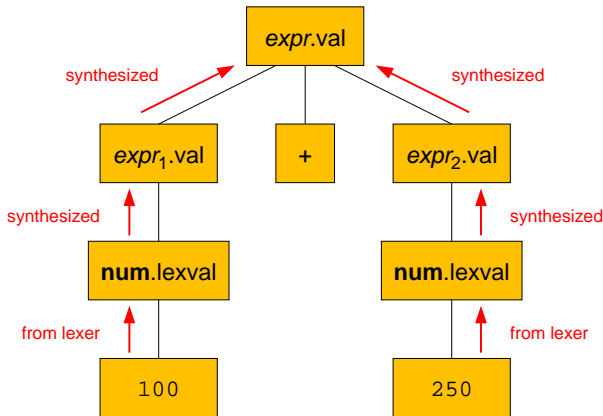
## Example (Synthesized Attributes)

**E.val = num.lexval**



# Example

## Example (Synthesized Attributes)





# Outline

- 1 Abstract Syntax Trees
- 2 Syntax-Directed Definitions
- 3 Synthesized Attributes
- 4 Inherited Attributes**
- 5 Examples
- 6 Assignment



# Inherited Attributes

## Definition

An **inherited attribute** is a property of a symbol (node) that is determined by its parent node and its siblings in the parse tree.

- In other words, if  $\beta$  is symbol on the right side of the production  $A \rightarrow \alpha\beta\gamma$ , then  $\beta$ 's inherited attributes are determined by the attributes of  $A$  and the other symbols in  $\alpha$  and  $\gamma$ .



# Example

`int a, b, c;`

`dcl => type list => int list => int list, id => int list, id, id => int id, id, id`

## Example (Inherited Attributes)

- Consider the grammar for a declaration containing one or more identifiers.

`int a, b, c;`

*dcl* → *type list*  
*list* → *list* , **id** | **id**  
*type* → **int** | **float**



# Example

## Example (Inherited Attributes)

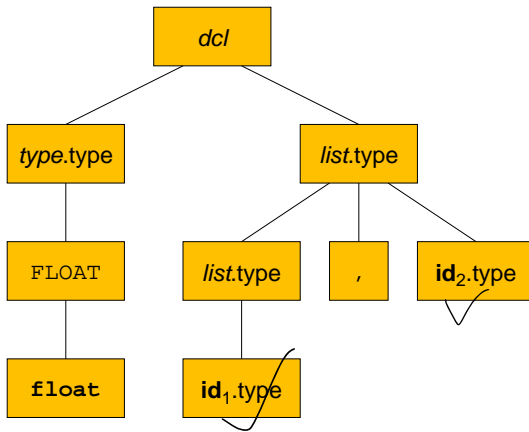
- For example, the declaration might be  
`float a, b, c;`
- The attribute “float” first appears as the type of the **float** token.
- From there it is passed to the identifiers **a**, then **b**, then **c**.



# Example

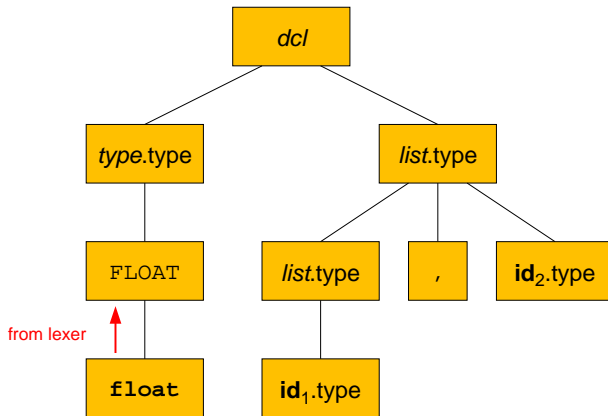
Example (Inherited Attributes)

**float a, b**



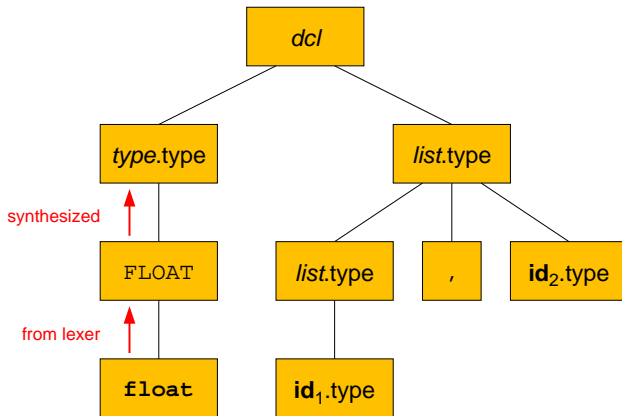
# Example

## Example (Inherited Attributes)



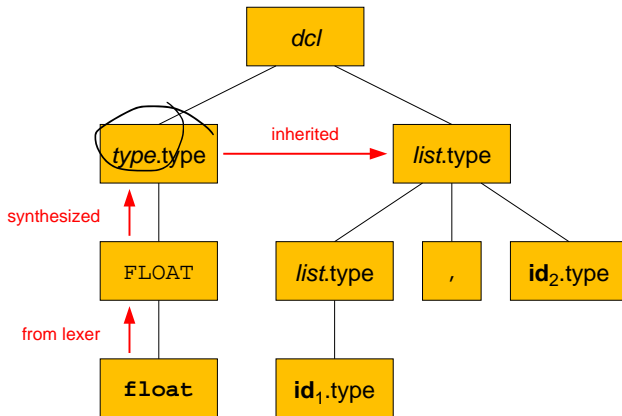
# Example

## Example (Inherited Attributes)



# Example

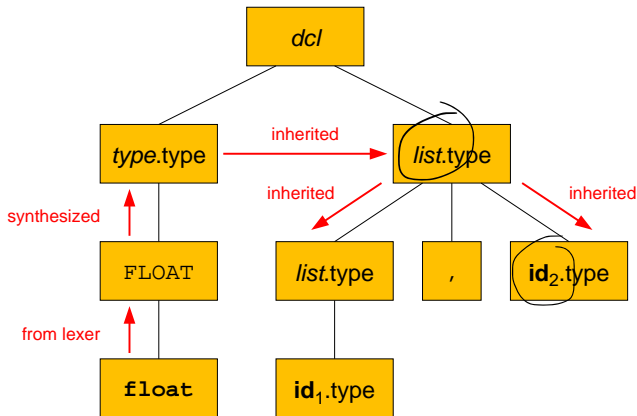
## Example (Inherited Attributes)





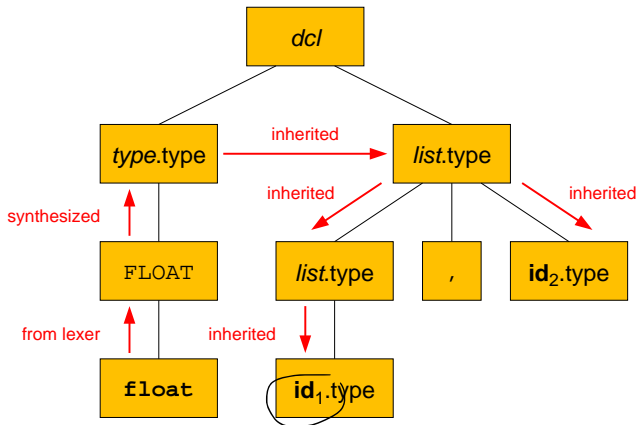
# Example

## Example (Inherited Attributes)



# Example

## Example (Inherited Attributes)



# Some Questions

## Questions

- In an expression tree, is the type of the expression at the root inherited or is it synthesized?
- Is the type used in an arithmetic operation an inherited attribute or an synthesized attribute of the operator?
- In an assignment statement, is the type assigned by the operator an inherited attribute or a synthesized attribute of the operator?



# Outline

- 1 Abstract Syntax Trees
- 2 Syntax-Directed Definitions
- 3 Synthesized Attributes
- 4 Inherited Attributes
- 5 Examples**
- 6 Assignment



# Synthesized Attributes

Example (Synthesized Attributes)

Let the grammar be

$$E \rightarrow E + T$$

$$E \rightarrow T$$

$$T \rightarrow T * F$$

$$T \rightarrow F$$

$$F \rightarrow ( E )$$

$$F \rightarrow \text{num}$$



# Synthesized Attributes

## Example (Synthesized Attributes)

- The attribute at every node is the value of the nonterminal.
- In every case, it is synthesized.

$$E.val = E.val + T.val$$

$$E.val = T.val$$

$$T.val = T.val \times F.val$$

$$T.val = F.val$$

$$F.val = E.val$$

$$F.val = \mathbf{num.lexval}$$

# Inherited Attributes

Example (Inherited Attributes)

Let the grammar be

$$E \rightarrow T E'$$

$$E' \rightarrow + T E'$$

$$E' \rightarrow \varepsilon$$

$$T \rightarrow F T'$$

$$T' \rightarrow * F T'$$

$$T' \rightarrow \varepsilon$$

$$F \rightarrow ( E )$$

$$F \rightarrow \text{num}$$

# Inherited Attributes

## Example (Inherited Attributes)

- The attribute at the nodes  $E$ ,  $T$ , and  $F$  is the value of the nonterminal.
- In some cases, it is synthesized.
- In other cases, it is inherited.





# Inherited Attributes

## Example (Inherited Attributes)

- For the production

$$F \rightarrow \text{num}$$

we have the rule

$$F.\text{val} = \text{num.lexval}$$

- For the production

$$F \rightarrow ( E )$$

we have the rule

$$F.\text{val} = E.\text{val}$$

# Inherited Attributes

Example (Inherited Attributes)

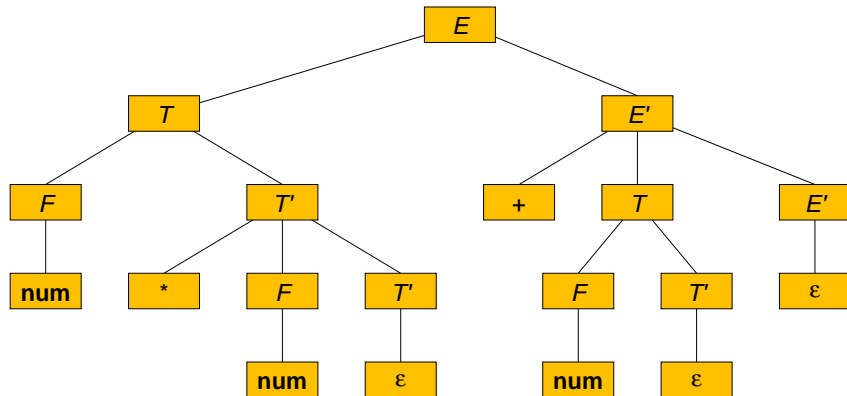
Consider the parse tree for the expression

$$3 * 4 + 5.$$



# Inherited Attributes

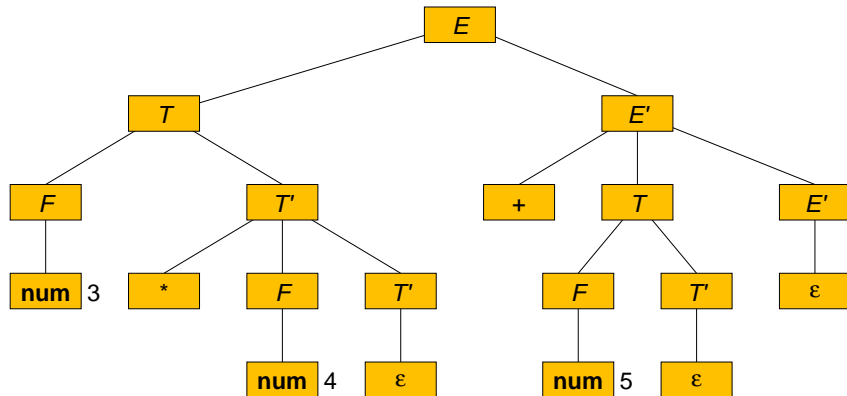
## Example (Inherited Attributes)



The parse tree

# Inherited Attributes

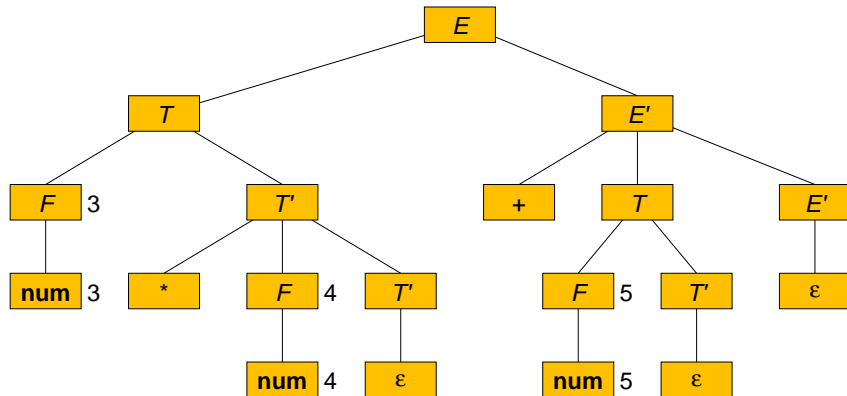
## Example (Inherited Attributes)



**num** gets its values from the lexer

# Inherited Attributes

## Example (Inherited Attributes)



$$F.val = num.lexval$$

# Inherited Attributes

Example (Inherited Attributes)

$$T'.inh = F.val$$

# Inherited Attributes

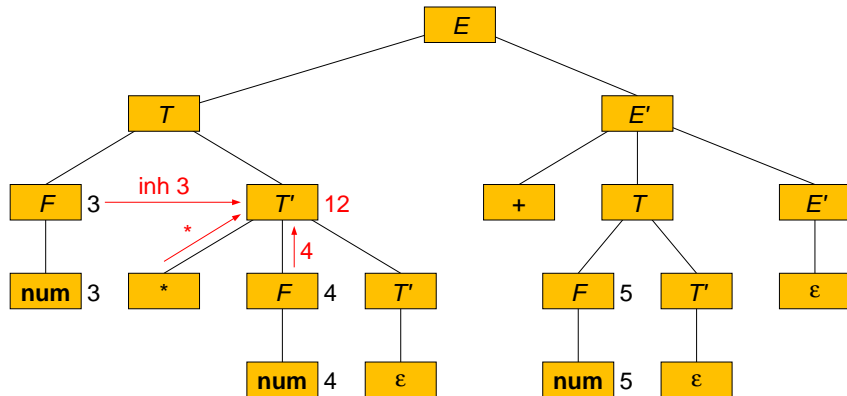
## Example (Inherited Attributes)

- How does  $T$  (in the production  $T \rightarrow F T'$ ) get its value?
- It must multiply 3 and 4 to get 12.
- So, first  $T'$  inherits 3 from  $F$ .
- Then, in the production  $T' \rightarrow * F T'_1$ ,  $T'_1$  inherits 12 from  $T'$  and  $F$ .
- Then,  $T'$  turns around and synthesizes 12 from  $T'_1$ .
- Then, back in the production  $T \rightarrow F T'$ ,  $T$  synthesizes 12 from  $T'$ .



# Inherited Attributes

## Example (Inherited Attributes)

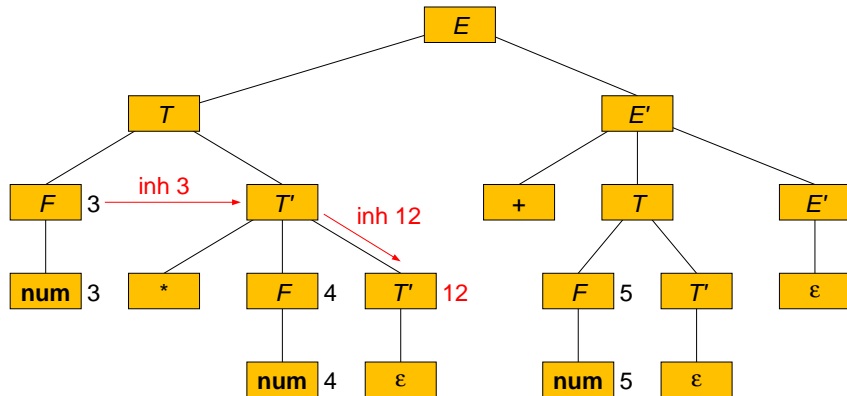


$$T'.inh = F.val$$



# Inherited Attributes

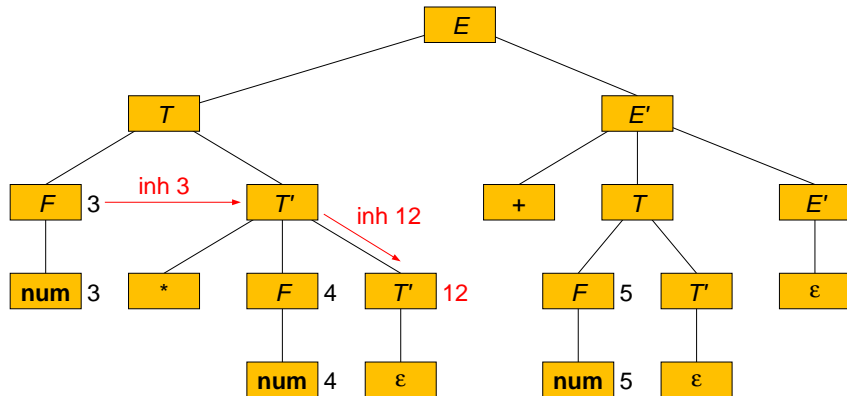
## Example (Inherited Attributes)



$$T_1'.inh = T'.inh \times F.val$$

# Inherited Attributes

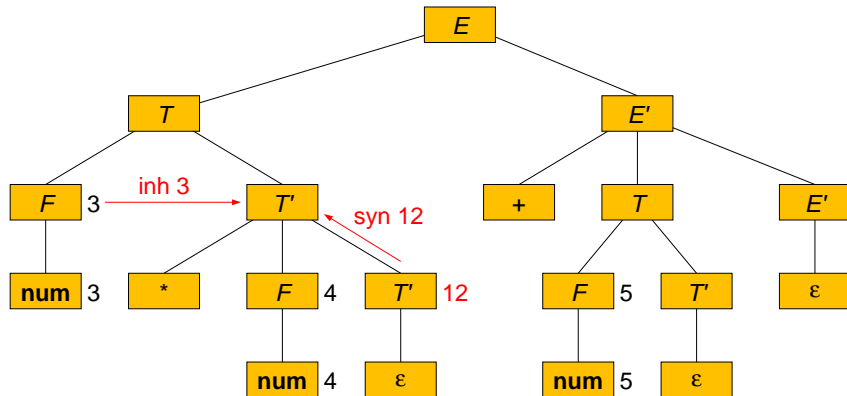
## Example (Inherited Attributes)



$$T'.syn = T'.inh$$

# Inherited Attributes

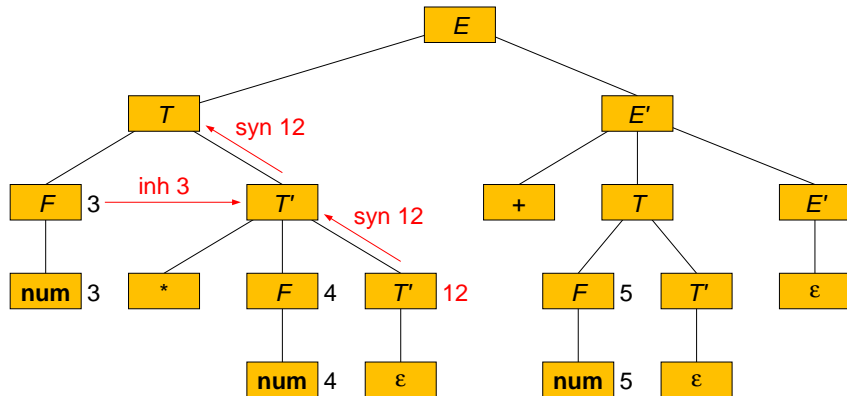
## Example (Inherited Attributes)



$$T'.\text{syn} = T'_1.\text{syn}$$

# Inherited Attributes

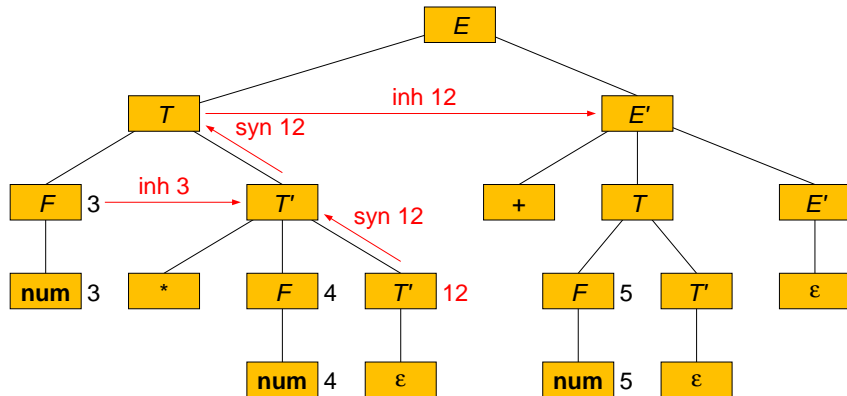
## Example (Inherited Attributes)



$$T.val = T'.syn$$

# Inherited Attributes

## Example (Inherited Attributes)



$$E'.inh = T.val$$

# Inherited Attributes

## Example (Inherited Attributes)

- We now have the rules

Production	Semantic Rules
$T \rightarrow F T'$	$T'.inh = F.val$ $T.val = T'.syn$
$T' \rightarrow * F T'_1$	$T'_1.inh = T'.inh \times F.val$ $T'.syn = T'_1.syn$
$T' \rightarrow \varepsilon$	$T'.syn = T'.inh$
$F \rightarrow \mathbf{num}$	$F.val = \mathbf{num.lexval}$



# Inherited Attributes

## Example (Inherited Attributes)

- As well as the rules

Production	Semantic Rules
$E \rightarrow T E'$	$E'.inh = T.val$ $E.val = E'.syn$
$E' \rightarrow + T E'_1$	$E'_1.inh = E'.inh + T.val$ $E'.syn = E'_1.syn$
$E' \rightarrow \varepsilon$	$E'.syn = E'.inh$



# Outline

- 1 Abstract Syntax Trees
- 2 Syntax-Directed Definitions
- 3 Synthesized Attributes
- 4 Inherited Attributes
- 5 Examples
- 6 Assignment**





# Assignment

## Homework

- p. 309: 1, 2, 3

