Syntax Directed Translation

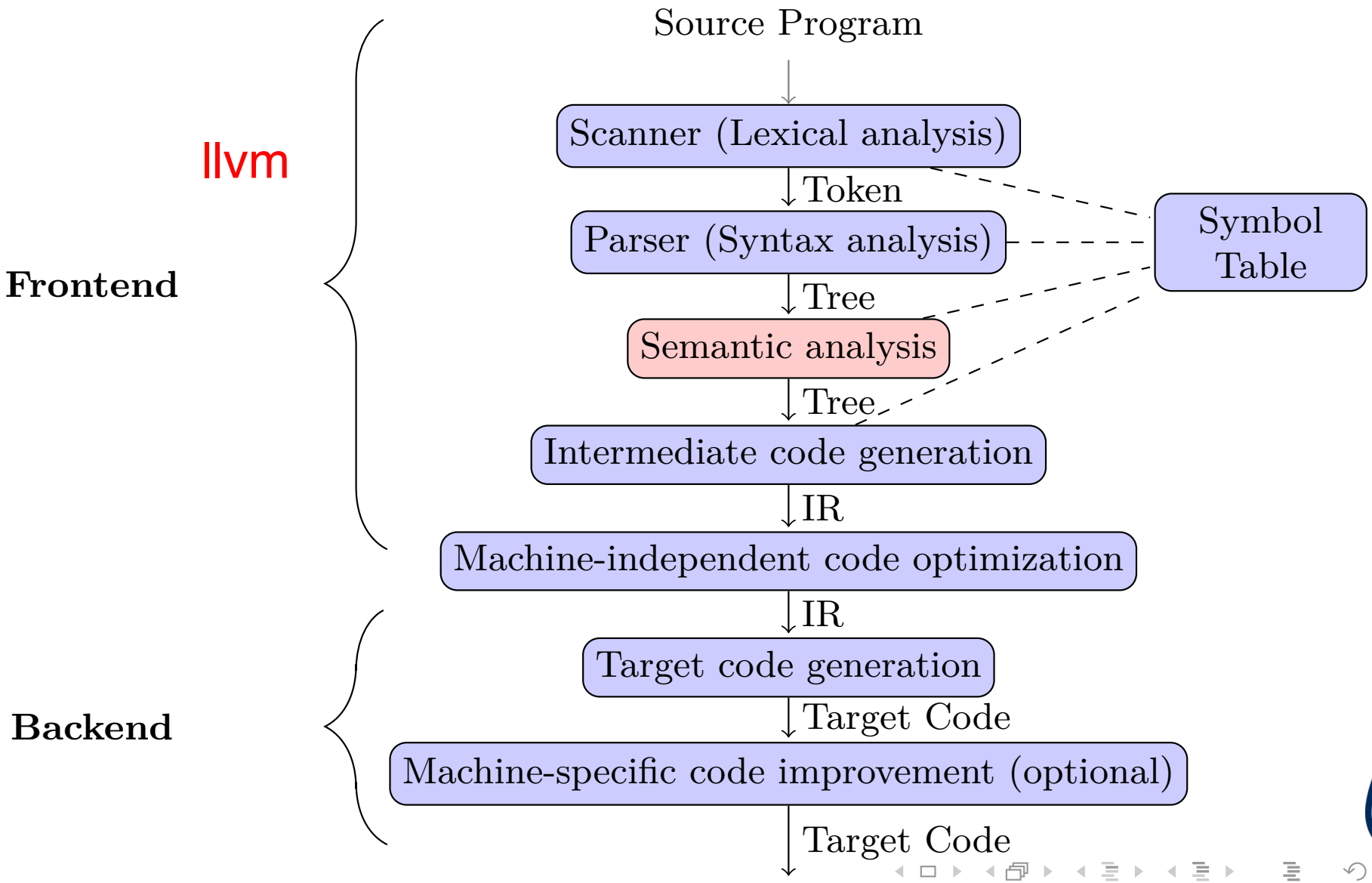
Lecture 10
Section 5.1 - 5.3

Rokan Uddin Faruqui

Associate Professor
Dept of Computer Science and Engineering
University of Chittaong, Bangladesh
Email: *rokan@cu.ac.bd*

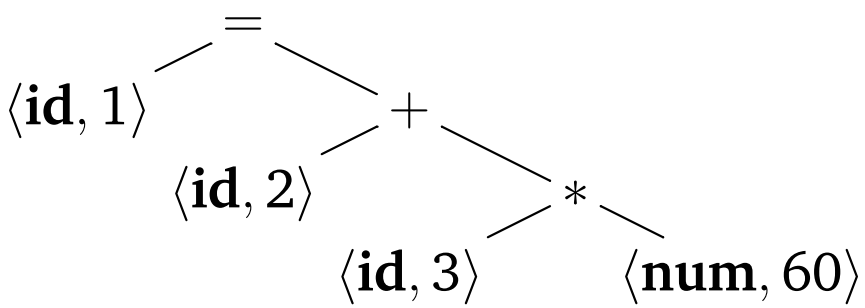
Outline

The Phases of Compilation



From Parse Tree/ Abstract Syntax Tree (AST)

```
position = initial + rate * 60
```

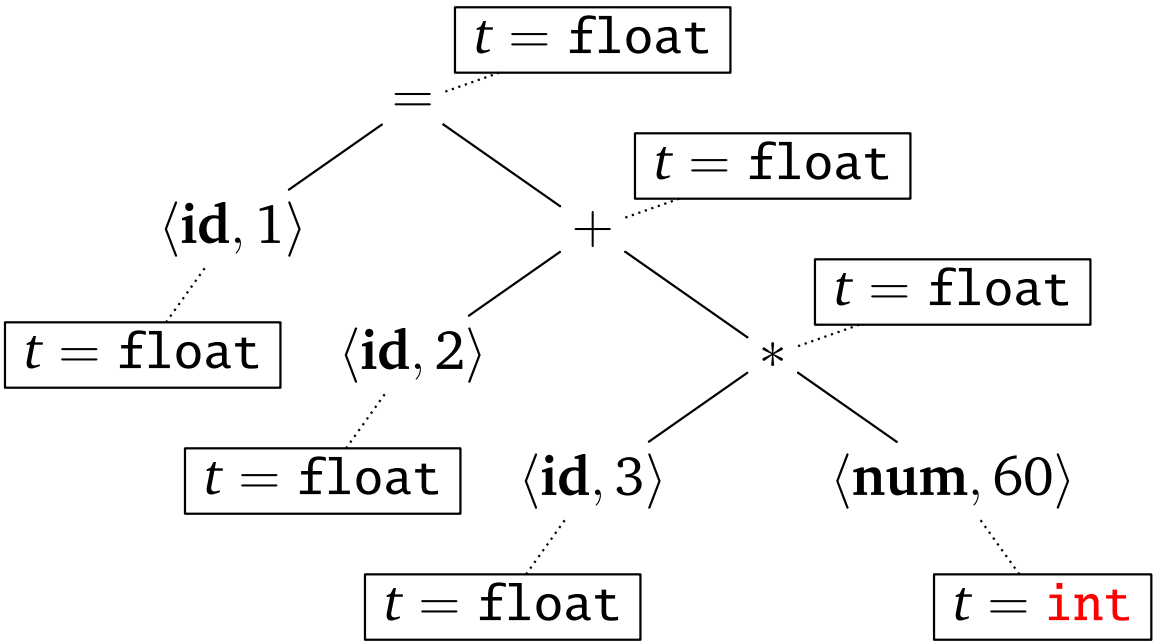


id	*lexeme*
1	`position`
2	`initial`
3	`rate`

Semantic Analysis

- Last "Front end" phase
- Catching all remaining errors

Type Checking, semantic analysis that results in annotated tree



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

Parse Trees

- 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.

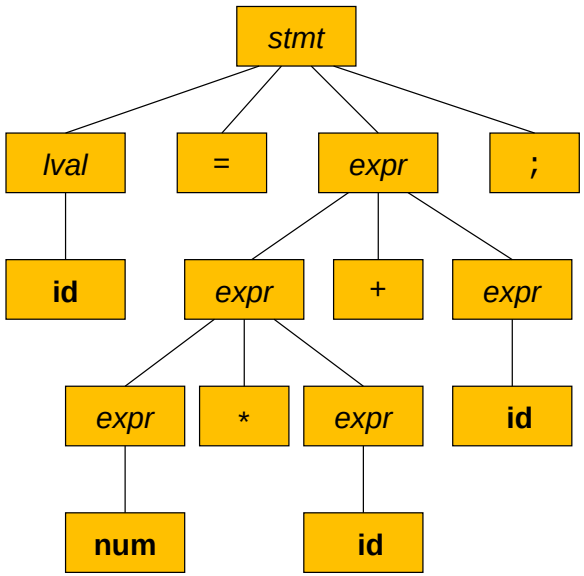
Abstract Syntax Trees

- 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.

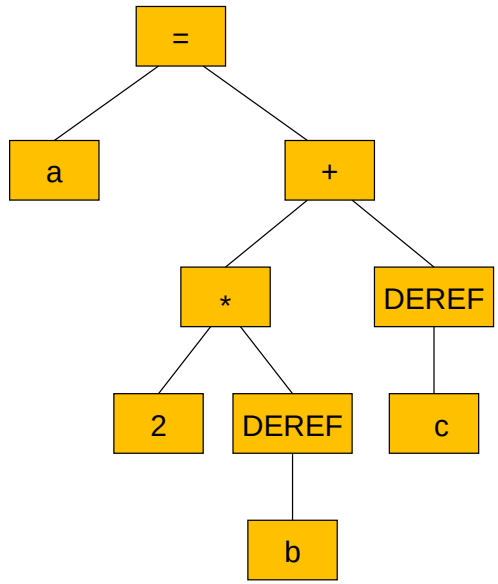
Syntax Trees vs. Parse Trees

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

Syntax-Directed Definitions

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

$$E \to E + E \mid \mathbf{num}$$

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

$$E.\text{val} = E_1.\text{val} + E_2.\text{val}$$
$$E.\text{val} = \mathbf{num}.\text{lexval}$$

Syntax-Directed Definitions

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

Outline

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 α.

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.

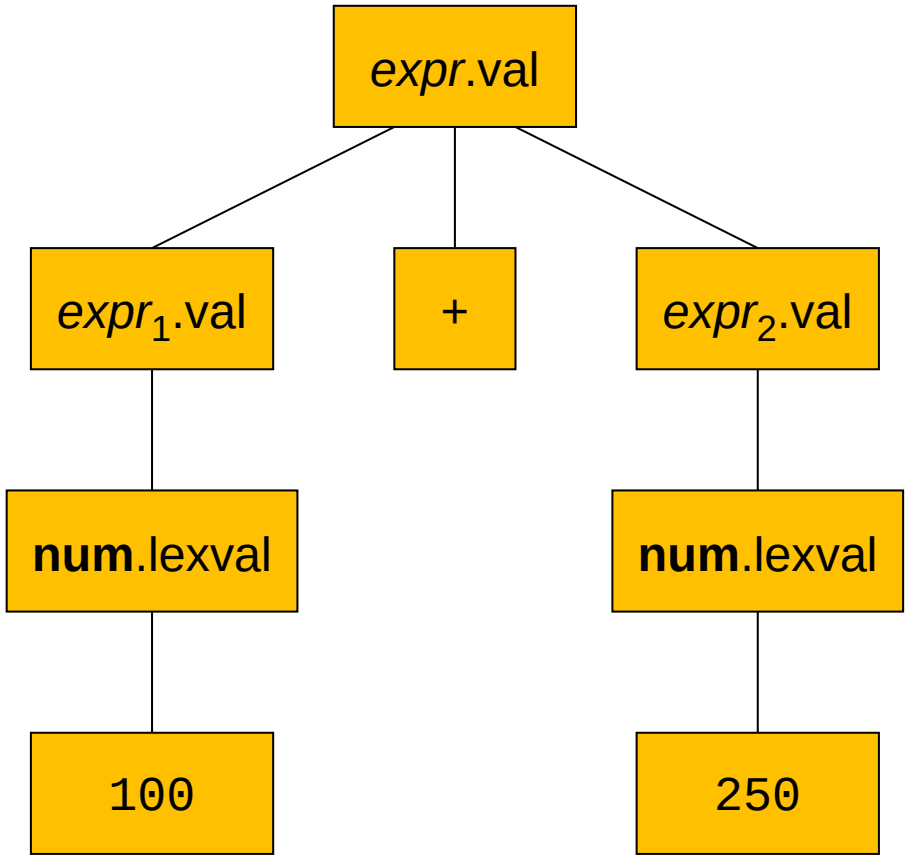
Example

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.

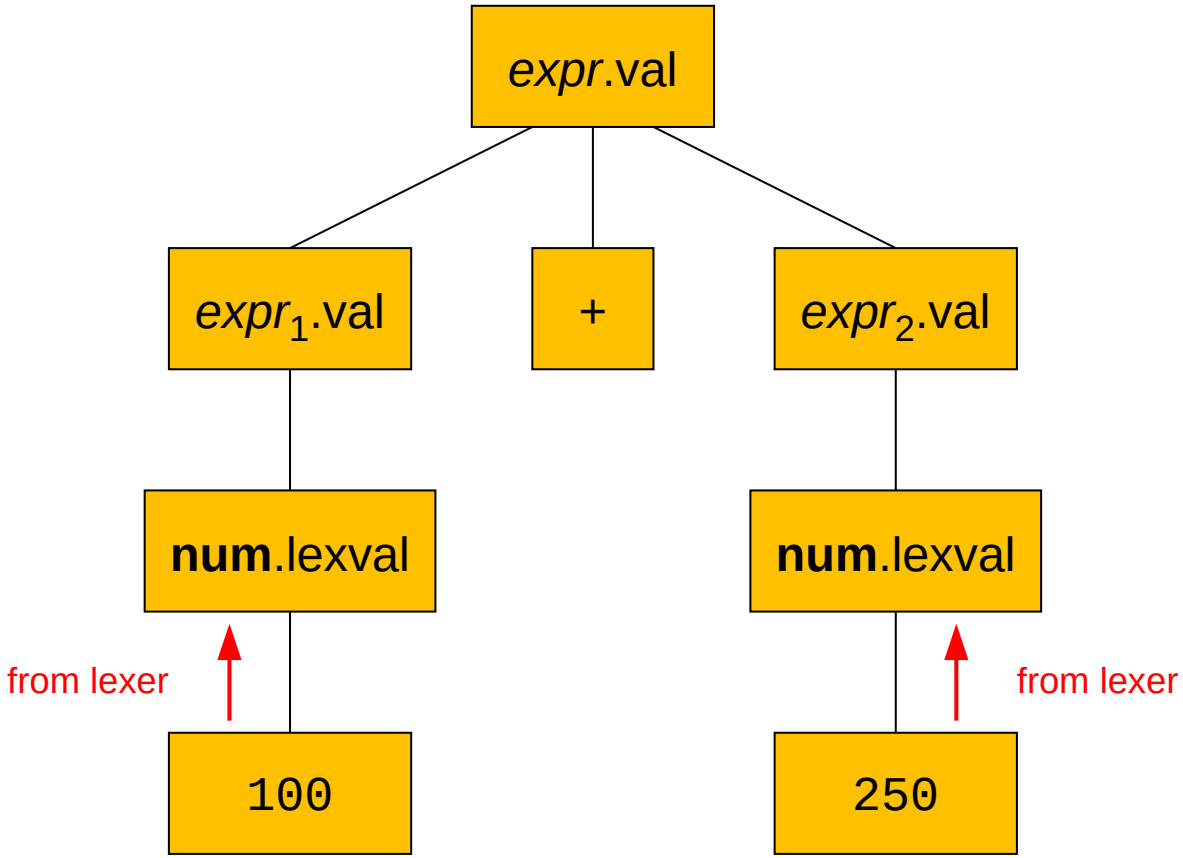
Example

Example (Synthesized Attributes)



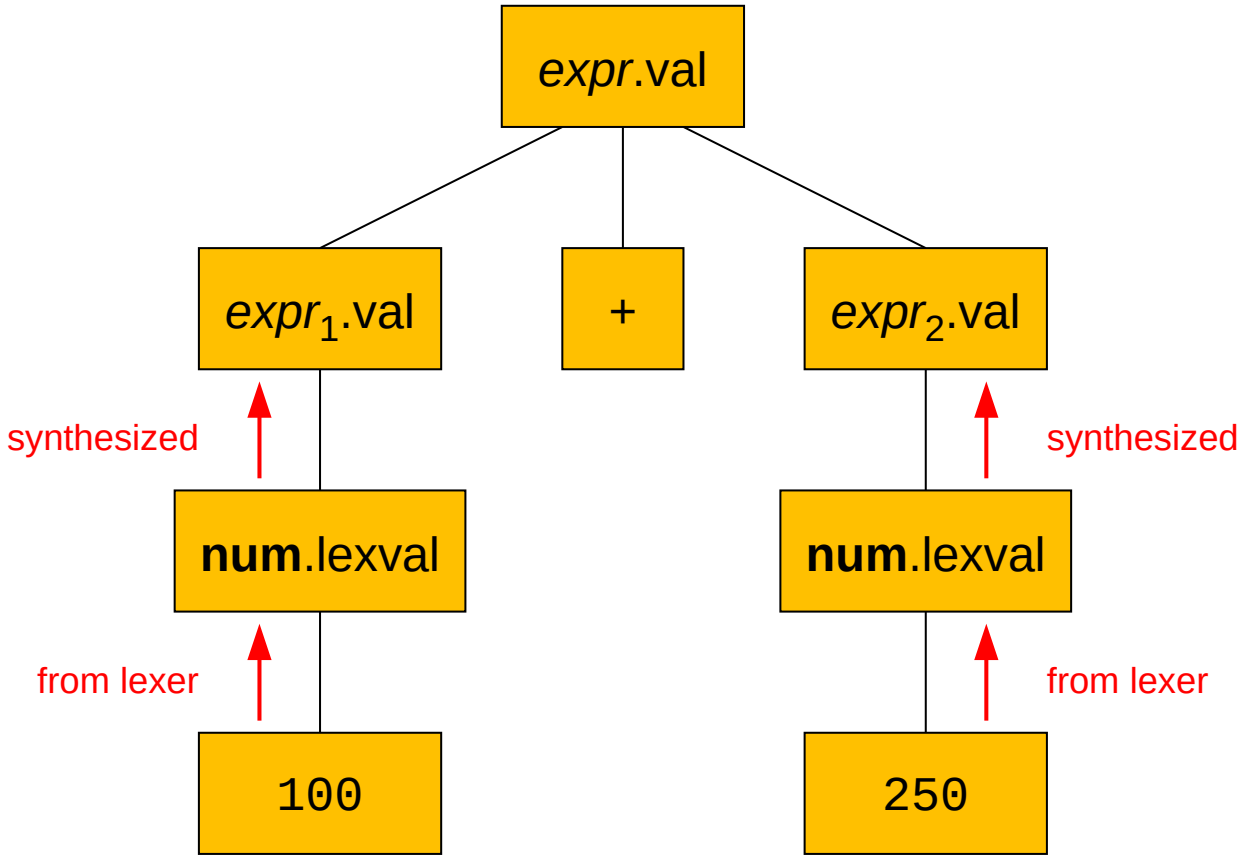
Example

Example (Synthesized Attributes)



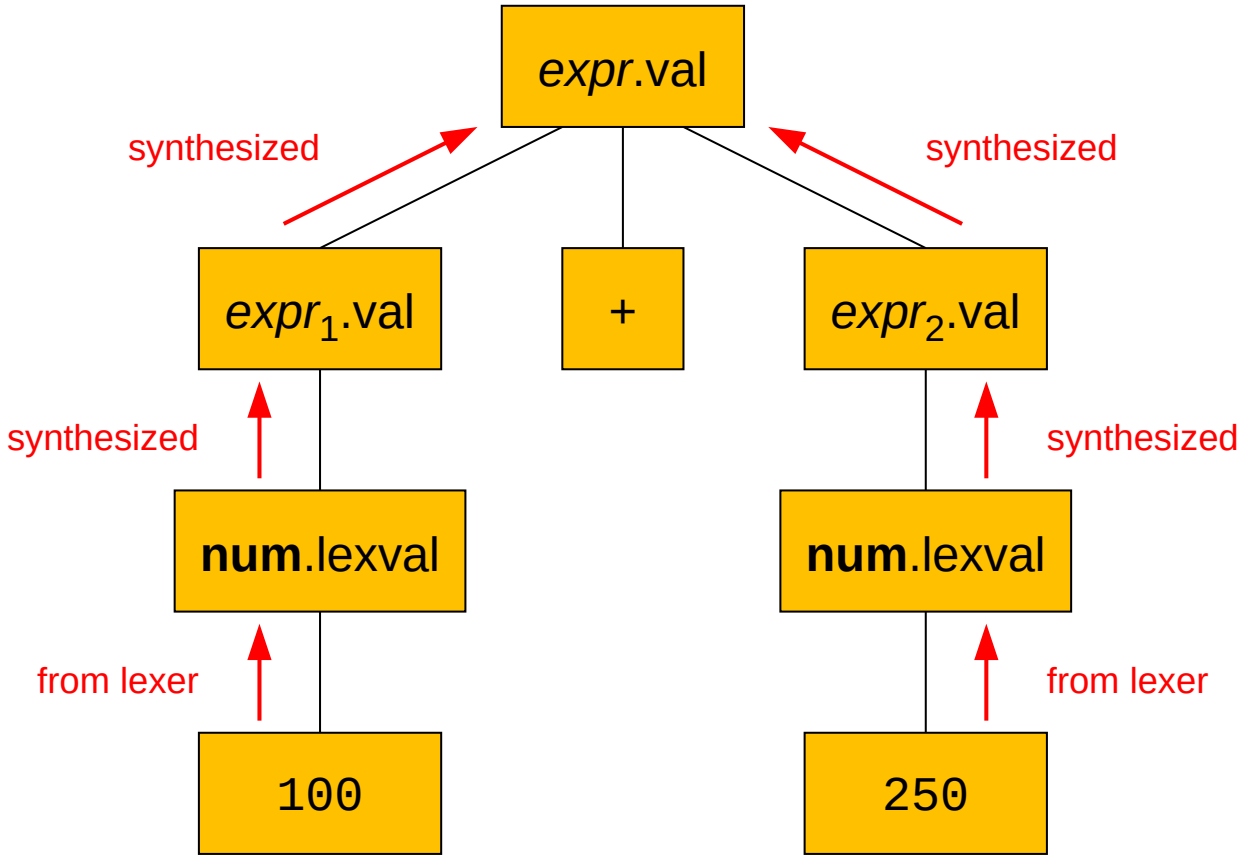
Example

Example (Synthesized Attributes)



Example

Example (Synthesized Attributes)



Outline

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

Example

Example (Inherited Attributes)

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

$$dcl \rightarrow type\ list$$
$$list \rightarrow list\ \textbf{,}\ \textbf{id} \mid \textbf{id}$$
$$type \rightarrow \textbf{int} \mid \textbf{float}$$

Example

Example (Inherited Attributes)

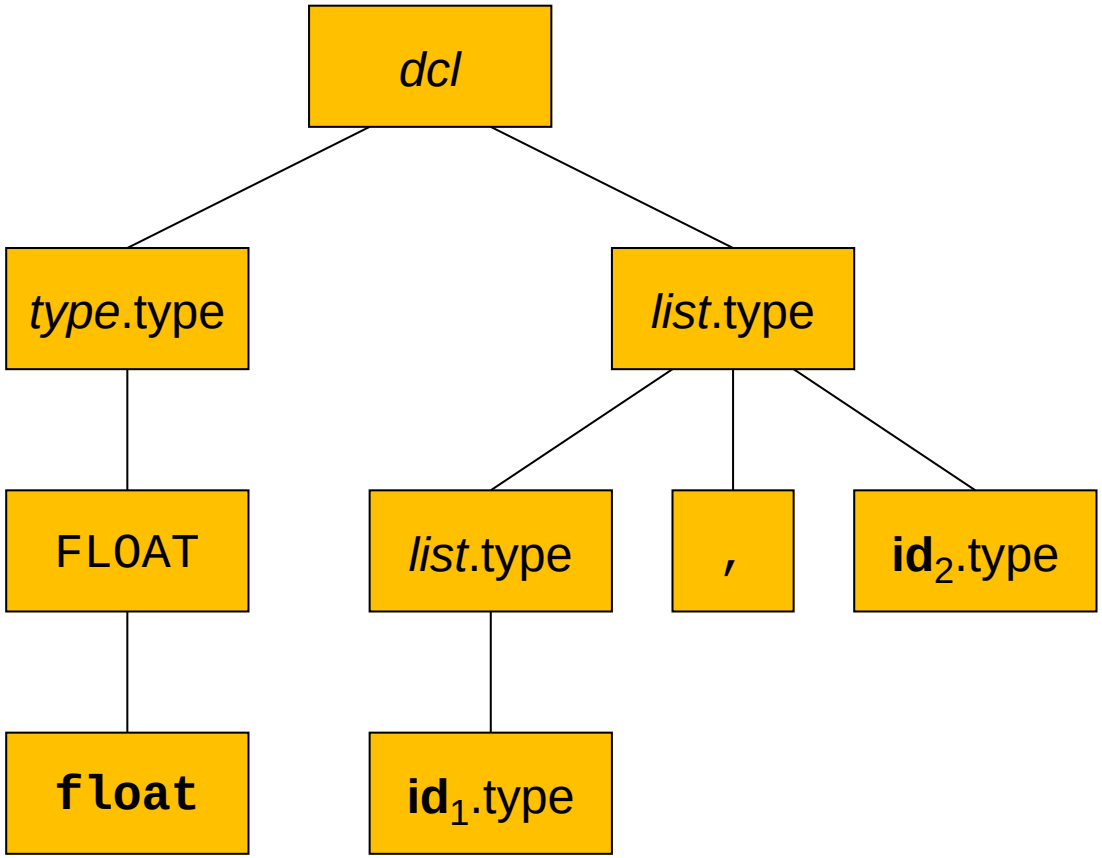
- For example, the declaration might be

  ```
  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`.

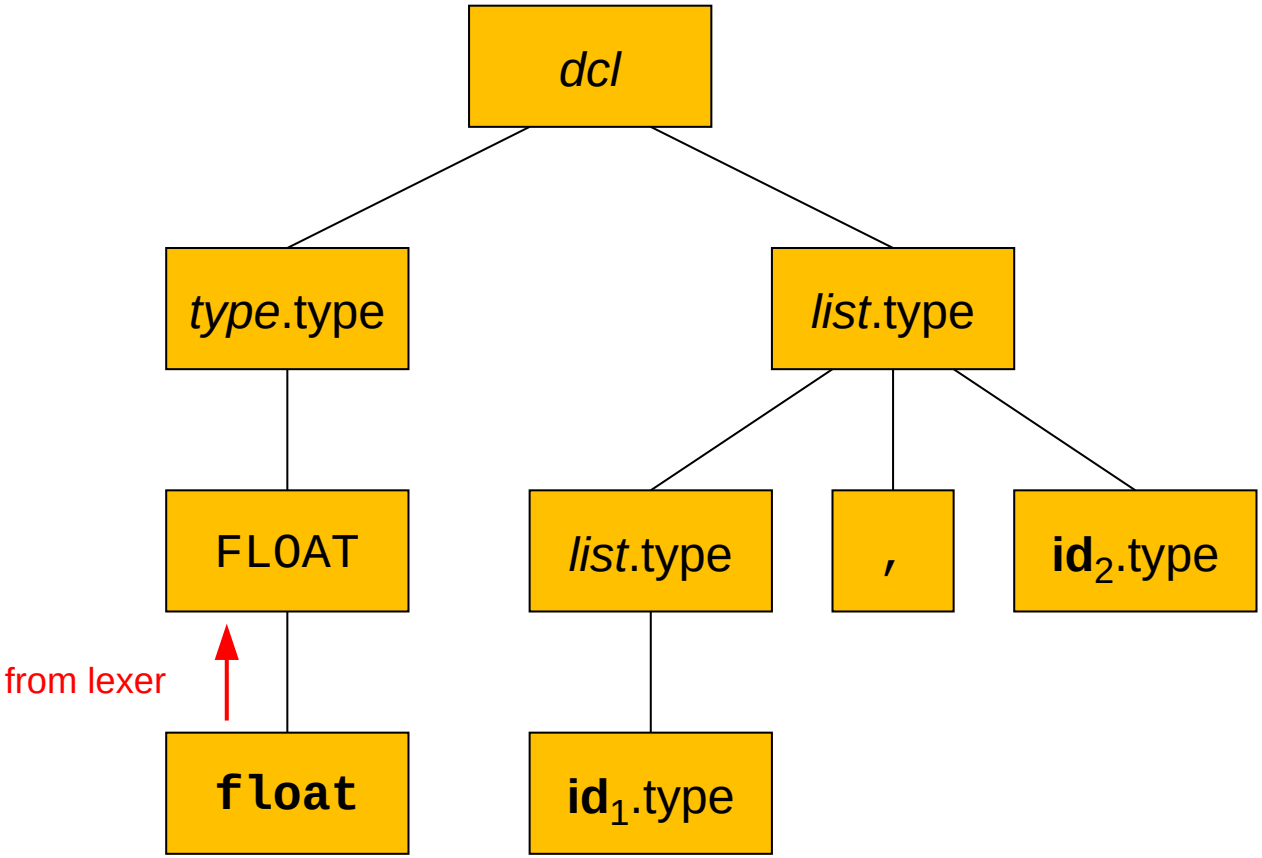
Example

Example (Inherited Attributes)



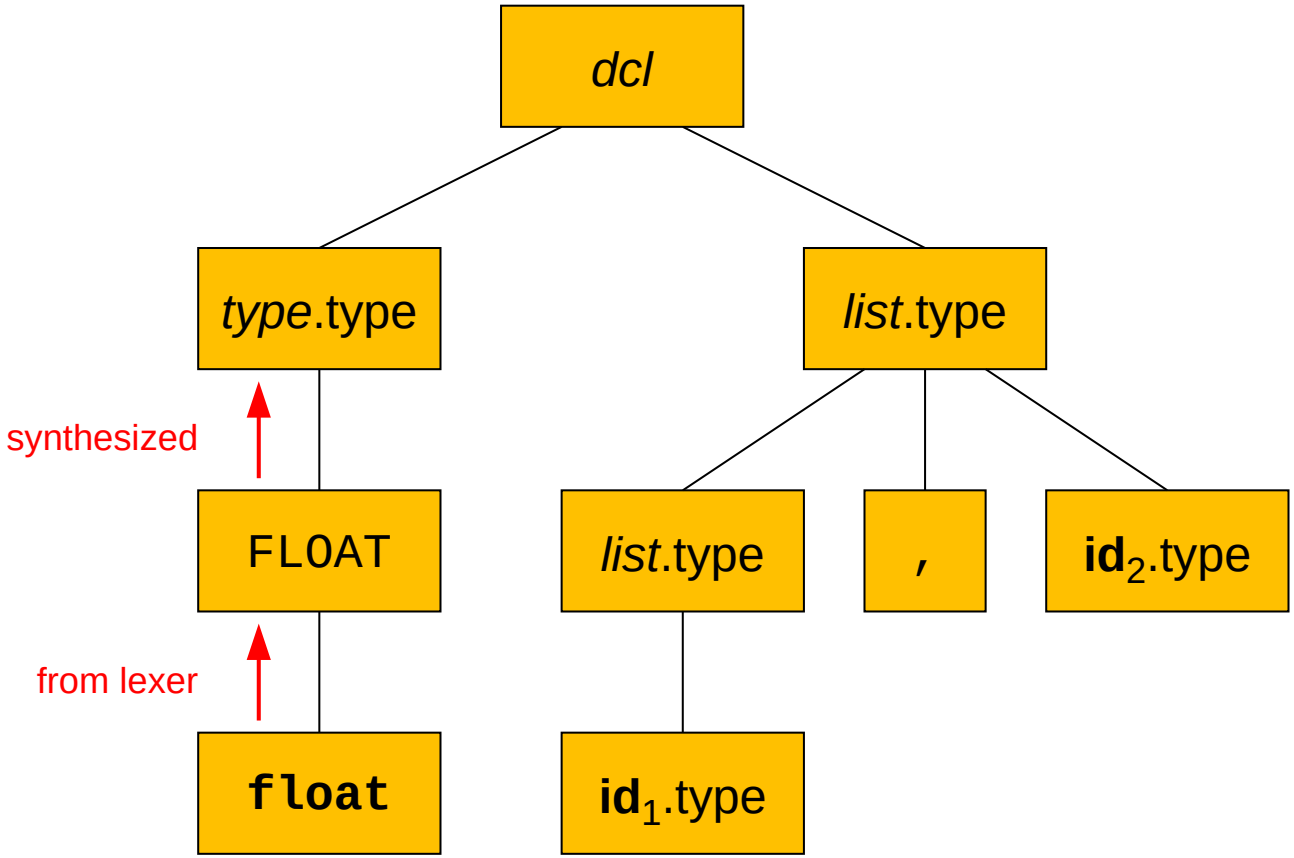
Example

Example (Inherited Attributes)



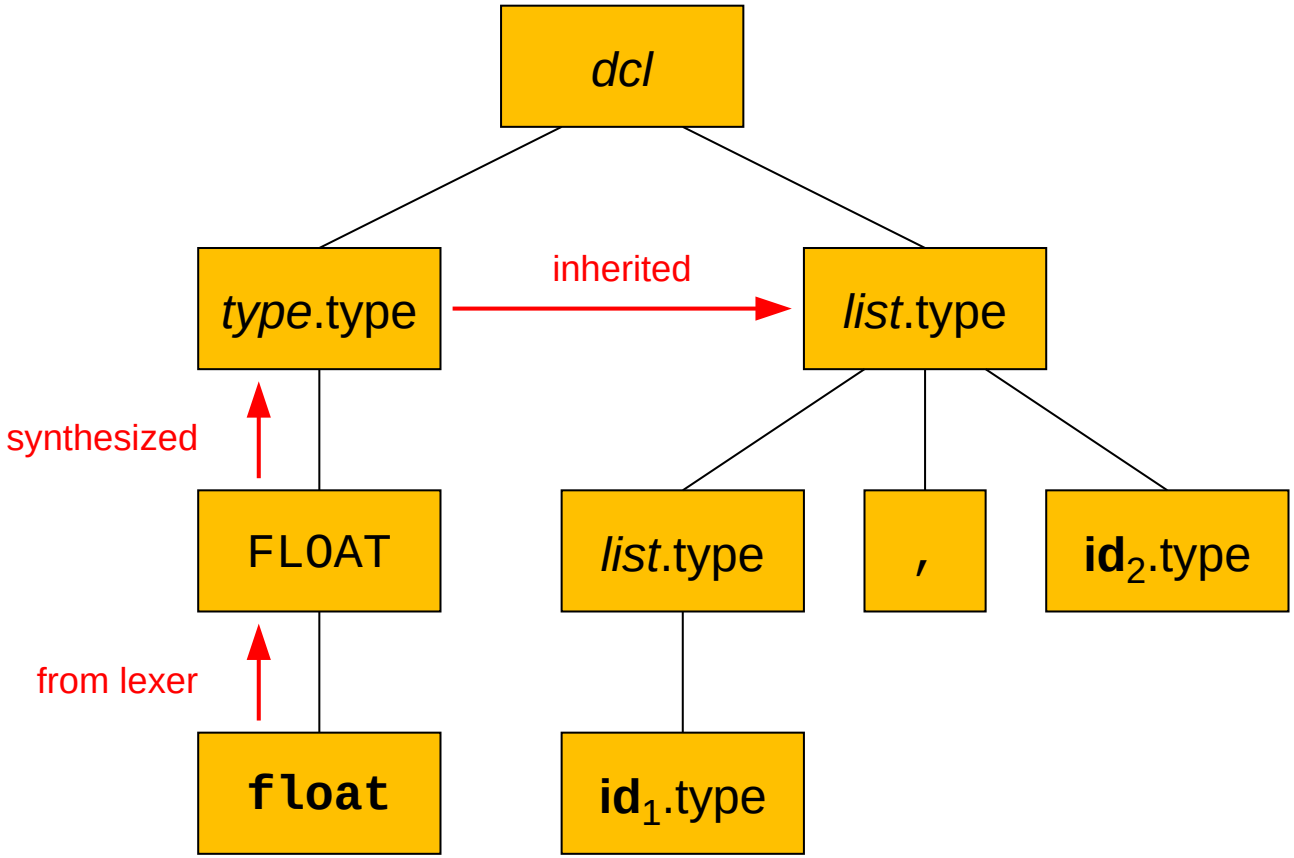
Example

Example (Inherited Attributes)



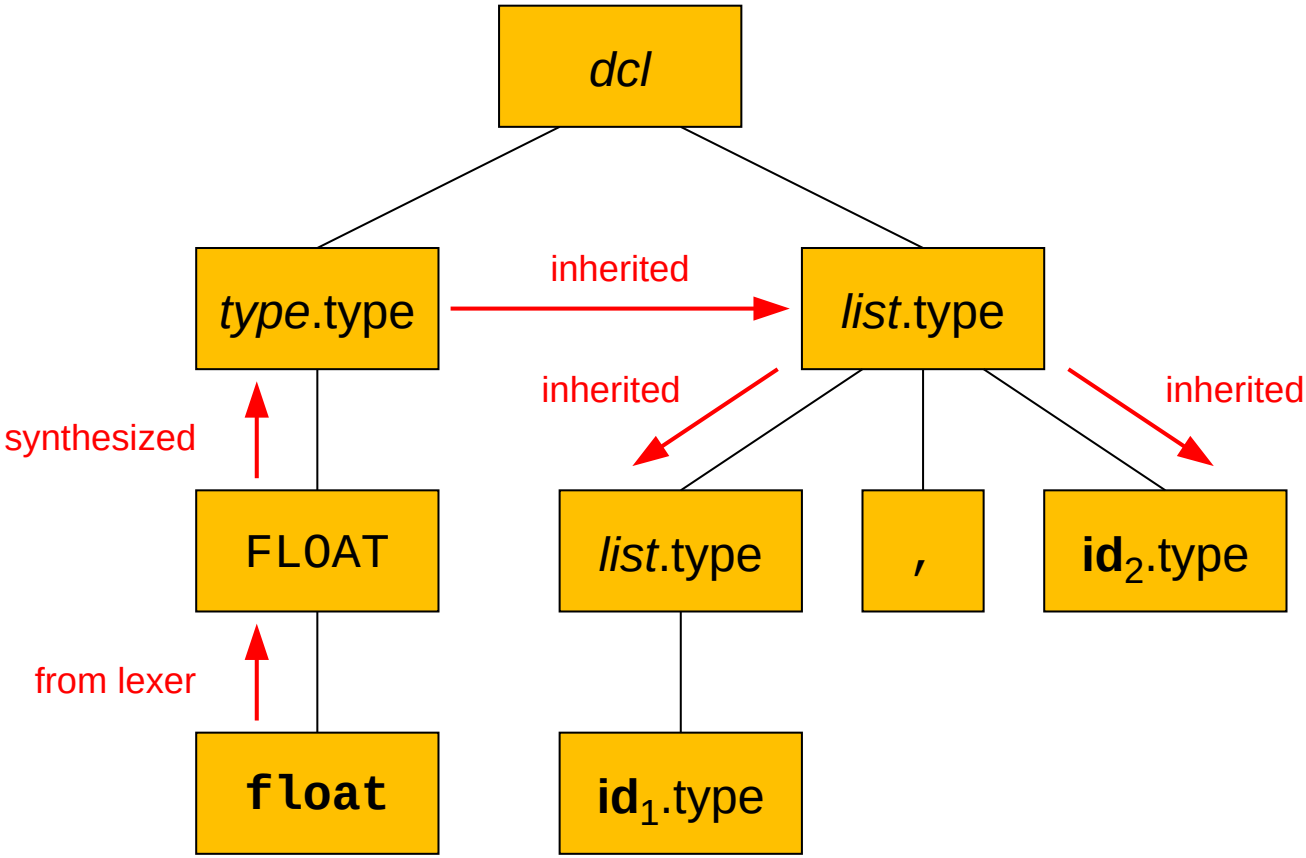
Example

Example (Inherited Attributes)



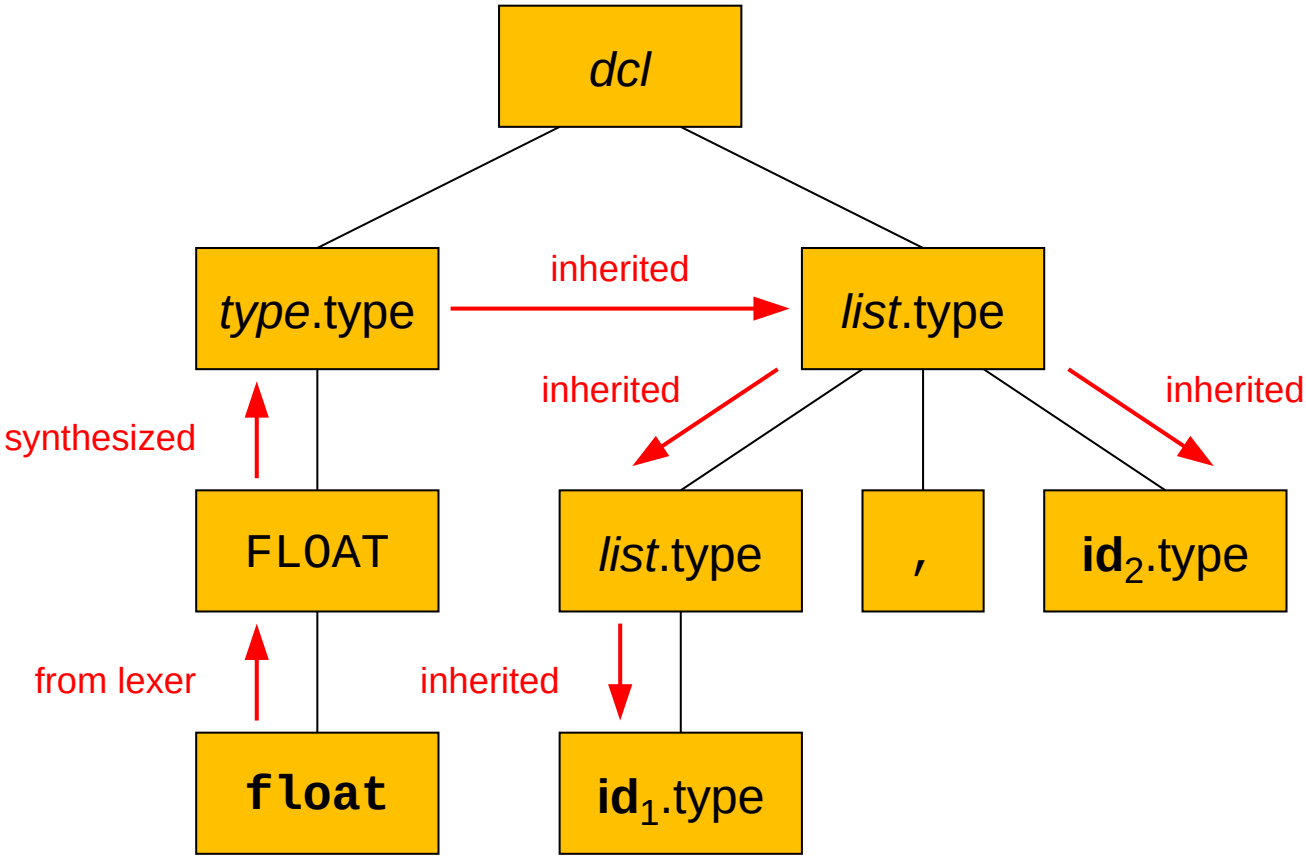
Example

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

Synthesized Attributes

Example (Synthesized Attributes)

Let the grammar be

$$E \rightarrow E + T$$
$$E \rightarrow T$$
$$T \rightarrow T * F$$
$$T \rightarrow F$$
$$F \rightarrow (E)$$
$$F \rightarrow \mathbf{num}$$

Synthesized Attributes

Example (Synthesized Attributes)

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

$$E.\text{val} = E.\text{val} + T.\text{val}$$
$$E.\text{val} = T.\text{val}$$
$$T.\text{val} = T.\text{val} \times F.\text{val}$$
$$T.\text{val} = F.\text{val}$$
$$F.\text{val} = E.\text{val}$$
$$F.\text{val} = \textbf{num}.\text{lexval}$$

Inherited Attributes

Example (Inherited Attributes) E->E + T

Let the grammar be

$$E \rightarrow T\ E'$$
$$E' \rightarrow +\ T\ E'$$
$$E' \rightarrow \varepsilon$$
$$T \rightarrow F\ T'$$
$$T' \rightarrow *\ F\ T'$$
$$T' \rightarrow \varepsilon$$
$$F \rightarrow (\ E\ )$$
$$F \rightarrow \mathbf{num}$$

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

$$F \rightarrow \mathbf{num}$$

we have the rule

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

- For the production

$$F \rightarrow (\ E\ )$$

we have the rule

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

Inherited Attributes

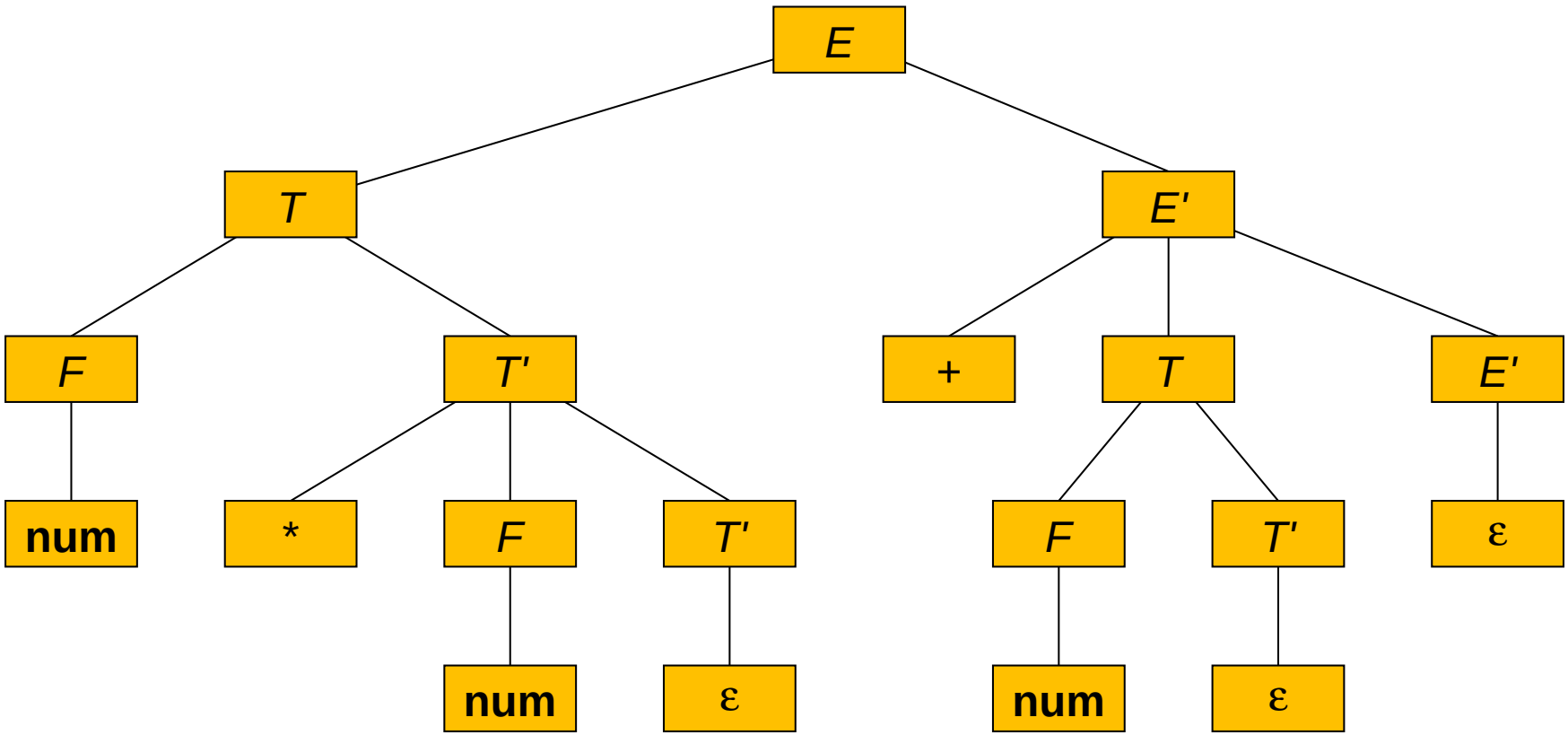
Example (Inherited Attributes)

Consider the parse tree for the expression

$$3 * 4 + 5.$$

Inherited Attributes

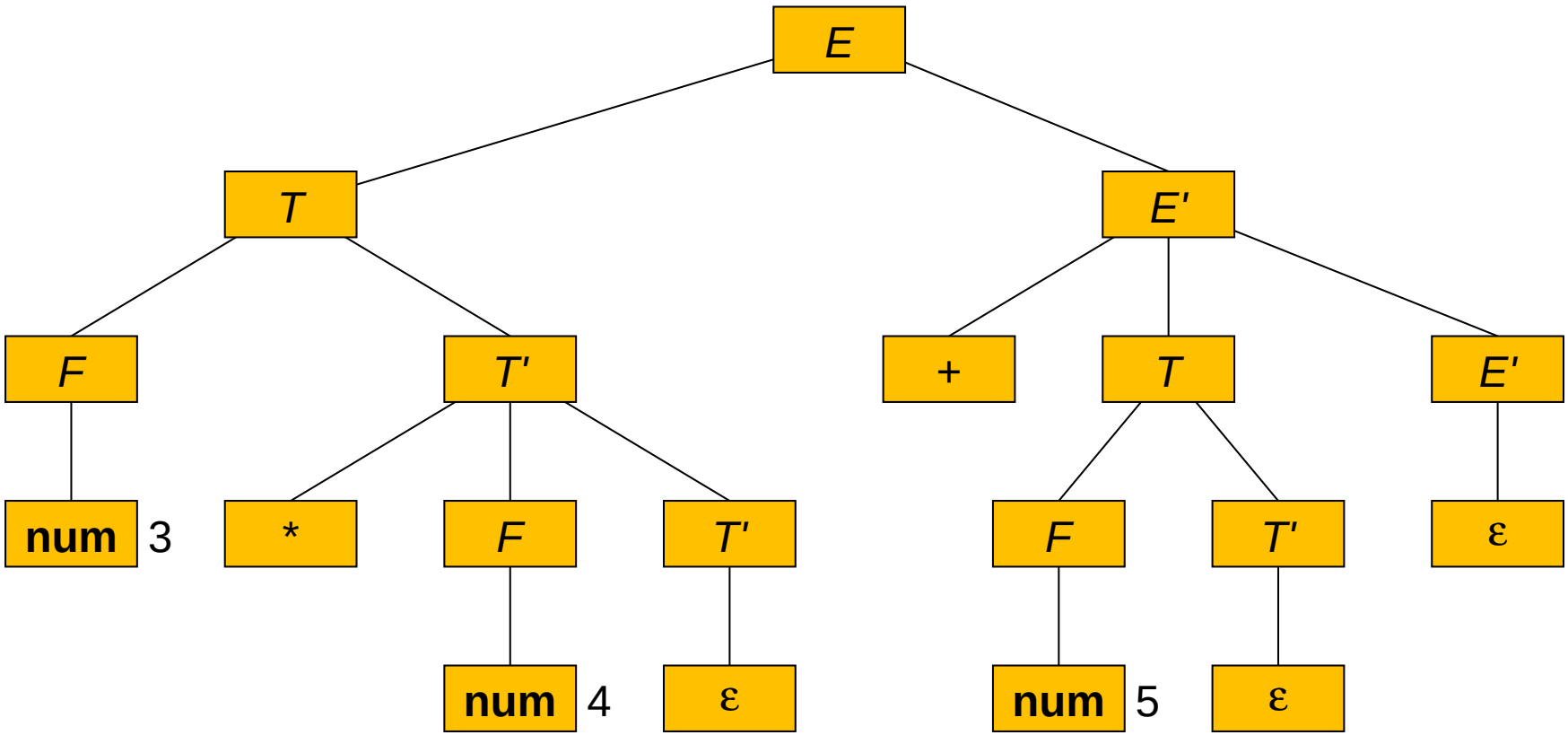
Example (Inherited Attributes)



The parse tree

Inherited Attributes

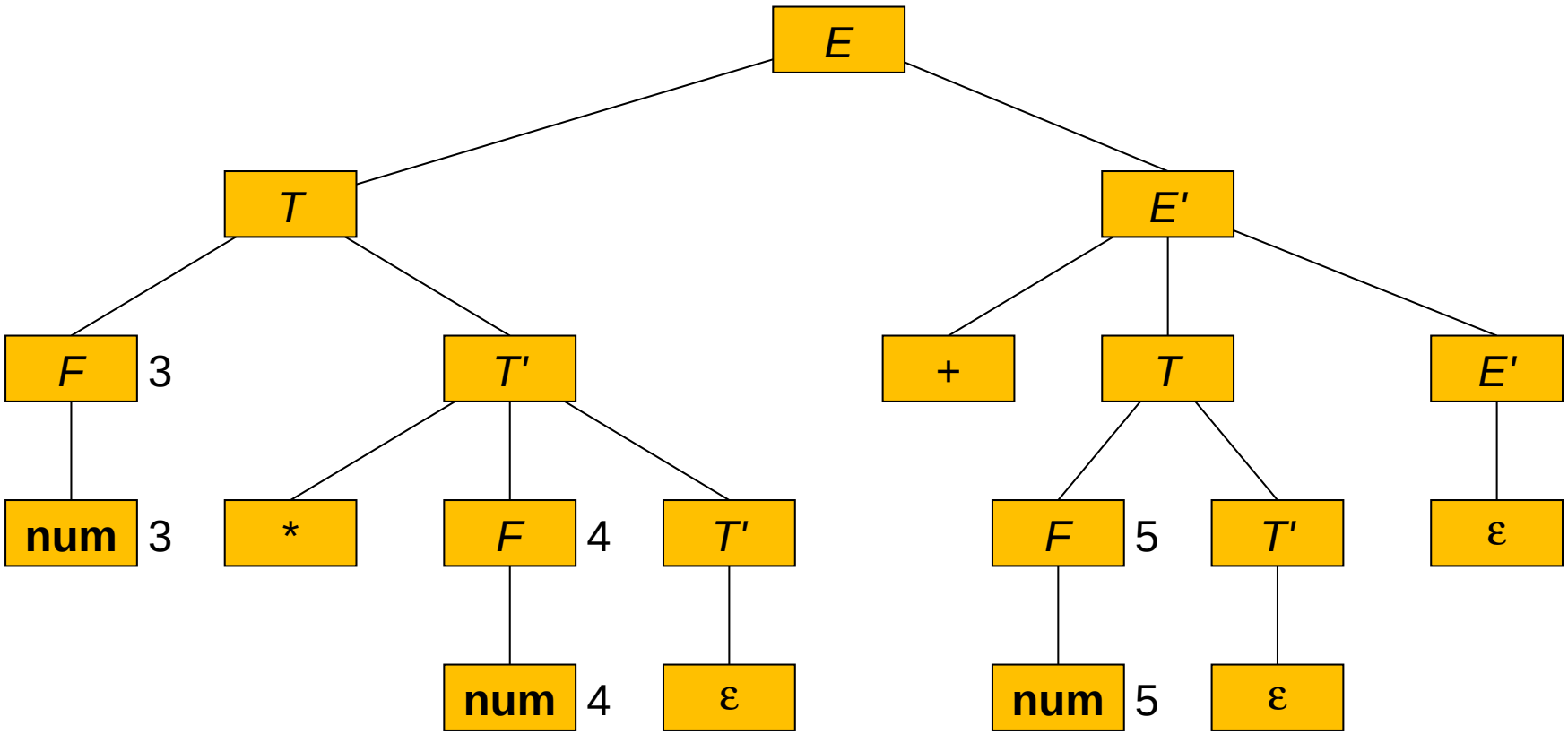
Example (Inherited Attributes)



num gets its values from the lexer

Inherited Attributes

Example (Inherited Attributes)



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

Inherited Attributes

Example (Inherited Attributes)

$$T'.\text{inh} = F.\text{val}$$

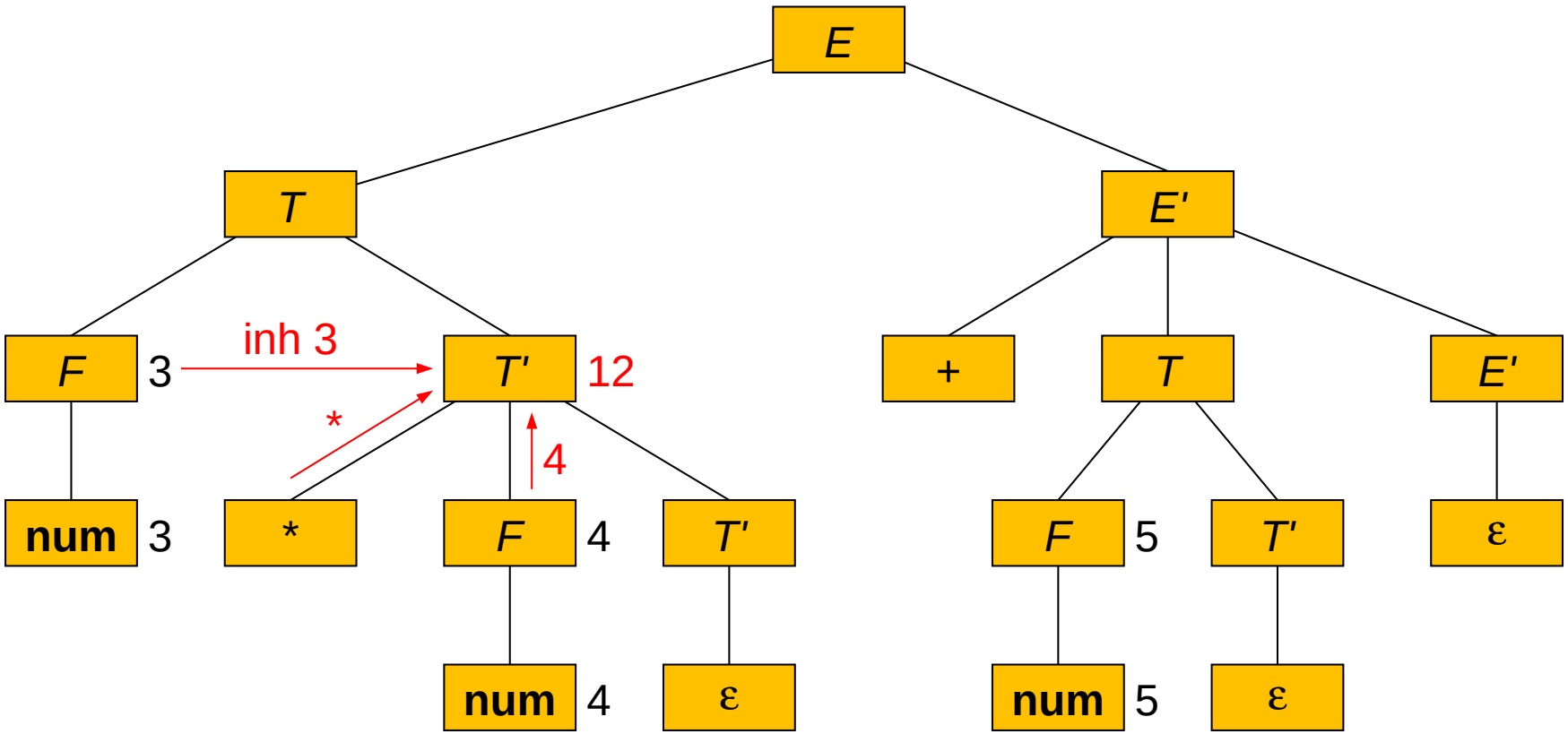
Inherited Attributes

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

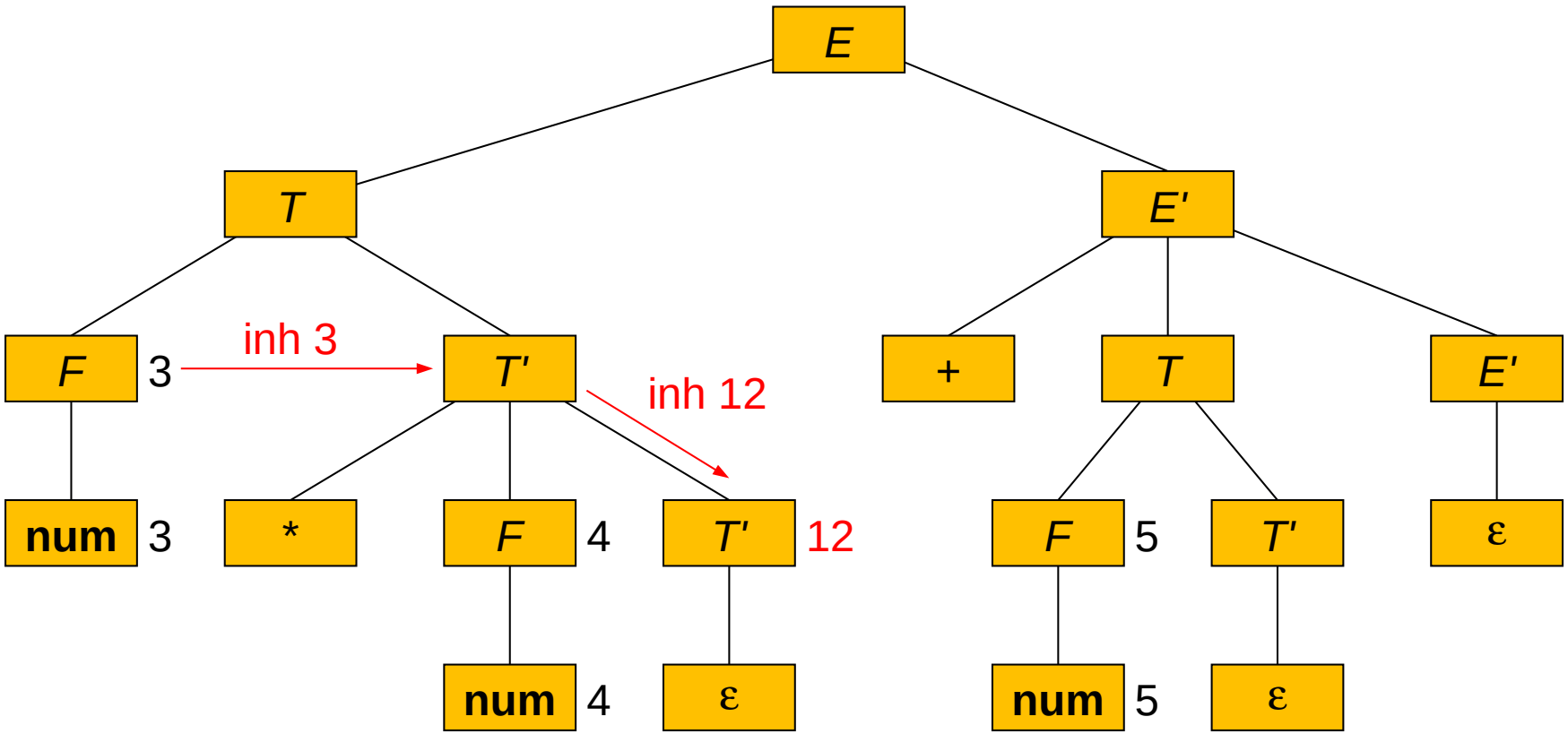
Example (Inherited Attributes)



$T'.\text{inh} = F.\text{val}$

Inherited Attributes

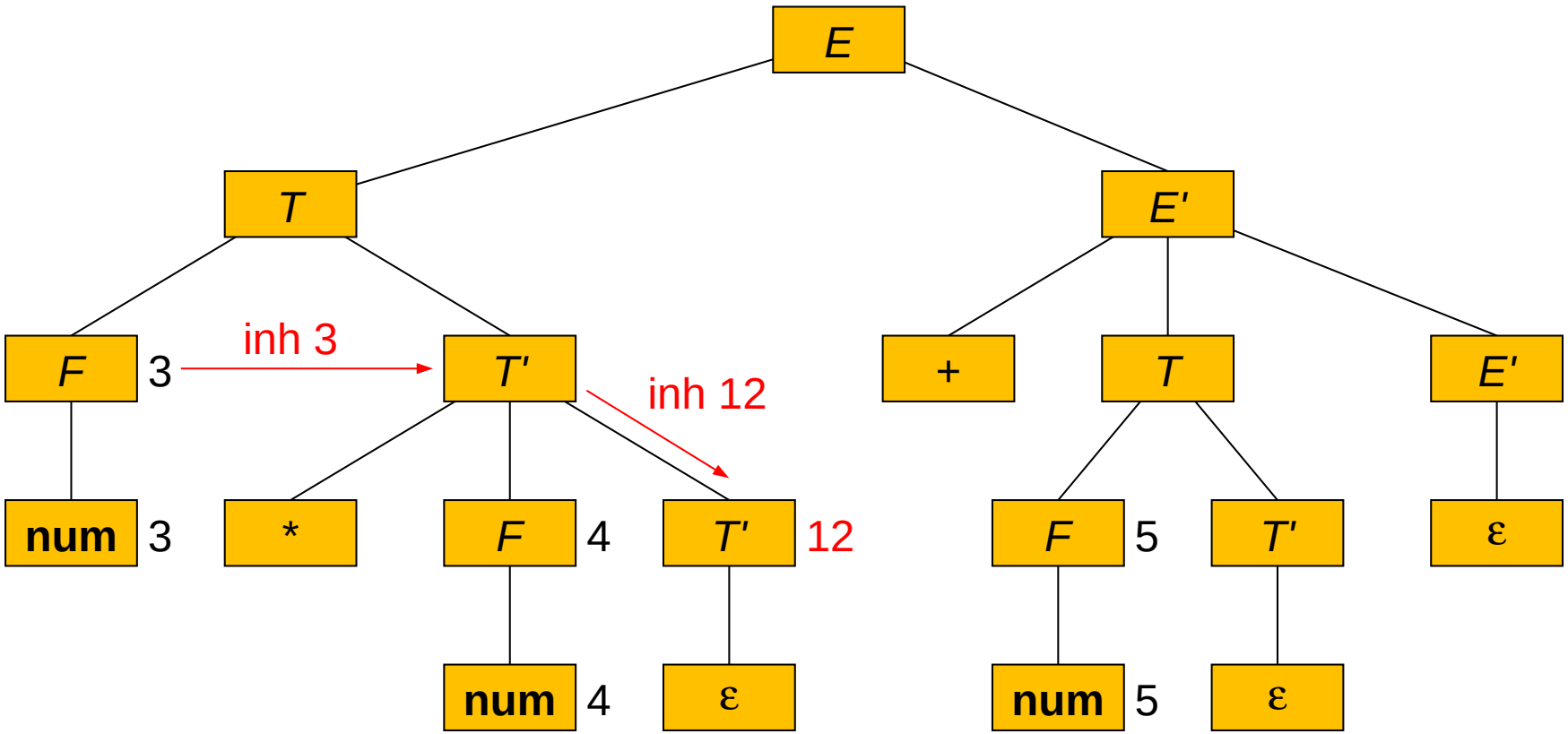
Example (Inherited Attributes)



$$T_1'.\text{inh} = T'.\text{inh} \times F.\text{val}$$

Inherited Attributes

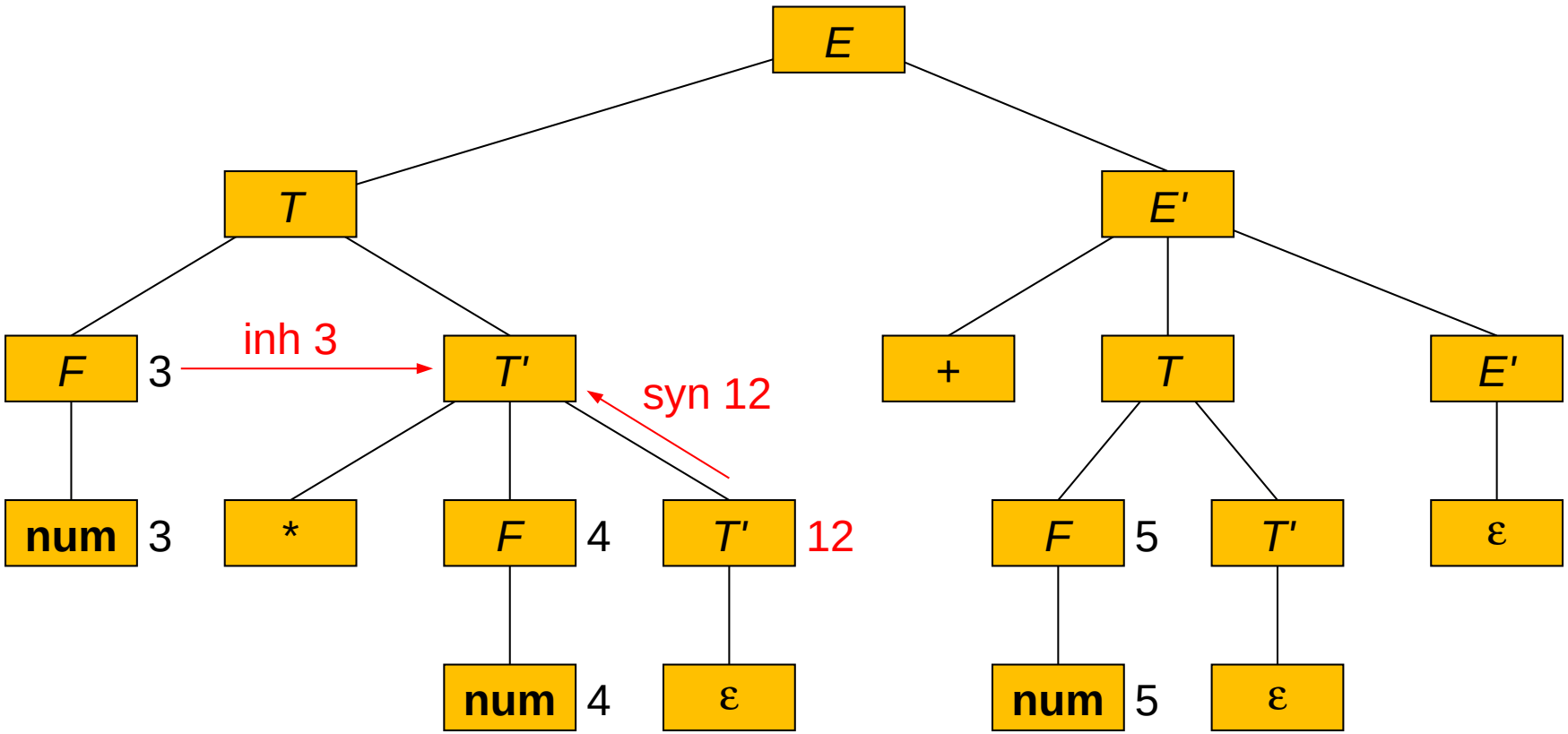
Example (Inherited Attributes)



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

Inherited Attributes

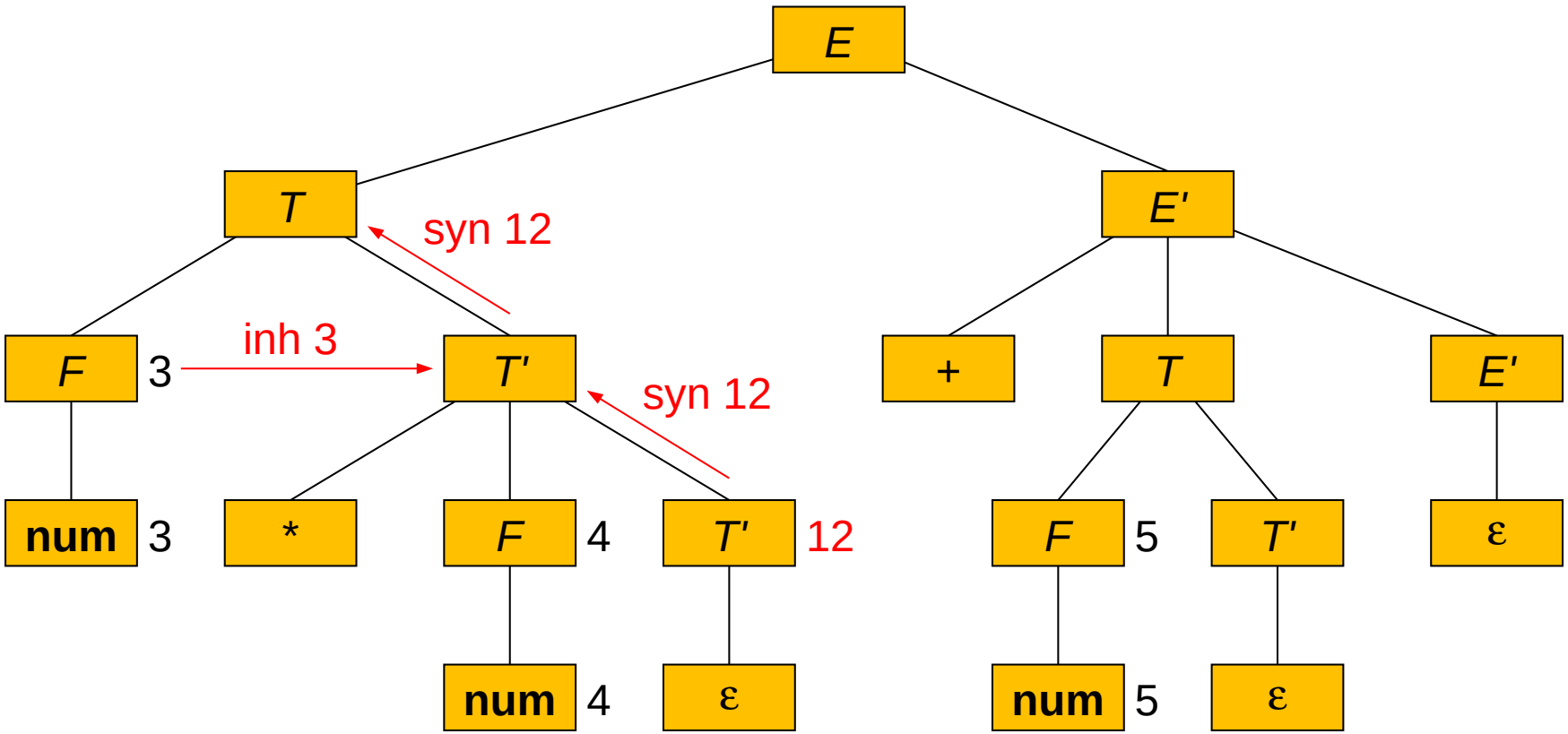
Example (Inherited Attributes)



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

Inherited Attributes

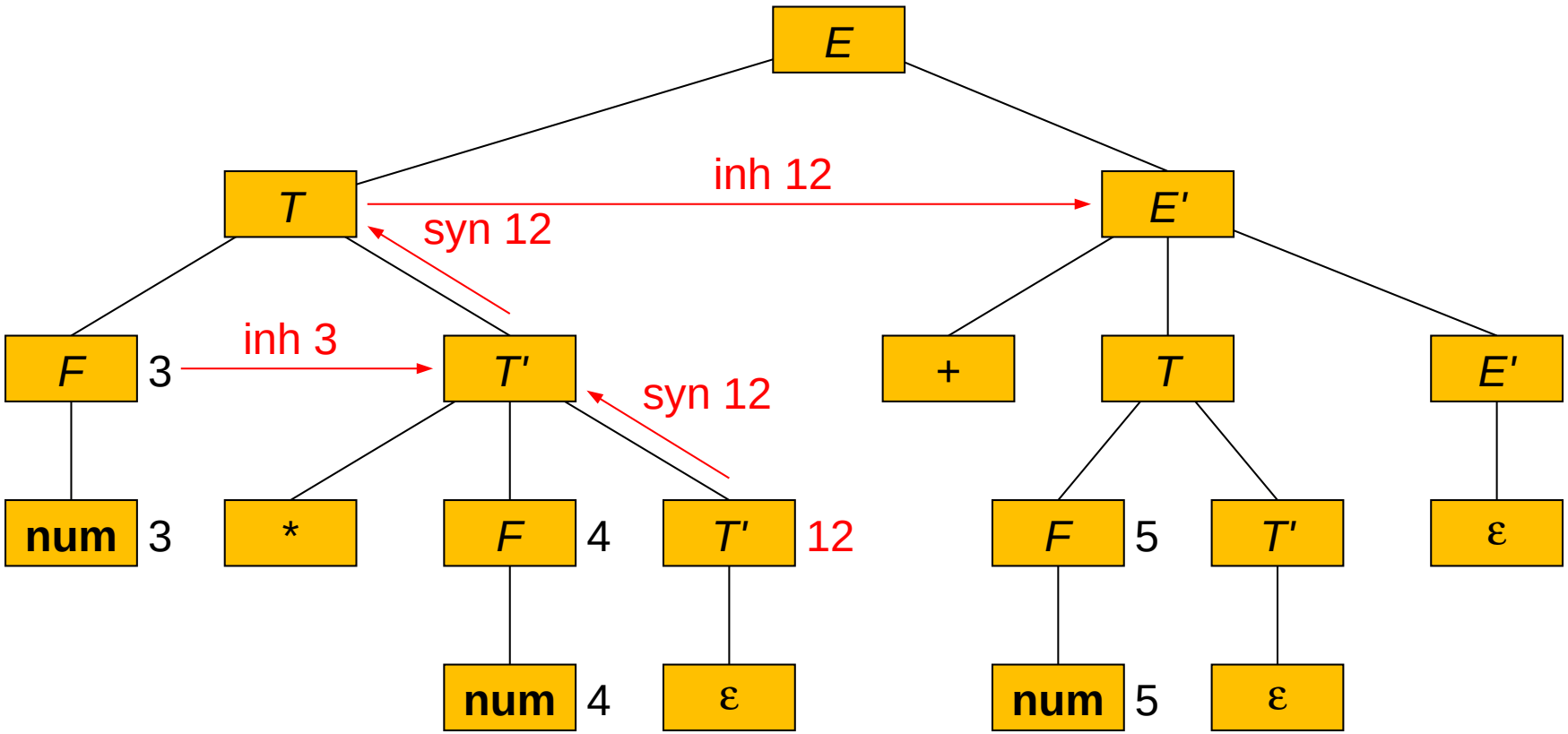
Example (Inherited Attributes)



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

Inherited Attributes

Example (Inherited Attributes)



$$E'.\text{inh} = T.\text{val}$$

Inherited Attributes

Example (Inherited Attributes)

- We now have the rules

Production	Semantic Rules
$T \to F\ T'$	$T'.\text{inh} = F.\text{val}$ $T.\text{val} = T'.\text{syn}$
$T' \to *\ F\ T_1'$	$T_1'.\text{inh} = T'.\text{inh} \times F.\text{val}$ $T'.\text{syn} = T_1'.\text{syn}$
$T' \to \varepsilon$	$T'.\text{syn} = T'.\text{inh}$
$F \to \textbf{num}$	$F.\text{val} = \textbf{num}.\text{lexval}$

Inherited Attributes

Example (Inherited Attributes)

- As well as the rules

Production	Semantic Rules
$E \to T\ E'$	$E'.\text{inh} = T.\text{val}$ $E.\text{val} = E'.\text{syn}$
$E' \to +\ T\ E'_1$	$E'_1.\text{inh} = E'.\text{inh} + T.\text{val}$ $E'.\text{syn} = E'_1.\text{syn}$
$E' \to \varepsilon$	$E'.\text{syn} = E'.\text{inh}$

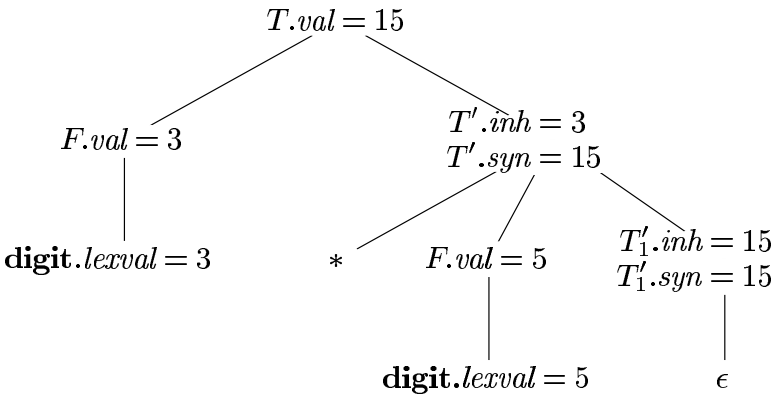
Outline


```
for(int i=0;i <=5; i++);
  {
   cout<<i;
  }
```

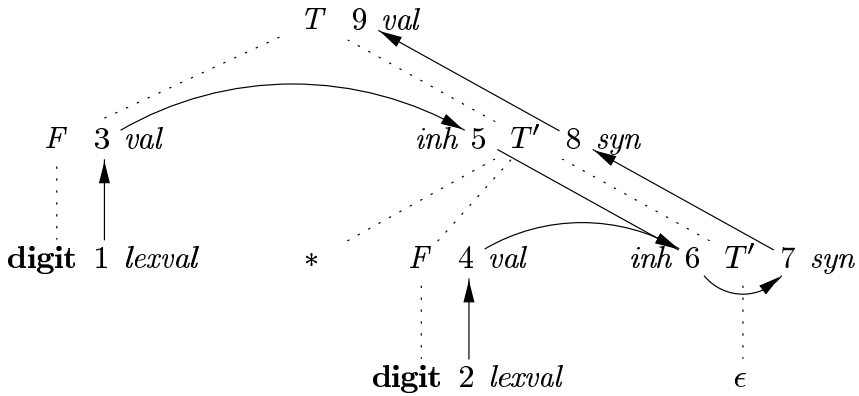
E = 4 + 5 E.val = E.val + T.val

Dependency Graph and Order of Evaluation:

$3 * 5$



Syntax



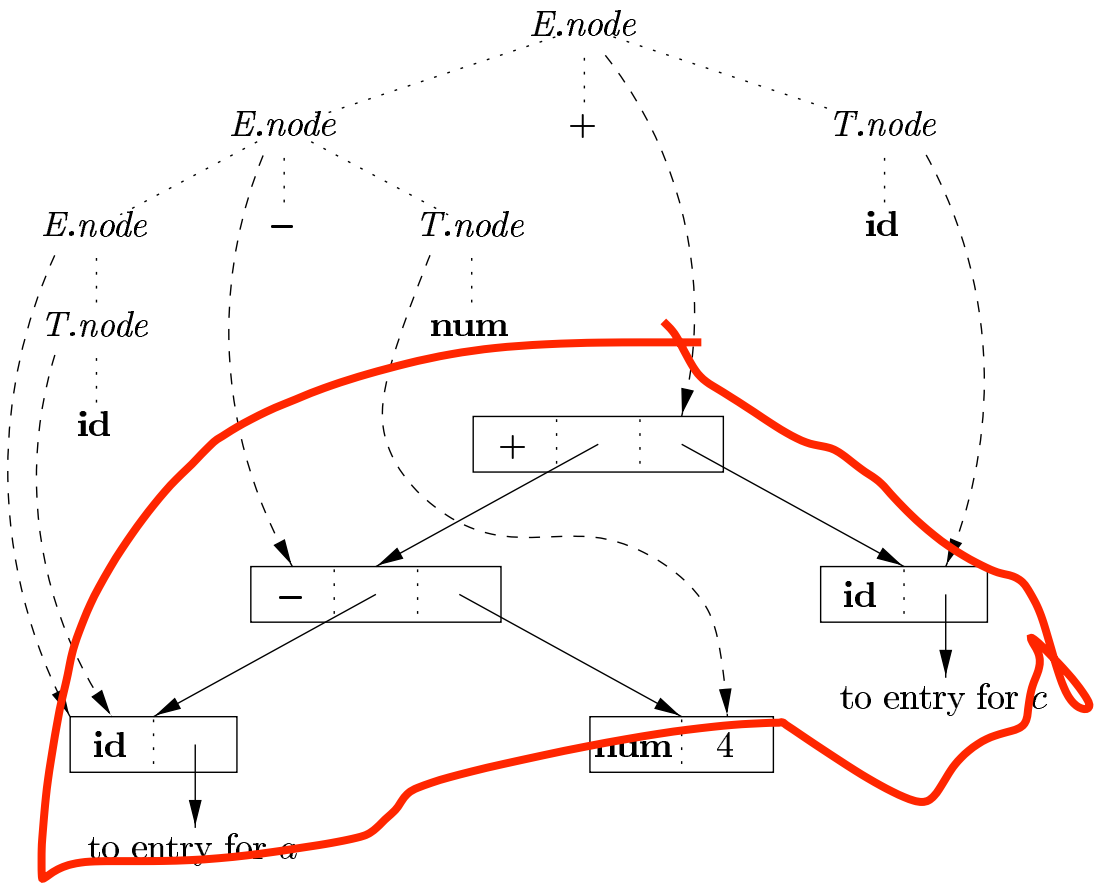
Semantic

Outline

Constructing syntax trees for simple expressions

	PRODUCTION	SEMANTIC RULES
1)	$E \to E_1 + T$	$E.node = \textbf{new}\ Node('+', E_1.node, T.node)$
2)	$E \to E_1 - T$	$E.node = \textbf{new}\ Node('-', E_1.node, T.node)$
3)	$E \to T$	$E.node = T.node$
4)	$T \to (\ E\ )$	$T.node = E.node$
5)	$T \to \textbf{id}$	$T.node = \textbf{new}\ Leaf(\textbf{id}, \textbf{id}.entry)$
6)	$T \to \textbf{num}$	$T.node = \textbf{new}\ Leaf(\textbf{num}, \textbf{num}.val)$

Syntax tree for $a - 4 + c$



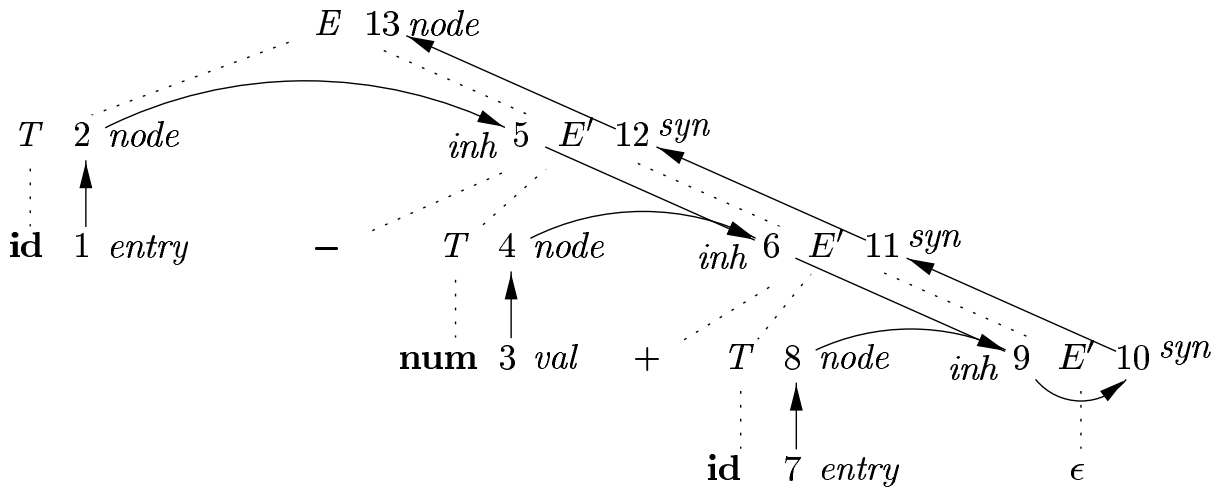
Steps for constructing a syntax tree for $a - 4 + c$

1) $p_1 = \textbf{new } Leaf(\textbf{id}, entry\text{-}a);$
2) $p_2 = \textbf{new } Leaf(\textbf{num}, 4);$
3) $p_3 = \textbf{new } Node('-', p_1, p_2);$
4) $p_4 = \textbf{new } Leaf(\textbf{id}, entry\text{-}c);$
5) $p_5 = \textbf{new } Node('+', p_3, p_4);$

Constructing syntax trees during top down parsing

	PRODUCTION	SEMANTIC RULES
1)	$E \rightarrow T\ E'$	$E.node = E'.syn$ $E'.inh = T.node$
2)	$E' \rightarrow +\ T\ E'_1$	$E'_1.inh = \textbf{new}\ Node('+', E'.inh, T.node)$ $E'.syn = E'_1.syn$
3)	$E' \rightarrow -\ T\ E'_1$	$E'_1.inh = \textbf{new}\ Node('-', E'.inh, T.node)$ $E'.syn = E'_1.syn$
4)	$E' \rightarrow \epsilon$	$E'.syn = E'.inh$
5)	$T \rightarrow (\ E\ )$	$T.node = E.node$
6)	$T \rightarrow \textbf{id}$	$T.node = \textbf{new}\ Leaf(\textbf{id}, \textbf{id}.entry)$
7)	$T \rightarrow \textbf{num}$	$T.node = \textbf{new}\ Leaf(\textbf{num}, \textbf{num}.val)$

Dependency graph for $a - 4 + c$ in top-down parsing



Outline

Assignment

Homework

- p. 309: 1, 2, 3