Abstract Syntax Trees

Lecture 14
Sections 5.1 - 5.4

Robb T. Koether

Hampden-Sydney College

Mon, Mar 2, 2009

Outline

1 Abstract Syntax Trees

2 Synthesized Attributes

3 Inherited Attributes

4 Expression Trees

5 Assignment

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.

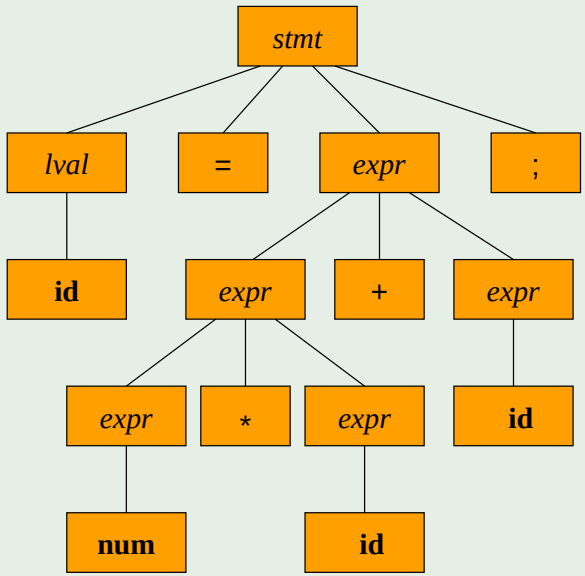
Abstract Syntax Trees

- 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.

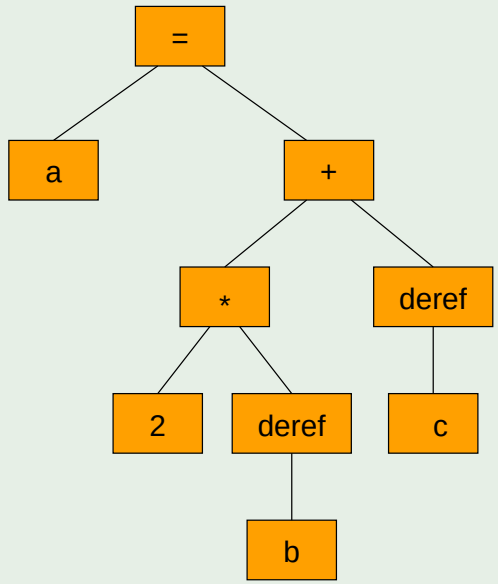
Syntax Trees vs. Parse Trees

Example (Syntax Trees and Parse Trees)

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



Parse Tree



Syntax Tree

Syntax Trees vs. Parse Trees

- Our `TreeBuilder` program will "convert" the parse tree into the syntax tree.
- The parse tree never really exists, except insofar as the parser follows its logical order.
- The `TreeBuilder` will simply build the syntax tree from the information obtained by the parser.
- Then the code generator will write the assembly code from the syntax tree.

Abstract Syntax Trees

- Recursive descent parsers generally create a single AST for the entire program.
- Our parser will generate a separate AST for each statement.
  - It will create a list of ASTs.
  - This will allow us to generate assembly code as the ASTs are created.
  - The trees will be connected both sequentially and through jump statements.

Syntax-Directed Definitions

Definition

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

- These attributes are stored in the nodes of the syntax tree.
- Each node has
  - A set of synthesized attributes, and
  - A set of inherited attributes.

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 $A \to \alpha$ is a production, then A's synthesized attributes are determined by the attributes of the symbols in α.

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 $A \to \alpha\beta\gamma$, then β's inherited attributes are determined by the attributes of A and the other symbols in α and γ.

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)

- Let the grammar be

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

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

$$\begin{aligned} E.\text{val} &= E_1.\text{val} + E_2.\text{val} \\ E.\text{val} &= \mathbf{num}.\text{lexval} \end{aligned}$$

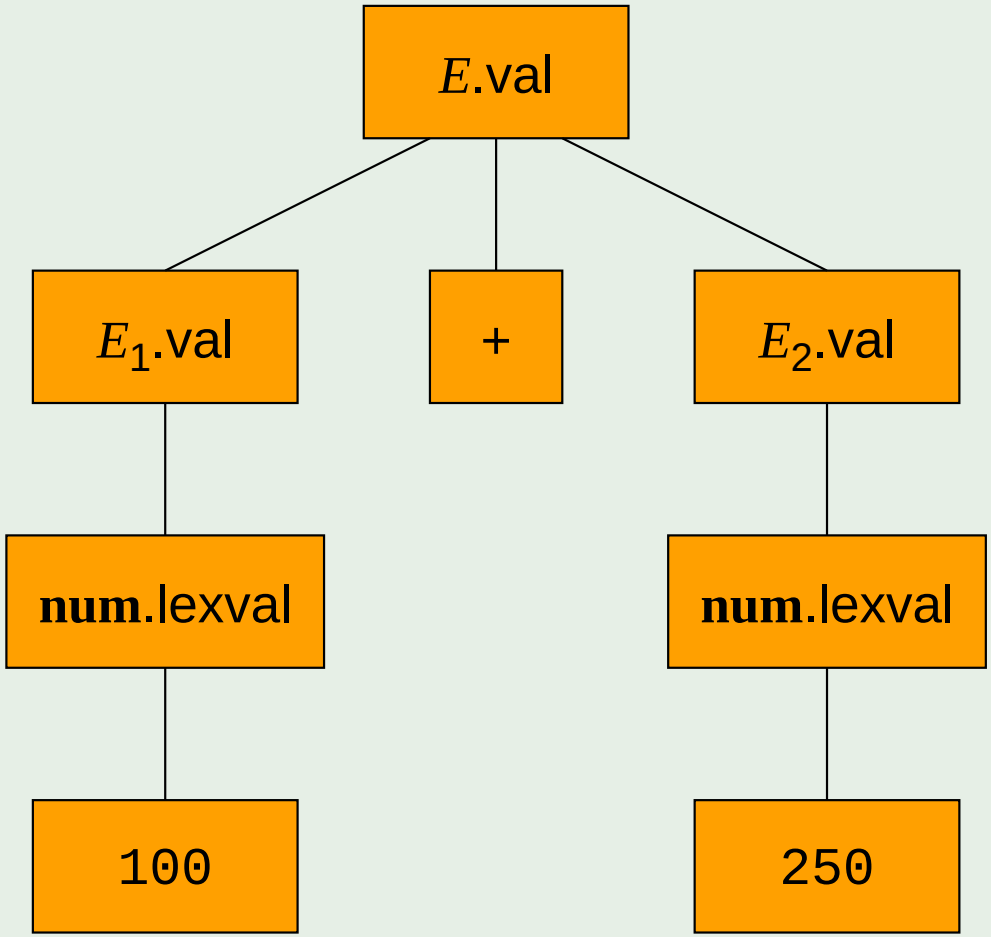
Example

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.

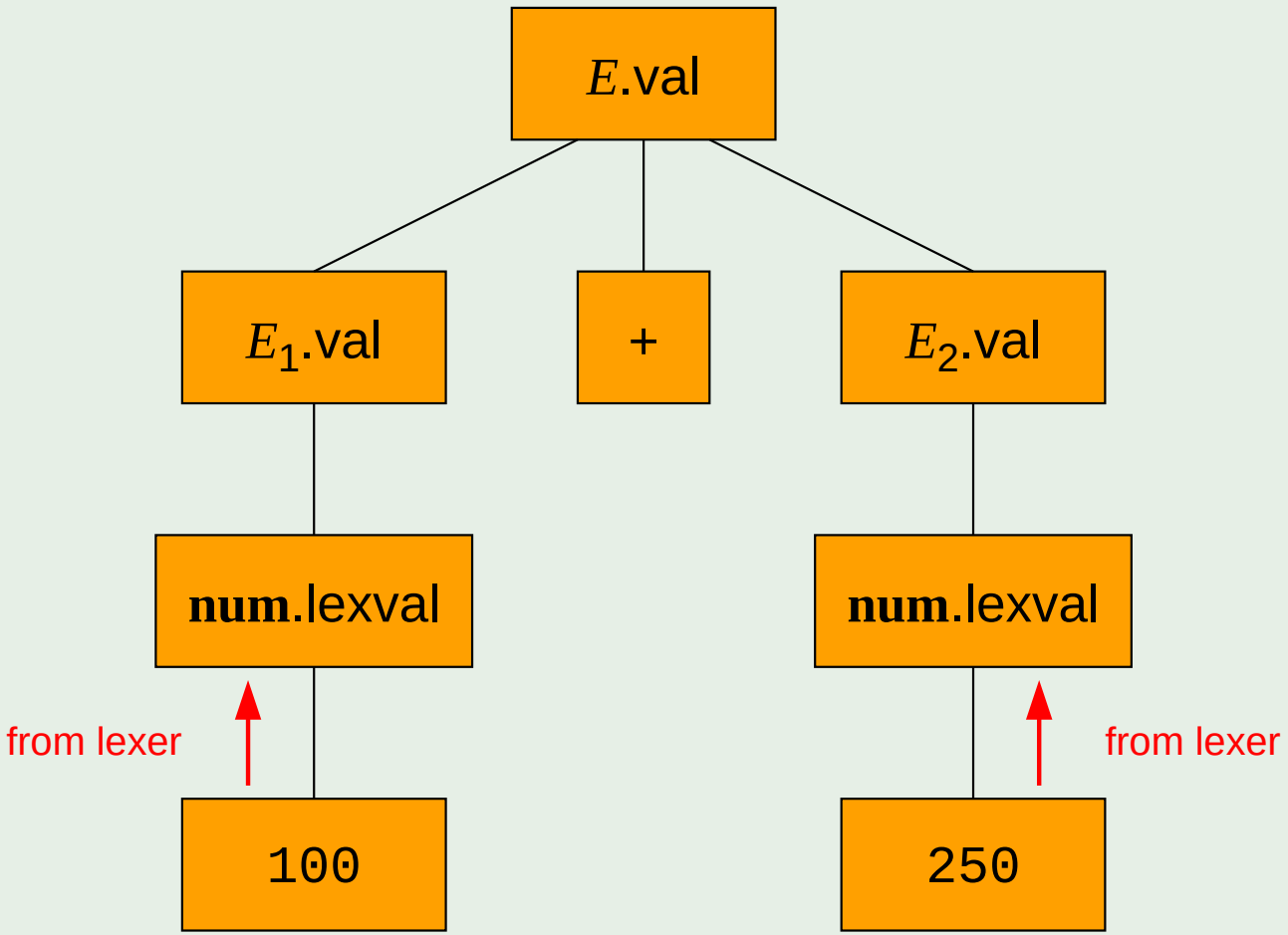
Example

Example (Synthesized Attributes)



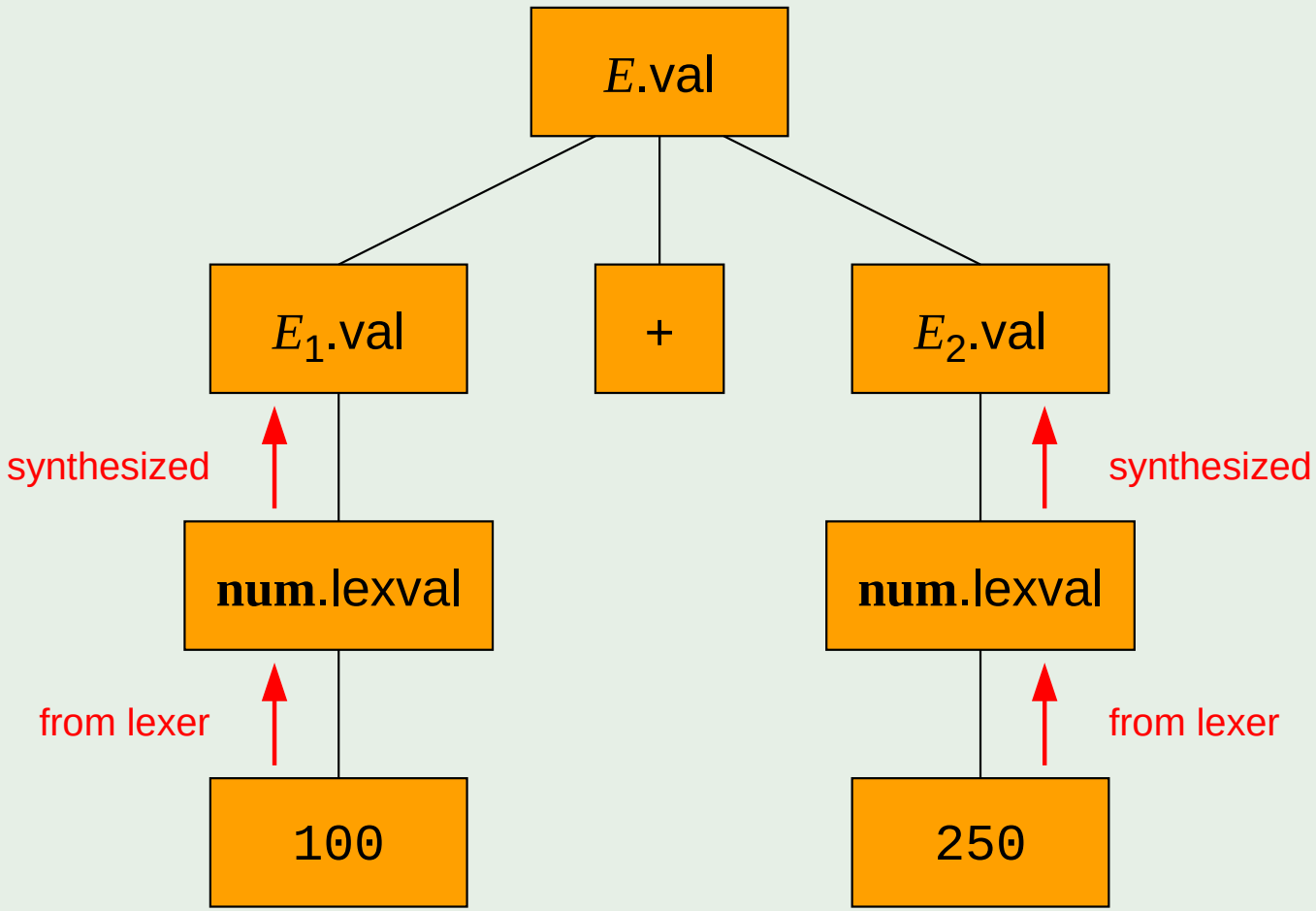
Example

Example (Synthesized Attributes)



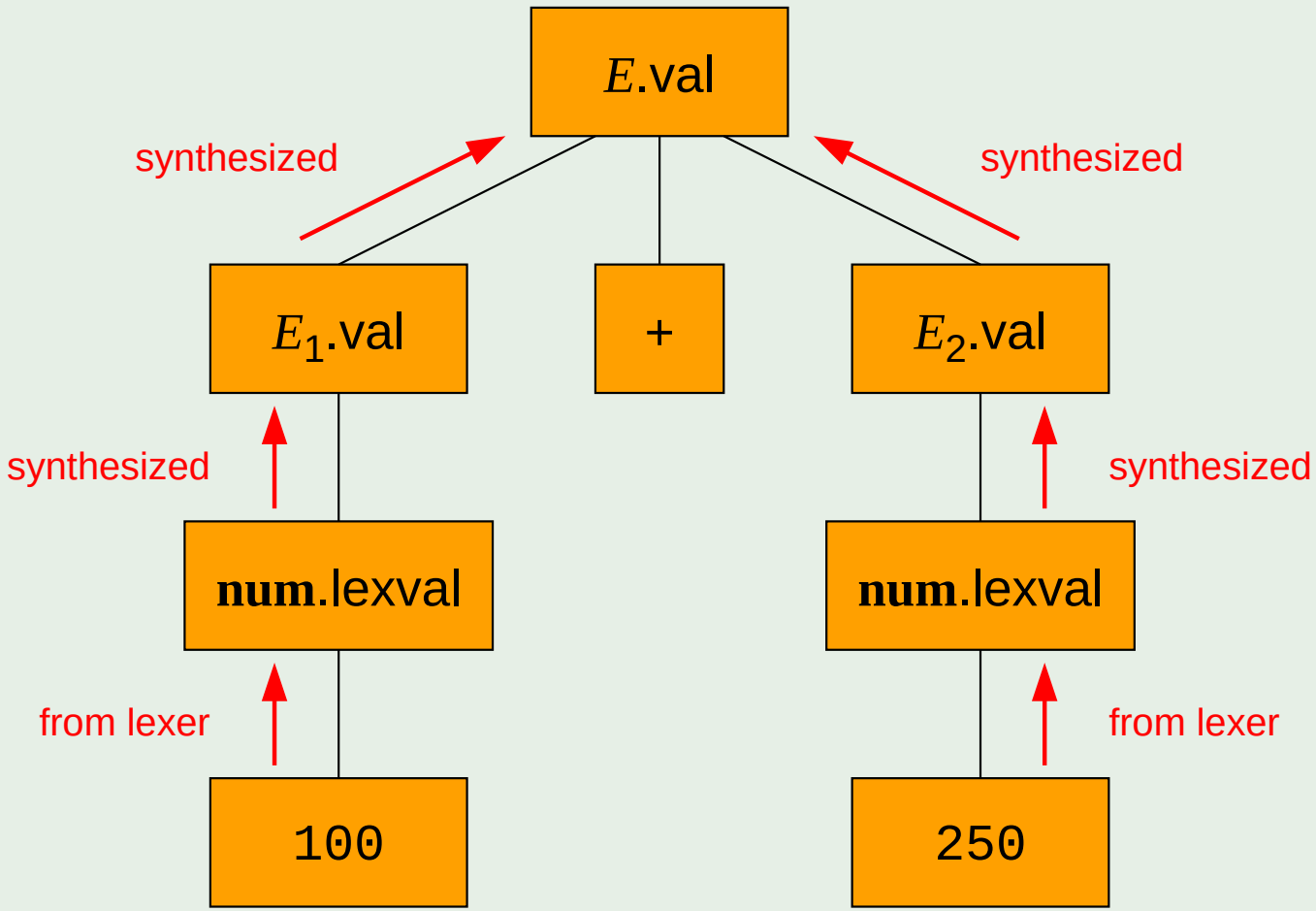
Example

Example (Synthesized Attributes)



Example

Example (Synthesized Attributes)



Example

Example (Inherited Attributes)

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

$$\begin{aligned} D &\rightarrow T\ L \\ L &\rightarrow L\ \mathbf{,}\ \mathbf{id} \mid \mathbf{id} \\ T &\rightarrow \mathbf{int} \mid \mathbf{float} \end{aligned}$$

- For example, the declaration might be

```
float a, b;
```

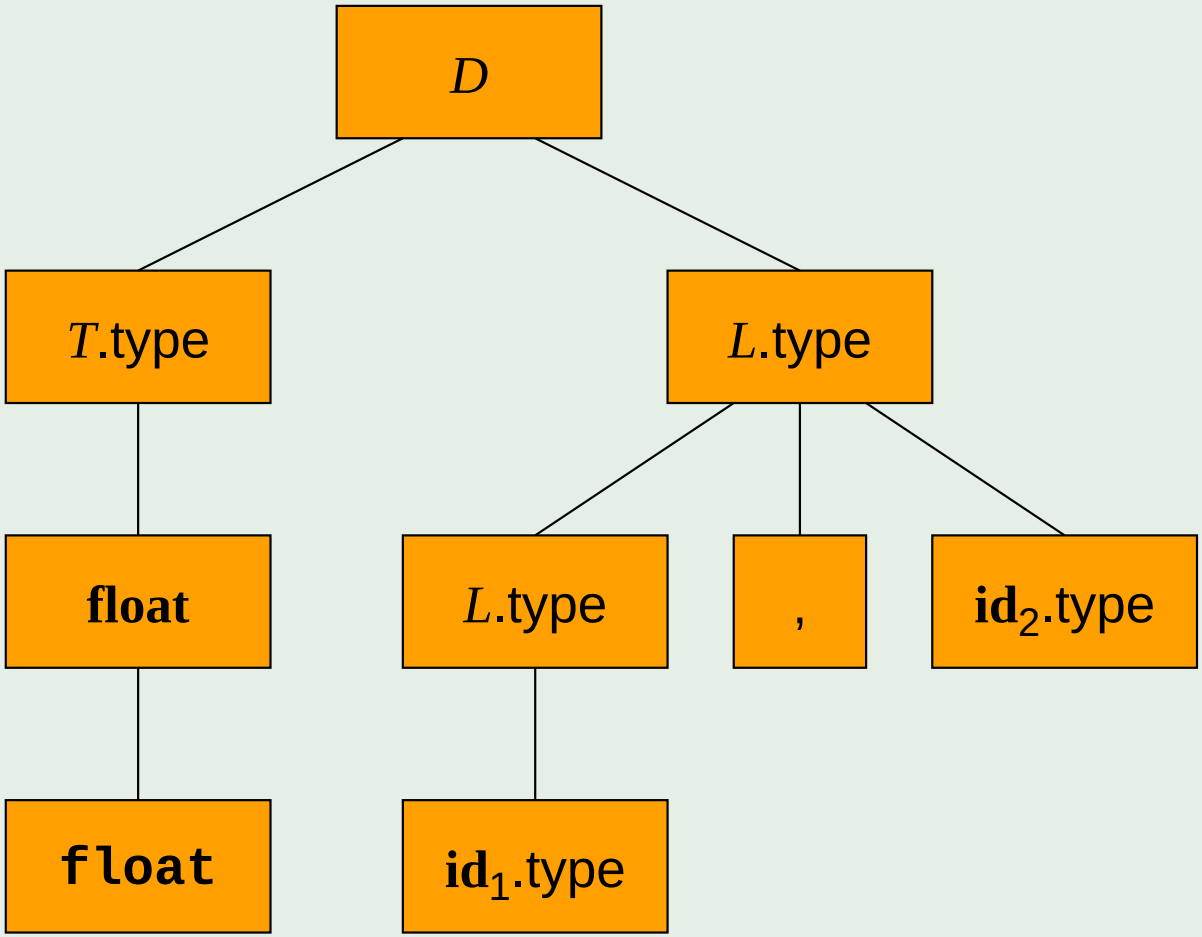
Example

Example (Inherited Attributes)

- The attribute **`float`** first appears as the type of the `float` token.
- From there it is passed to the identifiers `a` and `b`.

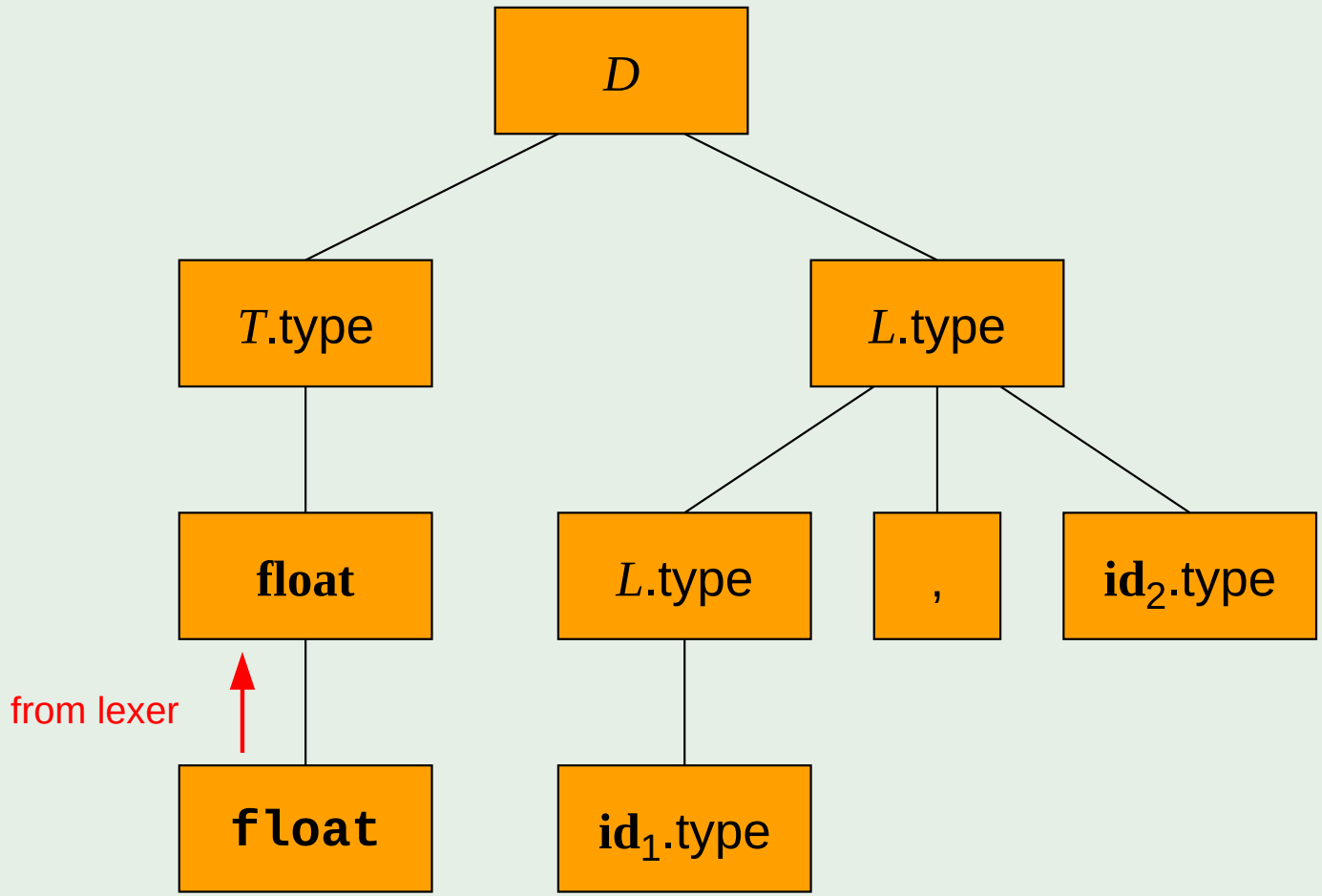
Example

Example (Inherited Attributes)



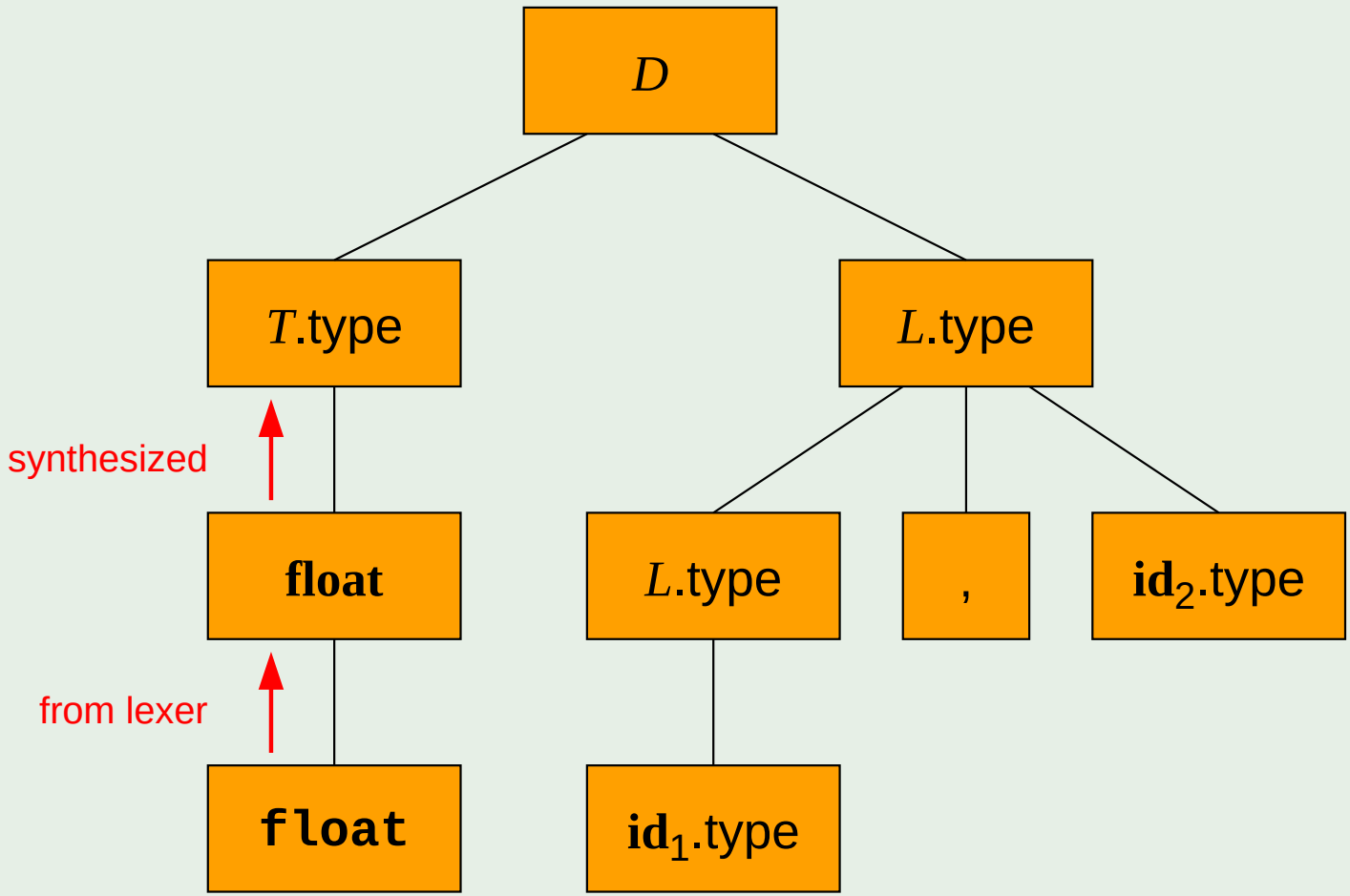
Example

Example (Inherited Attributes)



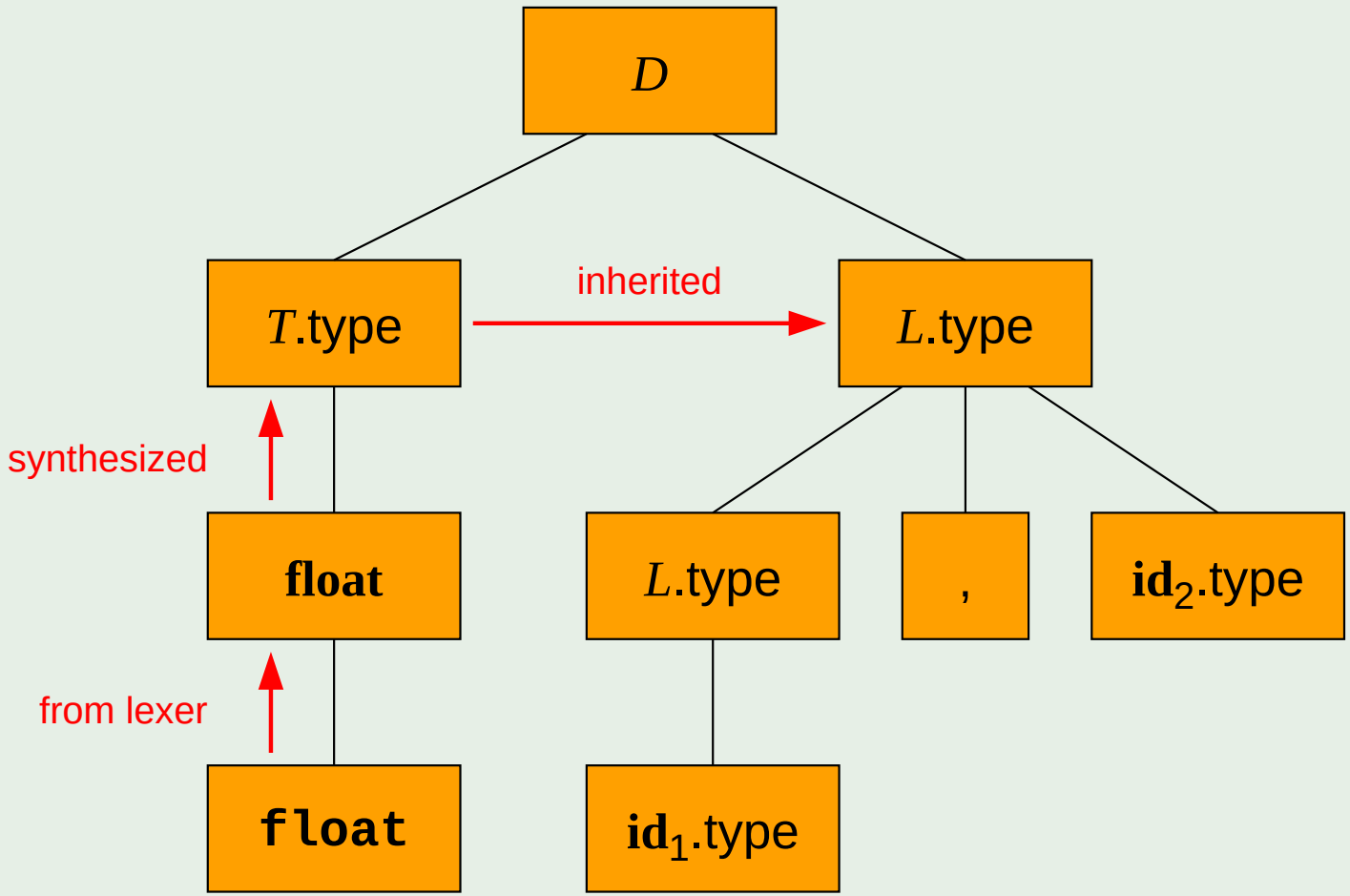
Example

Example (Inherited Attributes)



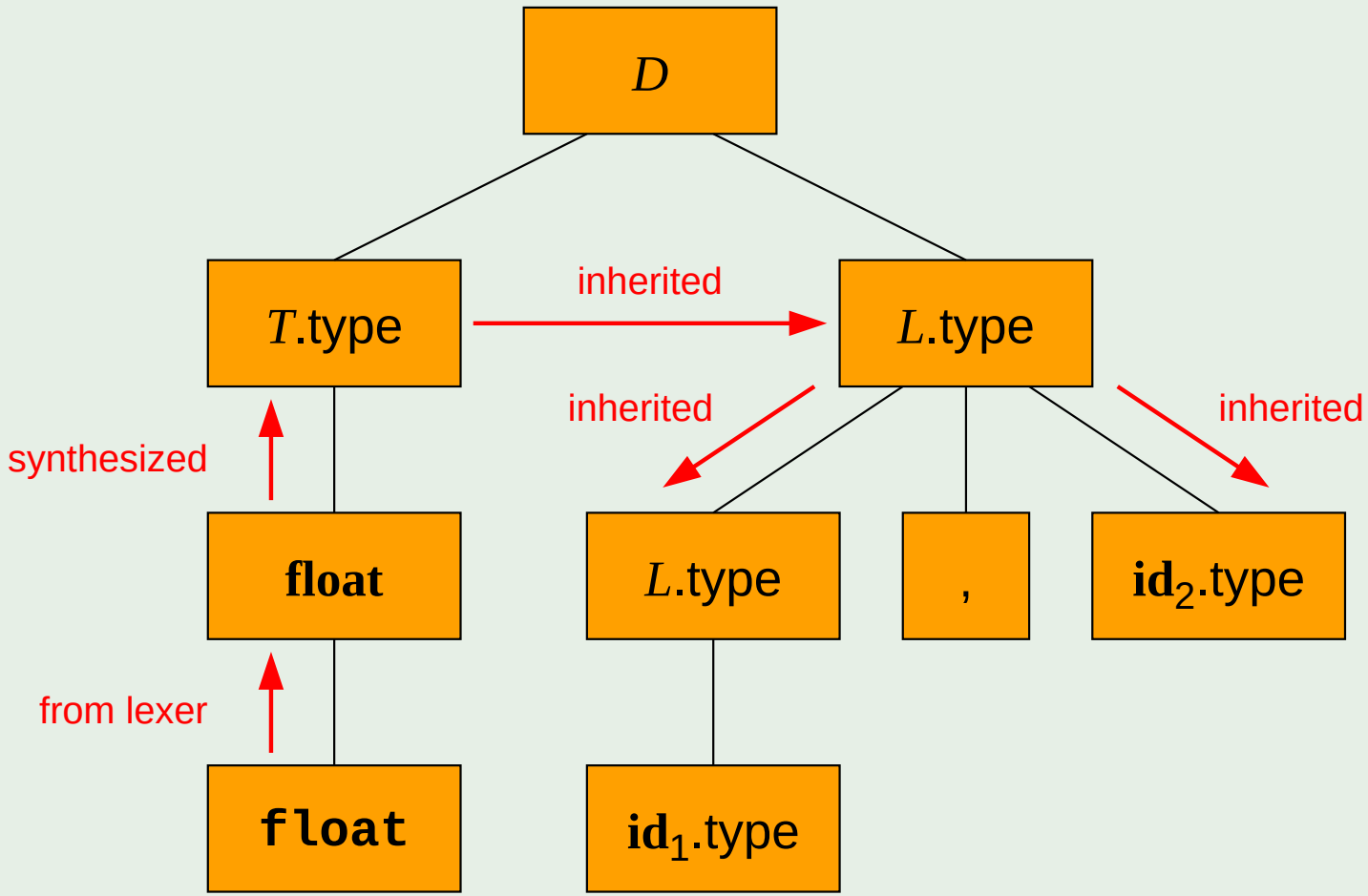
Example

Example (Inherited Attributes)



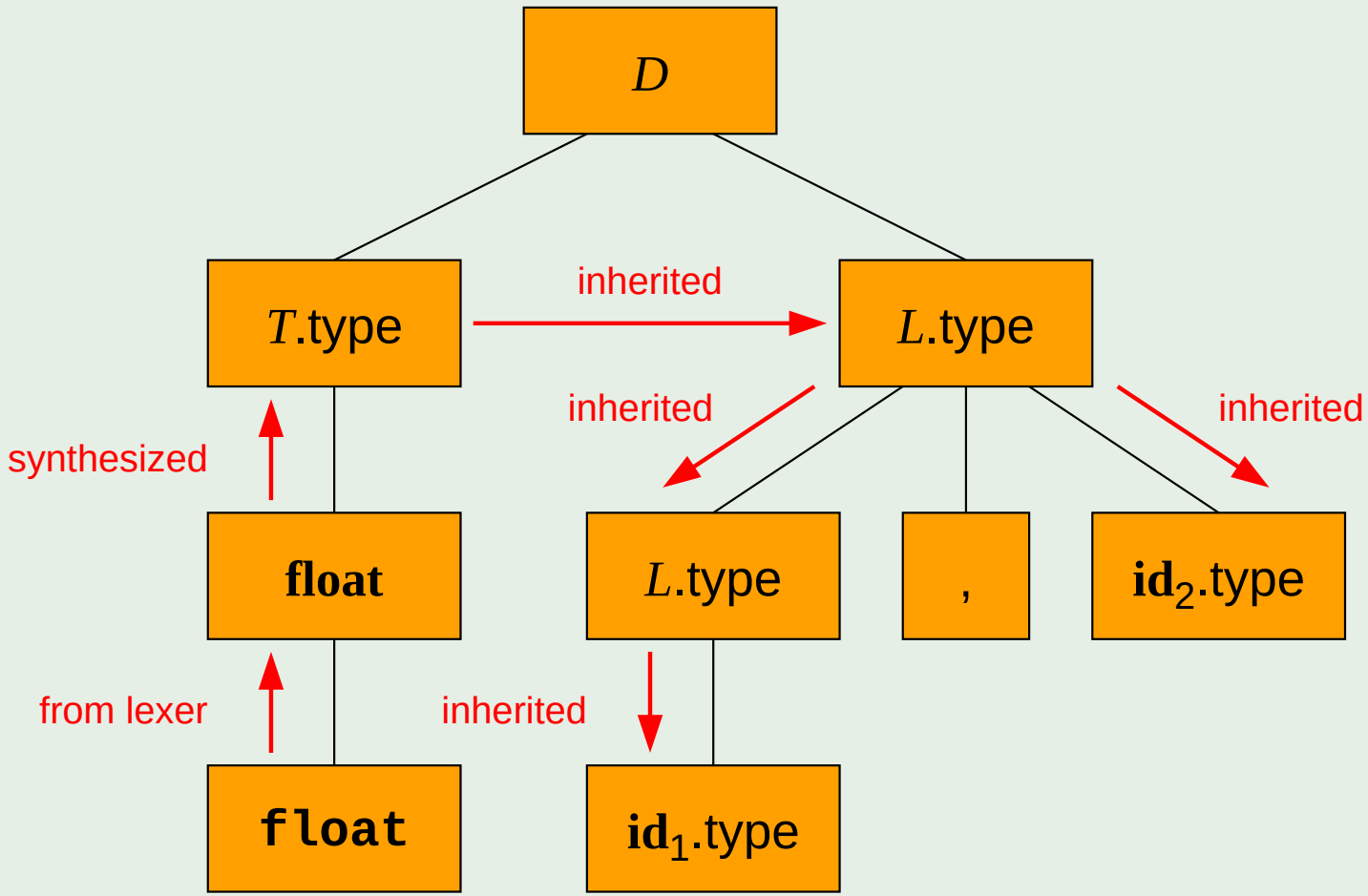
Example

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?

Example

- We will describe how to build an AST for an expression.
- We will use `TreeNode` constructors similar to the following.
  - `TreeNode(`*op*`,` *left*`,` *right*`)`
    - Join two existing trees, placing *op* at the root node.
  - `TreeNode(`*id*`,` *entry*`)`
    - Create a single-node tree with *id* at the root node.
  - `TreeNode(`*num*`,` *value*`)`
    - Create a single-node tree with *num* at the root node.

Example

Example (Expression Tree)

- To construct a tree for the expression

```
a - 4 + c
```

we do the following:

- $\text{tree}_1 =$ **new** `TreeNode(`**id**`,` $\text{idEntry}_{\texttt{a}}$`)`
- $\text{tree}_2 =$ **new** `TreeNode(`**num**`, 4)`
- $\text{tree}_3 =$ **new** `TreeNode(`**minus**`,` tree_1`,` tree_2`)`
- $\text{tree}_4 =$ **new** `TreeNode(`**id**`,` $\text{idEntry}_{\texttt{c}}$`)`
- $\text{tree}_5 =$ **new** `TreeNode(`**plus**`,` tree_3`,` tree_4`)`

Example

Example (Expression Tree)

Production	Semantic Rule
$E \to E_1 + E_2$	E.tree = **new** `TreeNode(`**plus**`,` E_1.tree`,` E_2.tree`);`
$E \to E_1 - E_2$	E.tree = **new** `TreeNode(`**minus**`,` E_1.tree`,` E_2.tree`);`
$E \to (E_1)$	E.tree = **new** `TreeNode(`E_1.tree`);`
$E \to$ **id**	E.tree = **new** `TreeNode(`**id**`,` **id**.entry`);`
$E \to$ **num**	E.tree = **new** `TreeNode(`**num**`,` **num**.val`);`

Assignment

Homework

- Read Sections 5.1 - 5.4, pages 279 - 301.