Parsing:
Derivations, Ambiguity, Precedence, Associativity

Lecture 8

Topics covered so far

- Regular languages and Finite automaton
- Parser overview
- Context-free grammars (CFG’s)
- Derivations
- First, follow, predict sets

Outline of Today's Lecture

- Ambiguity in context-free grammars
- Left and right derivations
- Associativity
- Precedence
- Bottom-up Parsing

CFGs (Cont.)

- A CFG consists of
  - A set of *terminals* T
  - A set of *non-terminals* N
  - A *start symbol* S (a non-terminal)
  - A set of *productions*

$$X \rightarrow Y_1 Y_2 \ldots Y_n$$

$$\text{where } X \in N \text{ and } Y_i \in T \cup N \cup \{\varepsilon\}$$

Examples of CFGs (cont.)

Simple arithmetic expressions:

$$
\begin{aligned}
E \quad &\rightarrow \quad E * E \\
&\mid \quad E + E \\
&\mid \quad (E) \\
&\mid \quad \mathrm{id}
\end{aligned}
$$

Arithmetic Example

Simple arithmetic expressions:

$$E \rightarrow E+E \mid E * E \mid (E) \mid \text{id}$$

Some elements of the language:

id	$\text{id} + \text{id}$
(id)	$\text{id} * \text{id}$
$(\text{id}) * \text{id}$	$\text{id} * (\text{id})$

Derivation Example

- Grammar

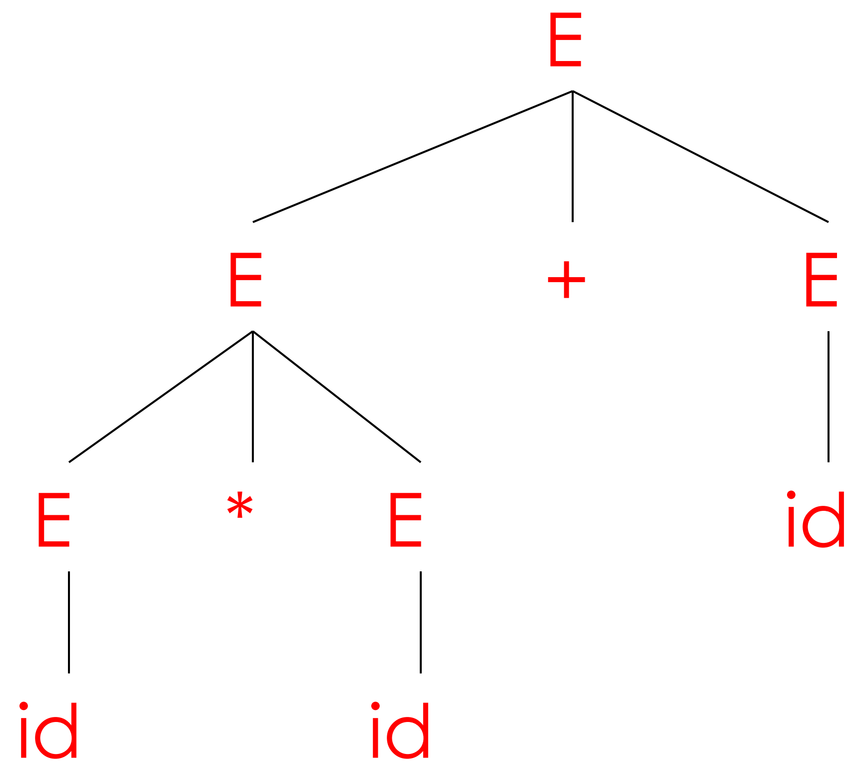
$$E \rightarrow E+E \mid E * E \mid (E) \mid id$$

- String

$$id * id + id$$

Derivation Example (Cont.)

$$
\begin{aligned}
& E \\
\rightarrow\ & E+E \\
\rightarrow\ & E * E+E \\
\rightarrow\ & id * E + E \\
\rightarrow\ & id * id + E \\
\rightarrow\ & id * id + id
\end{aligned}
$$

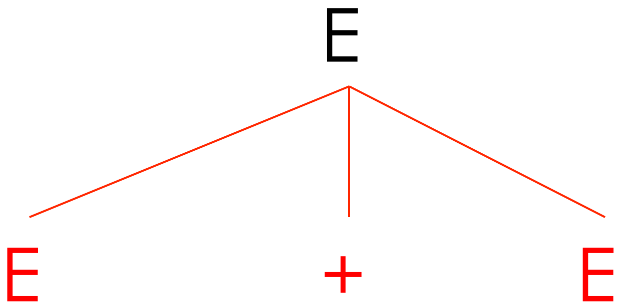


Derivation in Detail (1)

E

E

Derivation in Detail (2)



E

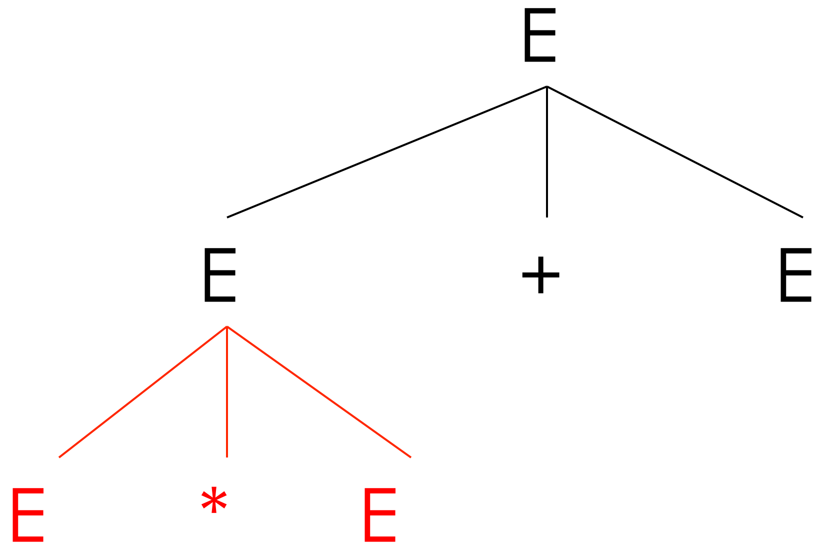
$\rightarrow$ E+E

Derivation in Detail (3)

$$E$$

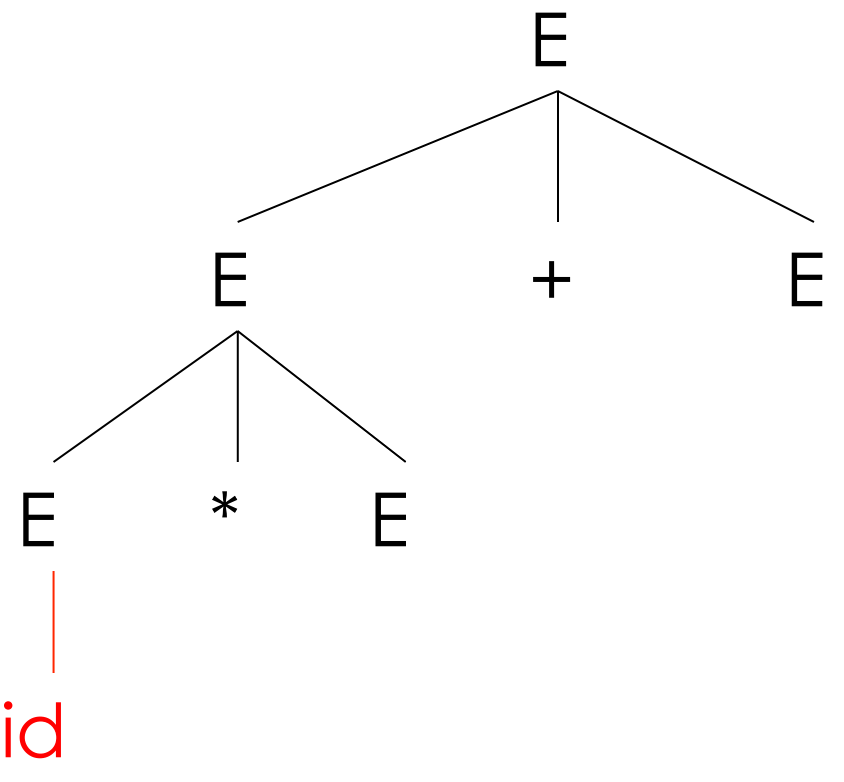
$$\rightarrow \quad E+E$$

$$\rightarrow \quad E * E+E$$



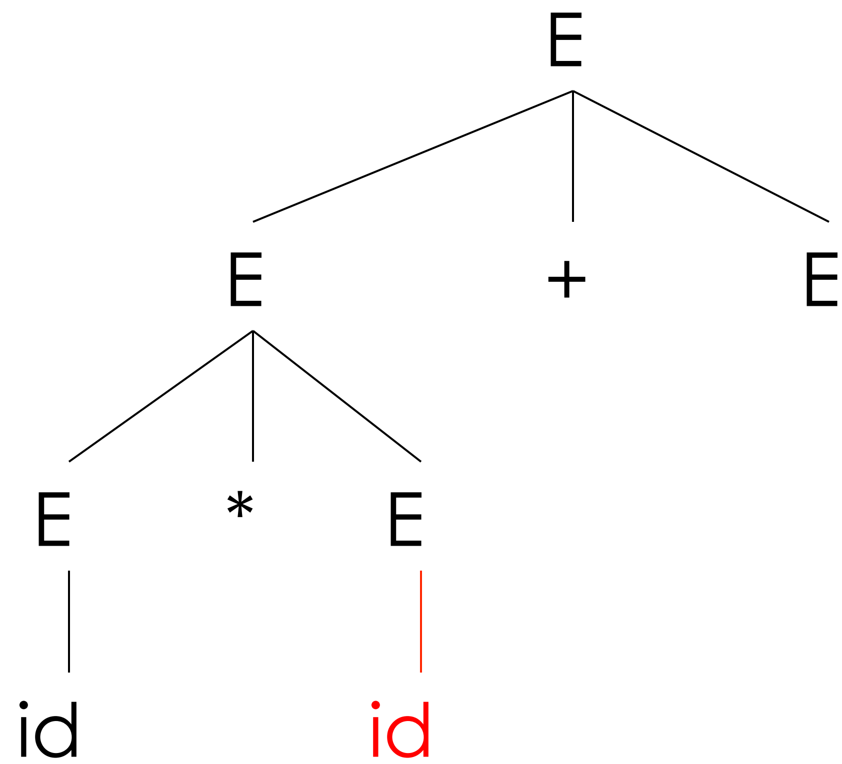
Derivation in Detail (4)

$$\begin{aligned} & E \\ \rightarrow\ & E+E \\ \rightarrow\ & E * E+E \\ \rightarrow\ & \mathrm{id} * E + E \end{aligned}$$



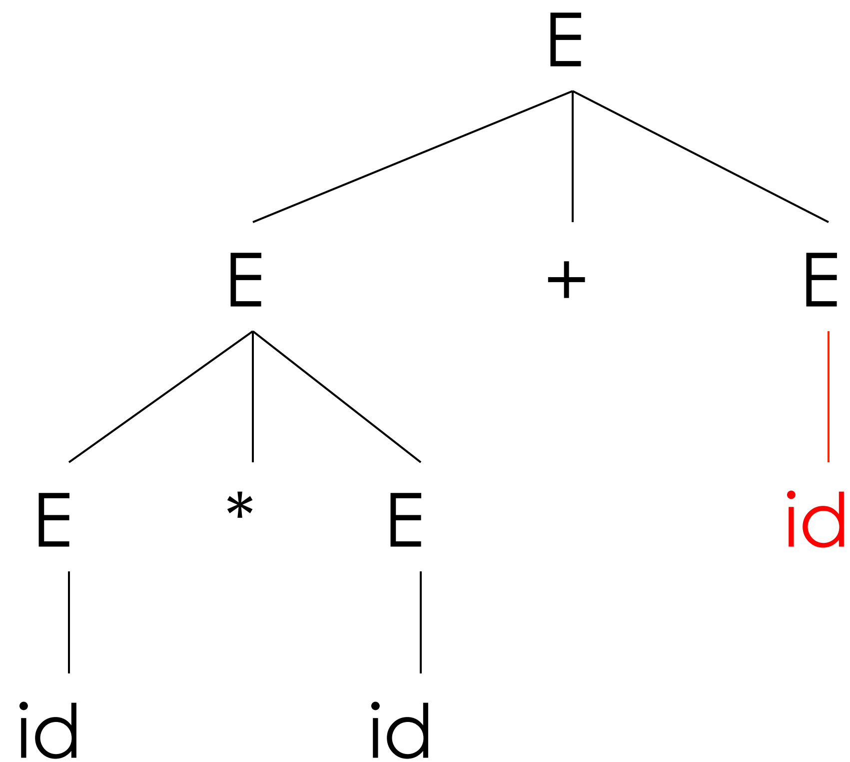
Derivation in Detail (5)

$$
\begin{aligned}
& E \\
\rightarrow \quad & E+E \\
\rightarrow \quad & E * E+E \\
\rightarrow \quad & id * E + E \\
\rightarrow \quad & id * id + E
\end{aligned}
$$



Derivation in Detail (6)

$$
\begin{aligned}
& E \\
\rightarrow\ & E+E \\
\rightarrow\ & E * E+E \\
\rightarrow\ & id * E + E \\
\rightarrow\ & id * id + E \\
\rightarrow\ & id * id + id
\end{aligned}
$$



Notes on Derivations

- A parse tree has
  - Terminals at the leaves
  - Non-terminals at the interior nodes

- An in-order traversal of the leaves is the original input

- The parse tree shows the association of operations, the input string does not

Left-most and Right-most Derivations

- The example is a *left-most* derivation
  - At each step, replace the left-most non-terminal

- There is an equivalent notion of a *right-most* derivation

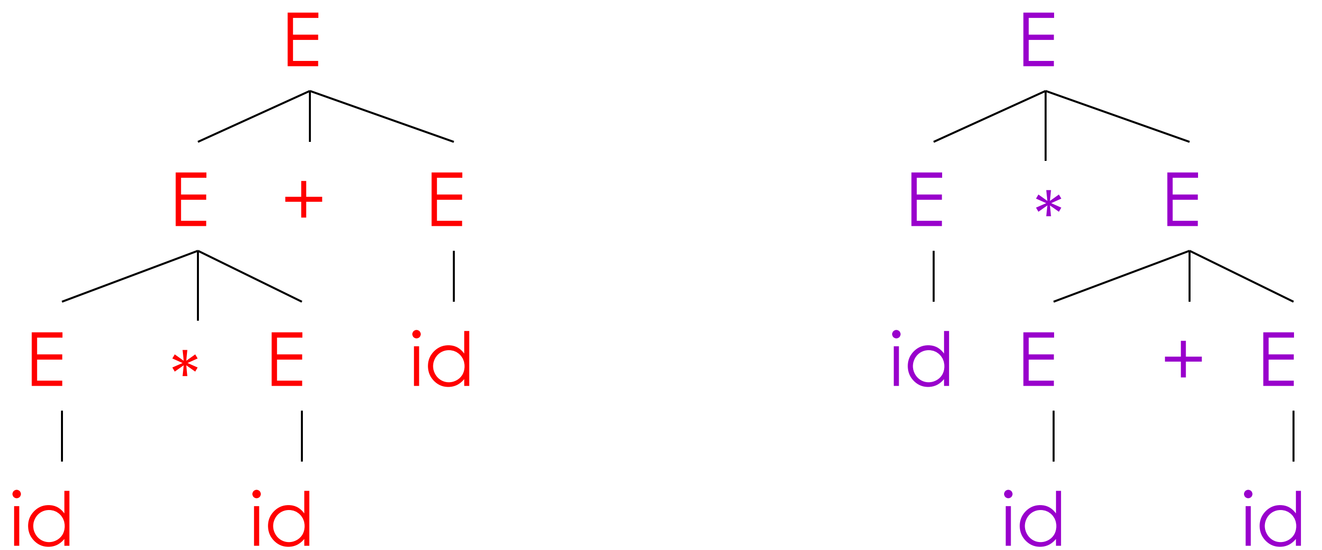
$$
\begin{aligned}
& E \\
\rightarrow\; & E+E \\
\rightarrow\; & E+id \\
\rightarrow\; & E * E + id \\
\rightarrow\; & E * id + id \\
\rightarrow\; & id * id + id
\end{aligned}
$$

Ambiguity

- Grammar $E \rightarrow E+E \mid E * E \mid (E) \mid id$
- String $id * id + id$

Ambiguity (Cont.)

This string has two parse trees



Ambiguity (Cont.)

- A grammar is *ambiguous* if it has more than one parse tree for some string
  - Equivalently, there is more than one right-most or left-most derivation for some string

- Ambiguity is **BAD**
  - Leaves meaning of some programs ill-defined

Dealing with Ambiguity

- There are several ways to handle ambiguity
- Most direct method is to rewrite grammar unambiguously

$$E \quad \rightarrow \quad E' + E \mid E'$$

$$E' \quad \rightarrow \quad id * E' \mid id \mid (E) * E' \mid (E)$$

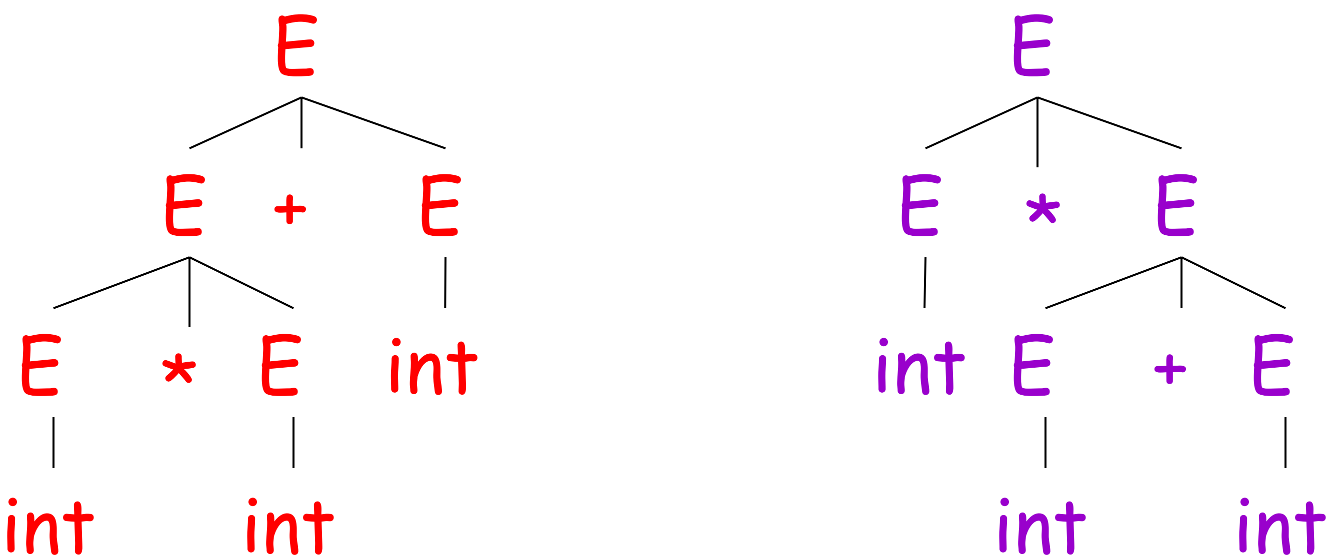
- Enforces precedence of * over +

Ambiguity in Arithmetic Expressions

- Recall the grammar

  E → E + E | E * E | (E) | int

- The string int * int + int has two parse trees:



Ambiguity: The Dangling Else

- Consider the grammar

  $E \rightarrow$ if E then E

  | if E then E else E

  | OTHER

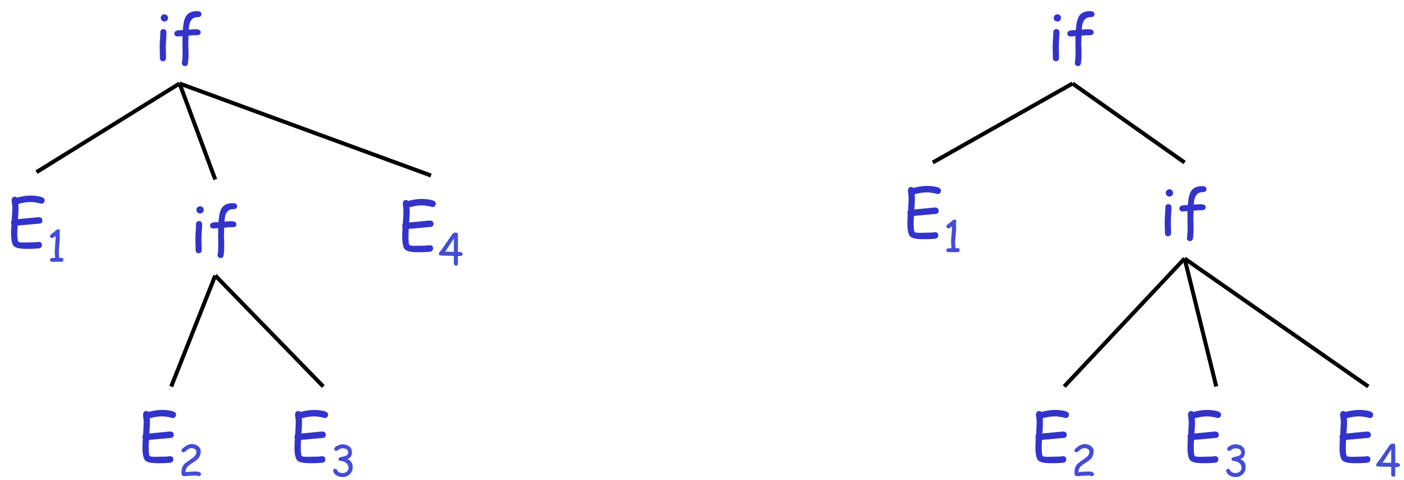
- This grammar is also ambiguous

The Dangling Else: Example

- The expression

  $\text{if } E_1 \text{ then if } E_2 \text{ then } E_3 \text{ else } E_4$

  has two parse trees



- Typically we want the second form

The Dangling Else: A Fix

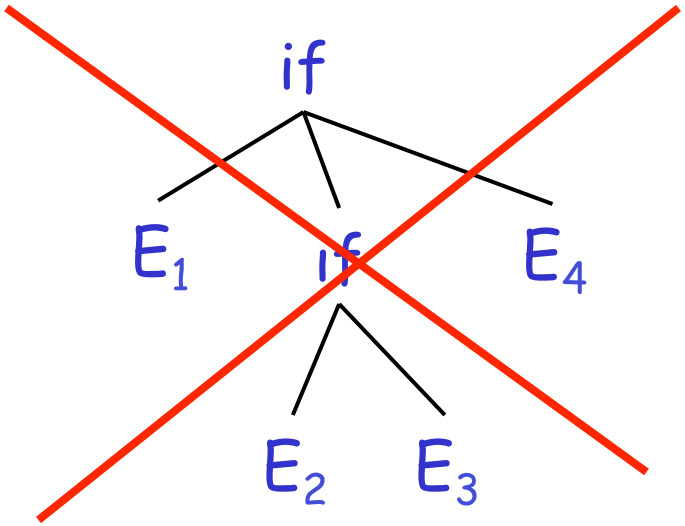
- else matches the closest unmatched then
- We can describe this in the grammar

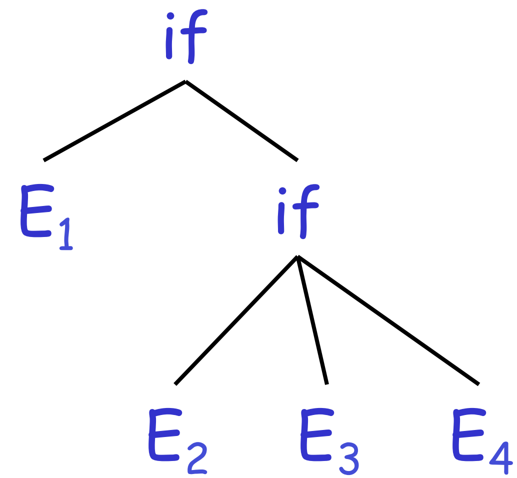
```
E → MIF                    /* all then are matched */
  | UIF                    /* some then is unmatched */
MIF → if E then MIF else MIF
    | OTHER
UIF → if E then E
    | if E then MIF else UIF
```

- Describes the same set of strings (Associativity)

The Dangling Else: Example Revisited

- The expression if E_1 then if E_2 then E_3 else E_4





- Not valid because the then expression is not a MIF

- A valid parse tree (for a UIF)

Ambiguity

- No general techniques for handling ambiguity
- Impossible to convert automatically an ambiguous grammar to an unambiguous one
- Used with care, ambiguity can simplify the grammar
  - Sometimes allows more natural definitions
  - We need disambiguation mechanisms

Precedence and Associativity Declarations

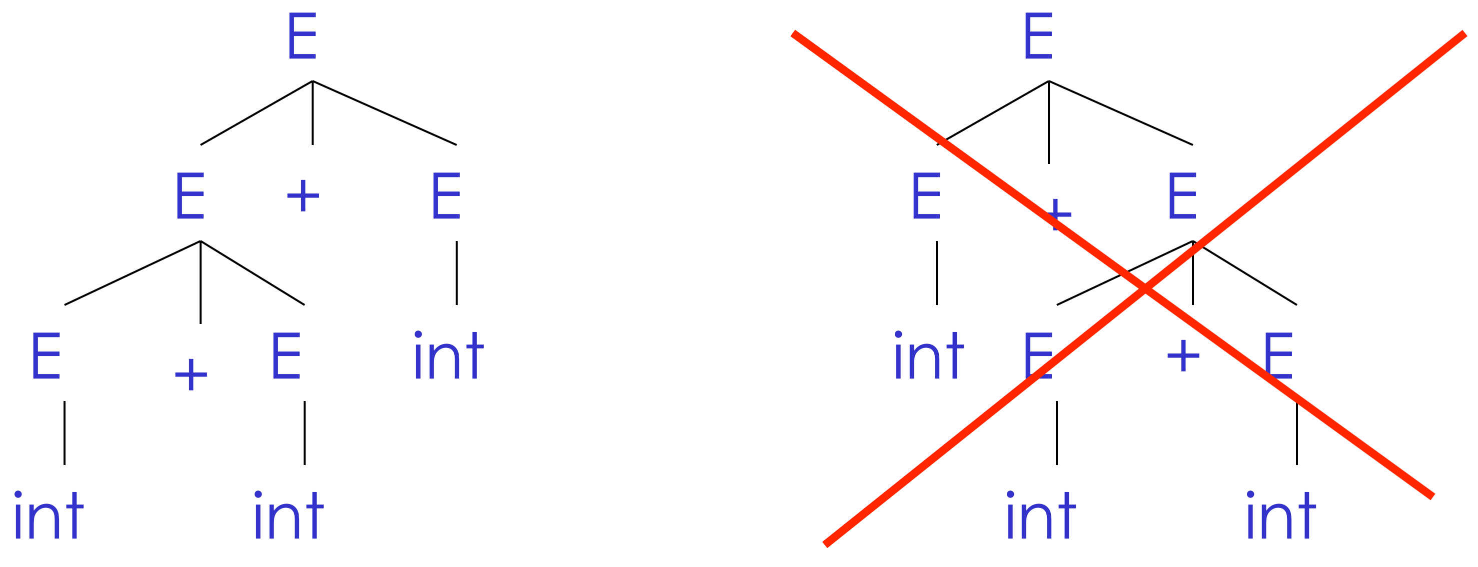
- Instead of rewriting the grammar
  - Use the more natural (ambiguous) grammar
  - Along with disambiguating declarations

- Most grammar generators allow <u>precedence and associativity declarations</u> to disambiguate grammars

- Examples …

Associativity Declarations

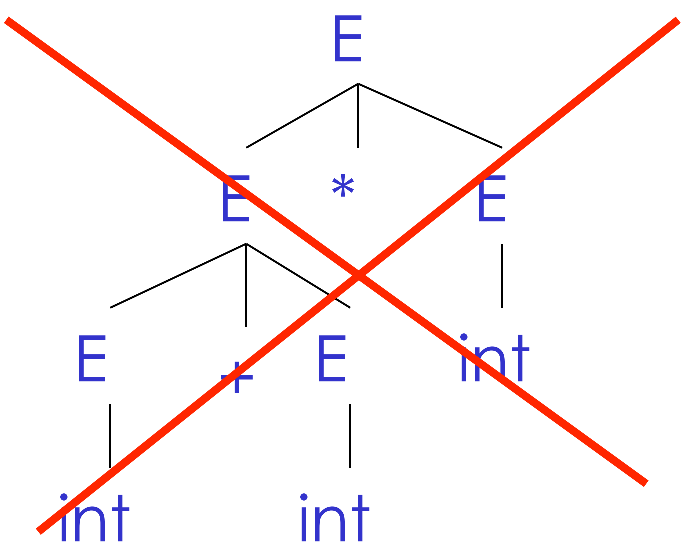
- Consider the grammar $E \rightarrow E + E \mid int$
- Ambiguous: two parse trees of int + int + int

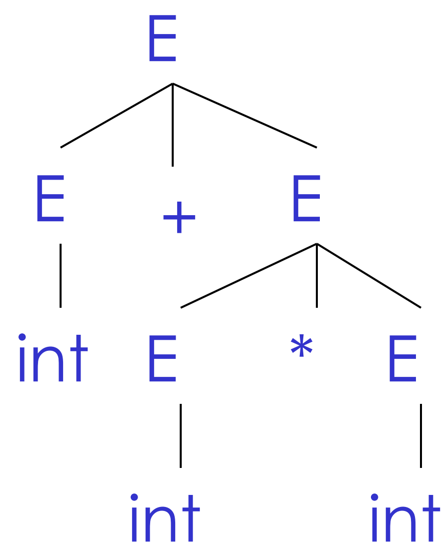


- Left associativity declaration: %left +

Precedence Declarations

- Consider the grammar $E \rightarrow E + E \mid E * E \mid int$
  - And the string int + int * int





- Precedence declarations: %left +
  %left *