

# Context-Free Grammars

CMPT 379: Compilers

Instructor: Anoop Sarkar

[anoopsarkar.github.io/compiler-class](https://anoopsarkar.github.io/compiler-class)

# Ambiguity

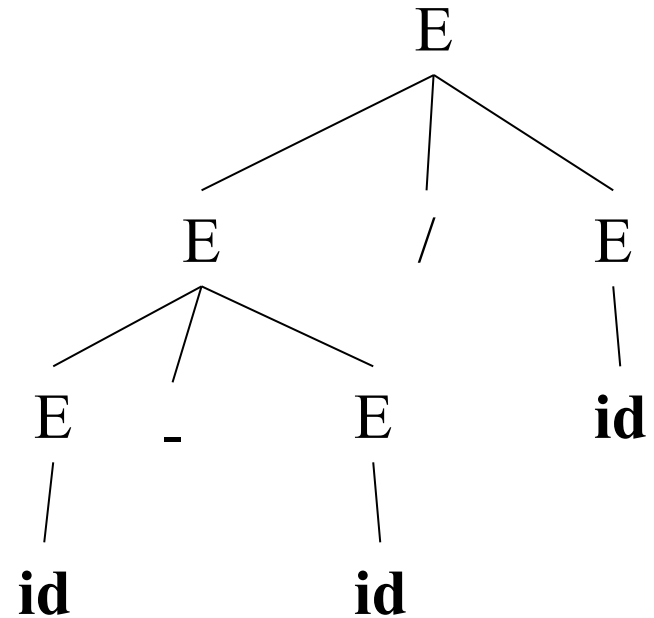
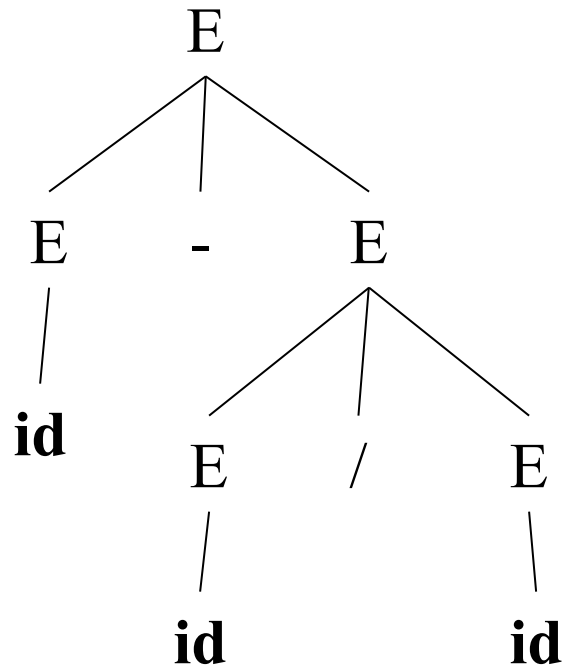
$E \rightarrow E - E$

$E \rightarrow E / E$

$E \rightarrow ( E )$

$E \rightarrow \text{id}$

**id - id / id**



# Ambiguity

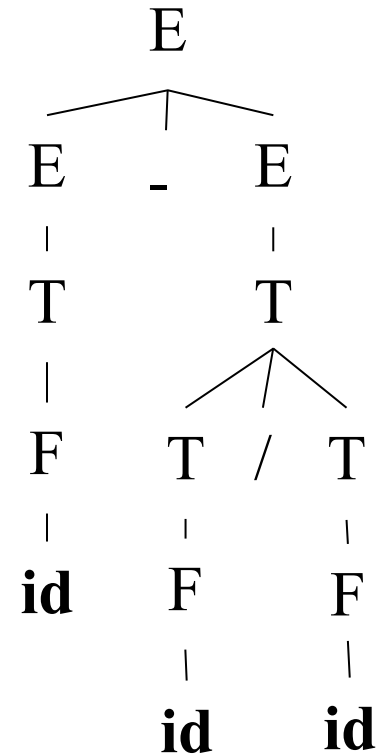
- Grammar is ambiguous if more than one parse tree is possible for some sentences
  - There is more than one leftmost (or rightmost) derivations
- Ambiguity is not acceptable in programming languages
  - Leaves meaning of some programs ill-defined
  - Unfortunately, it's undecidable to check whether a given CFG is ambiguous
  - Some CFLs are inherently ambiguous (do not have an unambiguous CFG)

# Ambiguity

- Handle ambiguity:
  - Rewrite the grammar unambiguously
  - Augment parser by enforcing precedence and associativity
- Consider the original ambiguous grammar:
$$\begin{array}{ll} E \rightarrow E - E & E \rightarrow E / E \\ E \rightarrow ( E ) & E \rightarrow \text{id} \end{array}$$
- How can we change the grammar to get only one tree for the input **id - id / id**

# Precedence

- Original ambiguous grammar:
  - $E \rightarrow E - E$        $E \rightarrow E / E$
  - $E \rightarrow ( E )$        $E \rightarrow \text{id}$
- Use different non-terminals for each Precedence level: (start from lowest level)
  - $E \rightarrow E - E$        $E \rightarrow T$
  - $T \rightarrow T / T$        $T \rightarrow F$
  - $F \rightarrow \text{id}$        $F \rightarrow ( E )$
- Input:  $\text{id} - \text{id} / \text{id}$



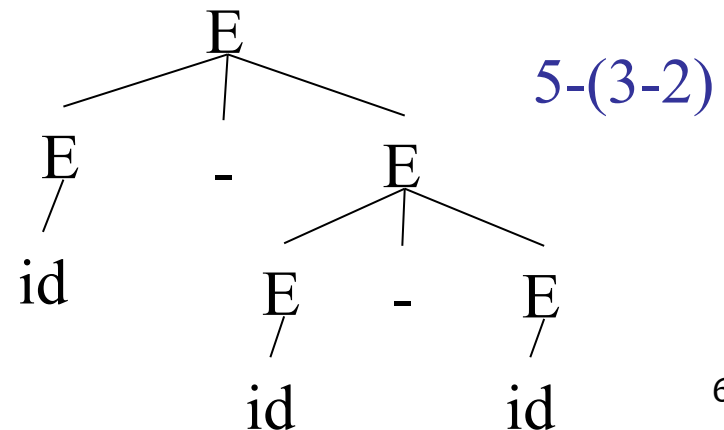
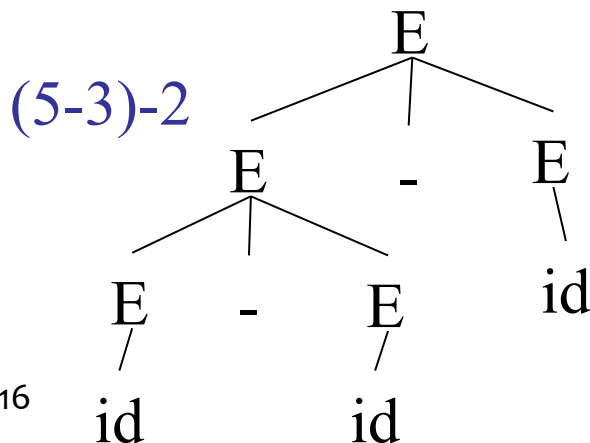
# Associativity

- The grammar capture operator precedence

- $E \rightarrow E - E$        $E \rightarrow T$
  - $T \rightarrow T / T$        $T \rightarrow F$
  - $F \rightarrow \text{id}$        $F \rightarrow ( E )$

- Still ambiguous!!  $\text{id} - \text{id} - \text{id}$  5-3-2

- “-” is left associative (operations are grouped from left)



# Recursion

- Grammar is **recursive** in nonterminal  $X$  if:
  - $X \rightarrow^+ \dots X \dots$ 
    - $\rightarrow^+$  means in one or more steps,  $X$  derives a sequence of symbols that includes  $X$
- Grammar is **left recursive** in  $X$  if:
  - $X \rightarrow^+ X \dots$ 
    - In one or more steps,  $X$  derives a sequence of symbols that **starts** with  $X$
- Grammar is **right recursive** in  $X$  if:
  - $X \rightarrow^+ \dots X$ 
    - In one or more steps,  $X$  derives a sequence of symbols that **ends** with  $X$

# Fix Associativity

- Left and right recursive in non-terminals E and T
  - $E \rightarrow E - E$                        $E \rightarrow T$
  - $T \rightarrow T / T$                        $T \rightarrow F$
  - $F \rightarrow \mathbf{id}$                        $F \rightarrow ( E )$
- Express operator associativity:
  - For left associativity use left recursion
  - For right associativity use right recursion
- Unambiguous grammar
  - $E \rightarrow E - T$                        $E \rightarrow T$
  - $T \rightarrow T / F$                        $T \rightarrow F$
  - $F \rightarrow \mathbf{id}$                        $F \rightarrow ( E )$



# Dangling else ambiguity

- Original Grammar (ambiguous)

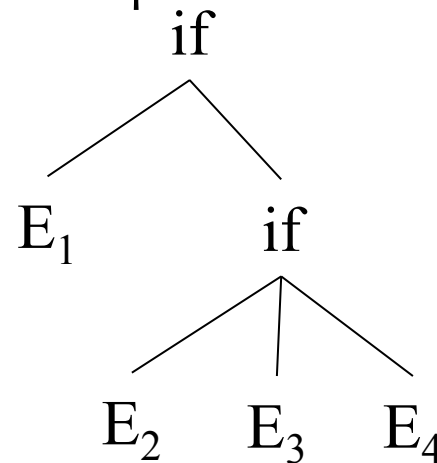
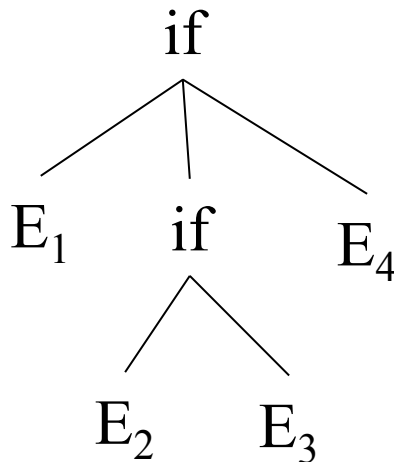
Stmt  $\rightarrow$  if Expr then Stmt else Stmt

Stmt  $\rightarrow$  if Expr then Stmt

Stmt  $\rightarrow$  Other

**else matches the closest unmatched then**

- if  $E_1$  then if  $E_2$  then  $E_3$  else  $E_4$



# Dangling else ambiguity

- Original Grammar (ambiguous)

Stmt  $\rightarrow$  if Expr then Stmt else Stmt

Stmt  $\rightarrow$  if Expr then Stmt

Stmt  $\rightarrow$  Other

**else matches the closest  
unmatched then**

- Unambiguous grammar

Stmt  $\rightarrow$  MatchedStmt      /\*all then are matched\*/

Stmt  $\rightarrow$  UnmatchedStmt    /\*some then are unmatched\*/

MatchedStmt  $\rightarrow$  if Expr then MatchedStmt else MatchedStmt

MatchedStmt  $\rightarrow$  Other

UnmatchedStmt  $\rightarrow$  if Expr then Stmt

UnmatchedStmt  $\rightarrow$  if Expr then MatchedStmt else UnmatchedStmt

# Dangling else ambiguity

- Check unambiguous dangling-else grammar with the following inputs:
  - if Expr then if Expr then Other else Other
  - if Expr then if Expr then Other else Other else Other
  - if Expr then if Expr then Other else if Expr then Other else Other

# Precedence and Associativity Declaration

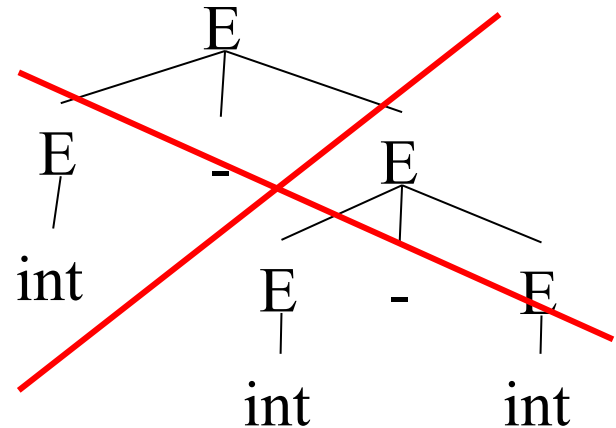
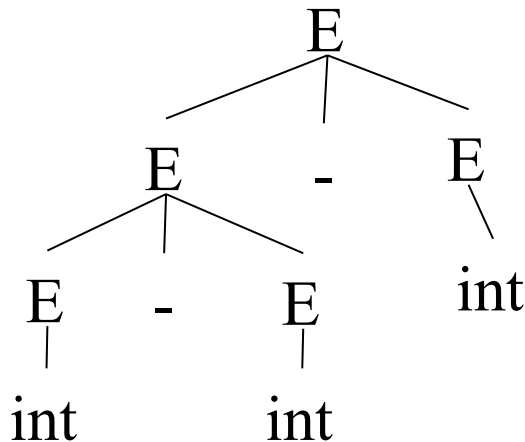
- Impossible to automatically convert an ambiguous grammar to an unambiguous one
- Used with care, ambiguity can simplify the grammar
  - Sometimes allow more natural definitions
  - We need disambiguation mechanisms

# Precedence and Associativity Declaration

- Instead of re-writing the grammar
  - Use the more natural (ambiguous) grammar
  - Along with disambiguation declarations
- Most tools allow **precedence and associativity declaration** to disambiguate grammars

# Associativity Declaration

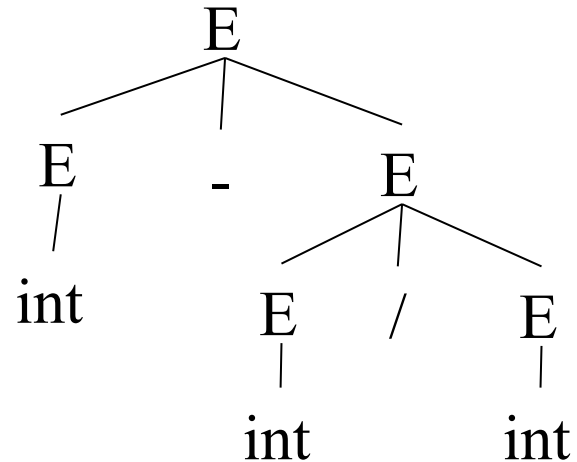
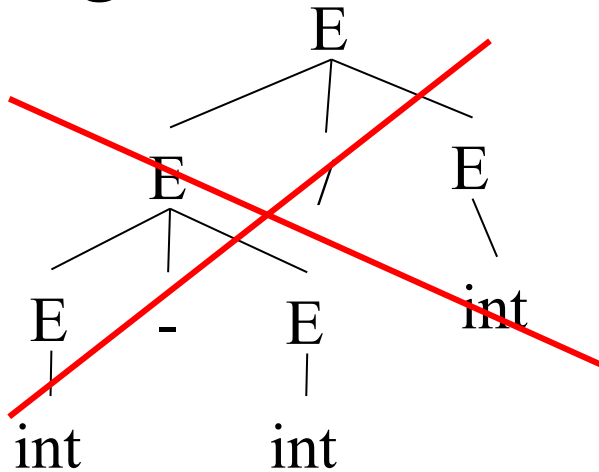
- Consider the grammar:
  - $E \rightarrow E - E \mid \text{int}$
- Ambiguous: two parse trees  $\text{int} - \text{int} - \text{int}$



- Left associativity declaration: `%left -`

# Precedence Declaration

- Consider the grammar:
  - $E \rightarrow E - E \mid E / E \mid \text{int}$
- Ambiguous: two parse trees  $\text{int} - \text{int} / \text{int}$



- Precedence declaration:  $\%left -$   
 $\%left /$

# Other Ambiguous Grammars

- Consider the grammar
$$\begin{array}{l} R \rightarrow R \mid R \\ \mid R R \\ \mid R '*' \\ \mid '(' R ')' \\ \mid a \\ \mid b \end{array}$$
- What does this grammar generate?
- What's the parse tree for  $a|b*a$
- Is this grammar ambiguous?