Ambiguity

Lecture 8

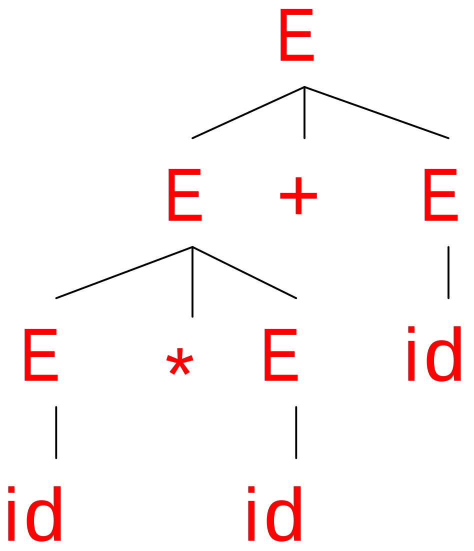
Announcement

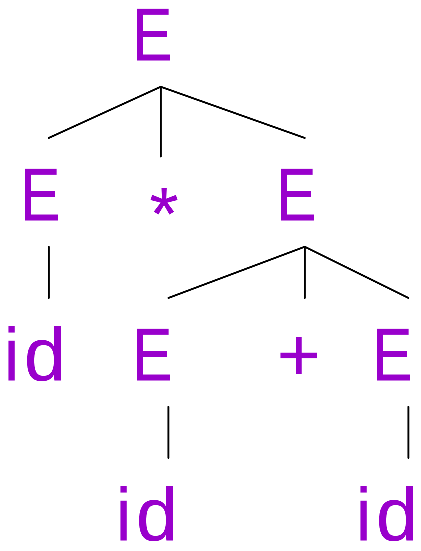
- Reading Assignment
  - "Context-Free Grammars" (Sections 4.1, 4.2)
- Programming Assignment 2
  - due Friday!
- Homework 1
  - due in a week (Wed Feb 21)
  - ***not Feb 25!***

Ambiguity = program structure not defined

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

String id * id + id has two parse trees:





Ambiguity

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

  (the three conditions are equivalent)

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

Dealing with Ambiguity

- There are several ways to handle ambiguity
- We'll discuss two of them:
  - rewriting the grammar
  - parser-generator declarations

Outline

- Rewriting:
  - Expression Grammars
    - precedence
    - associativity
  - IF-THEN-ELSE
    - the Dangling-ELSE problem
- Declarations
  - Expression Grammars
    - precedence
    - associativity

Expression Grammars (precedence)

- Rewrite the grammar
  - use a different nonterminal for each precedence level
  - start with the lowest precedence (MINUS)

E → E - E | E / E | (E) | id

rewrite to

E → E - E | T
T → T / T | F
F → id | (E)

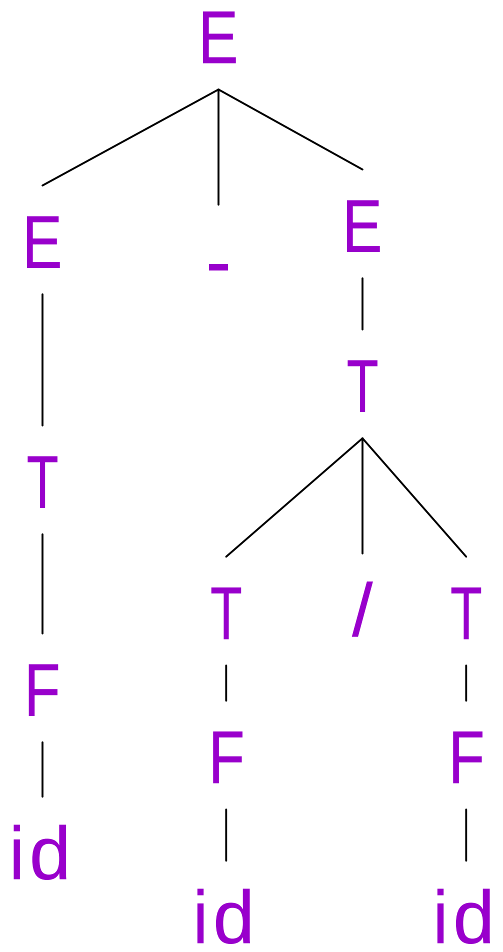
Example

parse tree for id – id / id

E → E - E | T

T → T / T | F

F → id | (E)



TEST YOURSELF #1

- Attempt to construct a parse tree for id-id/id that shows the *wrong* precedence.
- **Question:**
  - Why do you fail to construct this parse tree?

Associativity

- The grammar captures operator precedence, but it is still ambiguous!
  - fails to express that both subtraction and division are *left* associative;
    - e.g., 5-3-2 is equivalent to: ((5-3)-2) and *not* to: (5-(3-2)).

- **TEST YOURSELF #3**
  - Draw two parse trees for the expression 5-3-2 using the grammar given above; one that correctly groups 5-3, and one that incorrectly groups 3-2.

Recursion

- A grammar is **recursive in nonterminal X** if:
  - $X \rightarrow^{+} \ldots X \ldots$
    - recall that $\rightarrow^{+}$ means “in one or more steps, X derives a sequence of symbols that includes an X”
- A grammar is **left** recursive in X if:
  - $X \rightarrow^{+} X \ldots$
    - in one or more steps, X derives a sequence of symbols that *starts* with an X
- A grammar is **right** recursive in X if:
  - $X \rightarrow^{+} \ldots X$
    - in one or more steps, X derives a sequence of symbols that *ends* with an X

How to fix associativity

- The grammar given above is both left and right recursive in nonterminals exp and term
  - try at home: write the derivation steps that show this.

- To correctly expresses operator associativity:
  - For left associativity, use left recursion.
  - For right associativity, use right recursion.

- Here's the correct grammar:

  E → E – T | T
  T → T / F | F
  F → id | (E)

Ambiguity: The Dangling Else

- Consider the grammar

$$
\begin{aligned}
E \rightarrow\ & \text{if } E \text{ then } E \\
\mid\ & \text{if } E \text{ then } E \text{ else } E \\
\mid\ & \text{print}
\end{aligned}
$$

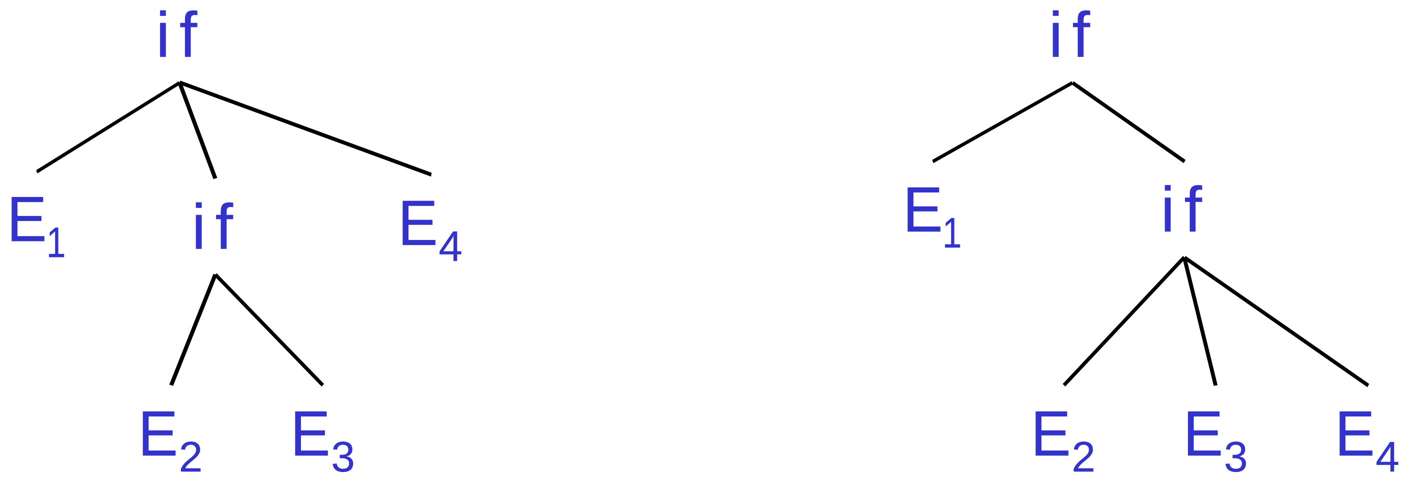
- This grammar is also ambiguous

The Dangling Else: Example

- The expression

if E_1 then if E_2 then E_3 else E_4

has two parse trees



- Typically we want the second form

The Dangling Else: A Fix

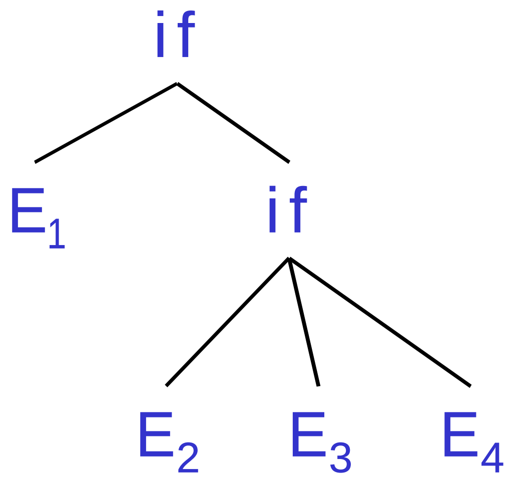
- else matches the closest unmatched then
- We can describe this in the grammar

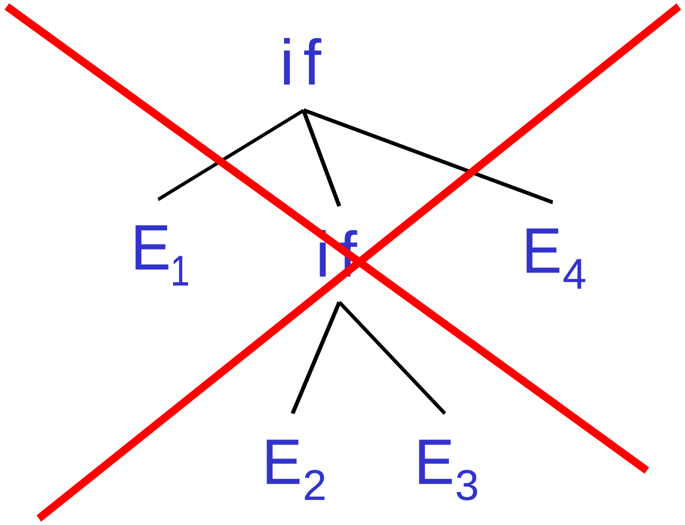
```
E →  MIF                    /* all then are matched */
   | UIF                    /* some then are unmatched */
MIF → if E then MIF else MIF
    | print
UIF → if E then E
    | if E then MIF else UIF
```

- Describes the same set of strings

The Dangling Else: Example Revisited

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





- A valid parse tree (for a UIF)

- Not valid because the then expression is not a MIF

Precedence and Associativity Declarations

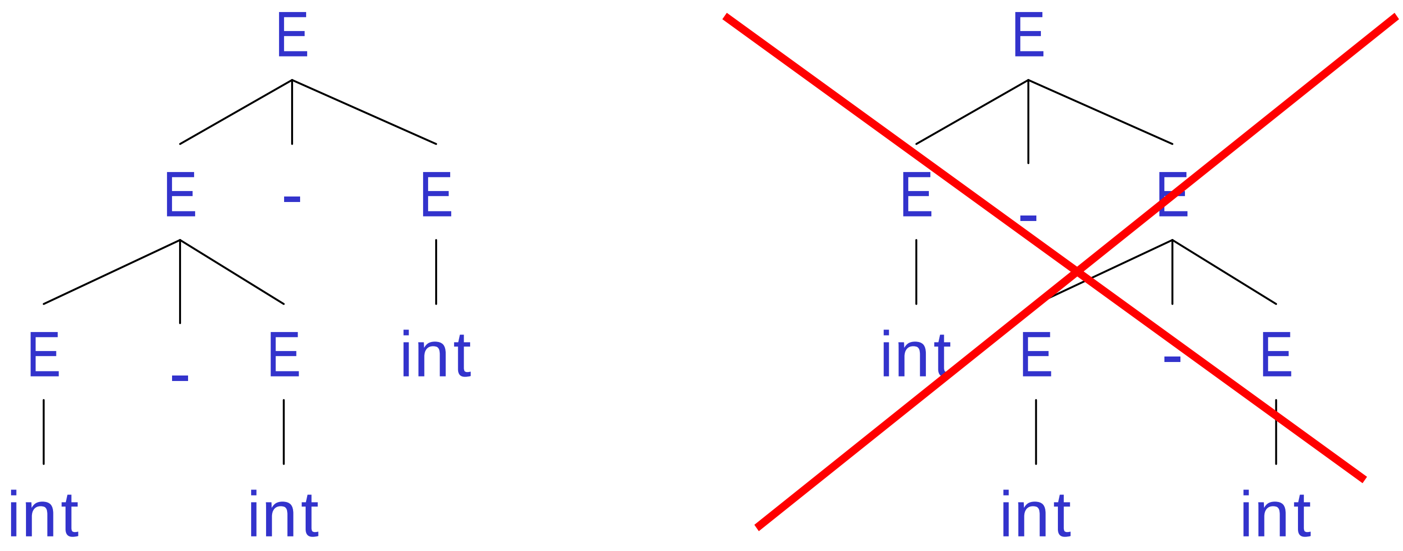
- Instead of rewriting the grammar
  - Use the more natural (ambiguous) grammar
  - Along with disambiguating declarations

- Most parser generators allow precedence and associativity declarations to disambiguate grammars

- Examples …

Associativity Declarations

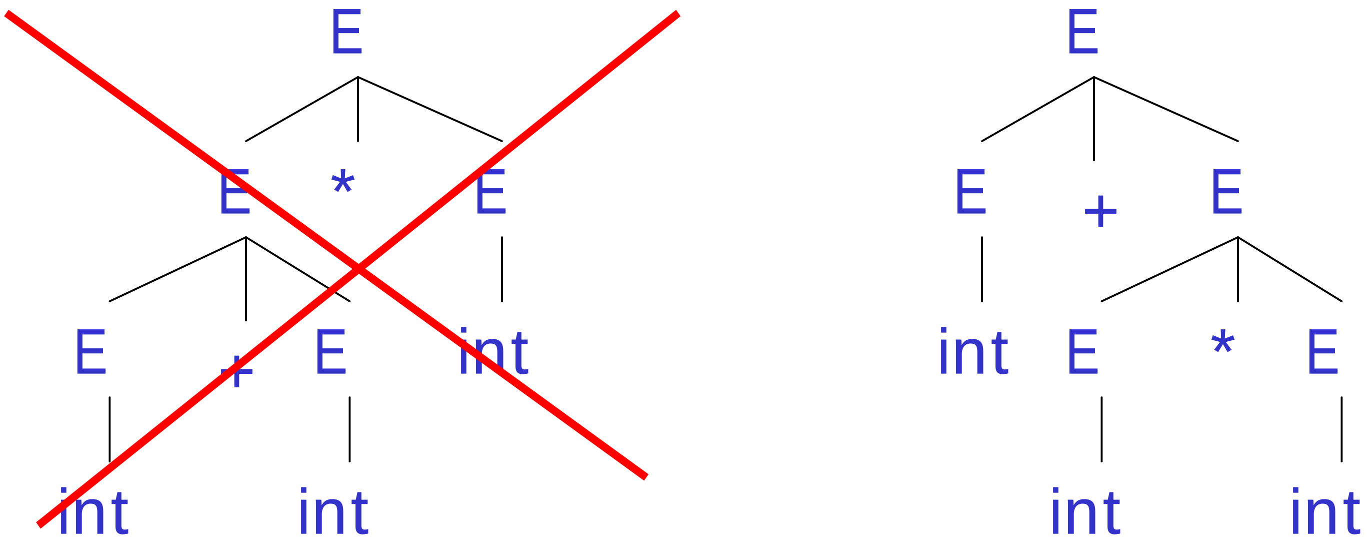
- Consider the grammar E → E - E | int
- Ambiguous: two parse trees of int - int - int



- Left associativity declaration: %left +

Precedence Declarations

- Consider the grammar E → E + E | E * E | int
  - And the string int + int * int



- Precedence declarations: %left +
  %left *