

CS 536

CFGs for Syntax Definition

Roadmap

- Last time
 - Defined context-free grammar basics
- This time
 - CFGs for syntax design
 - Language membership
 - List grammars
 - Resolving ambiguity

CFG Review

- $G = (N, \Sigma, P, S)$
- \Rightarrow means *derives*
+
 \Rightarrow means *derives in 1 or more steps*
- CFG generates a string by applying productions until no non-terminals remain

Example: Nested parens

$$N = \{ Q \}$$

$$\Sigma = \{ (,) \}$$

$$P = Q \rightarrow (Q)$$

| ϵ

$$S = Q$$

Formal CFG Language Definition

Let $G = (N, \Sigma, P, S)$ be a CFG. Then

implement the rule one or more times

$$L(G) = \left\{ w \mid S \xRightarrow{+} w \right\} \text{ where}$$

S is the start nonterminal of G

w is a sequence of terminals or ε

CFGs as Language Definition

- CFG productions define the *syntax* of a language

1. $Prog \rightarrow \mathbf{begin} \textit{Stmts} \mathbf{end}$
2. $Stmts \rightarrow \textit{Stmts} \mathbf{semicolon} \textit{Stmt}$
3. $\quad \quad \quad | \textit{Stmt}$
4. $Stmt \rightarrow \mathbf{id} \mathbf{assign} \textit{Expr}$
5. $Expr \rightarrow \mathbf{id}$
6. $\quad \quad \quad | \textit{Expr} \mathbf{plus} \mathbf{id}$

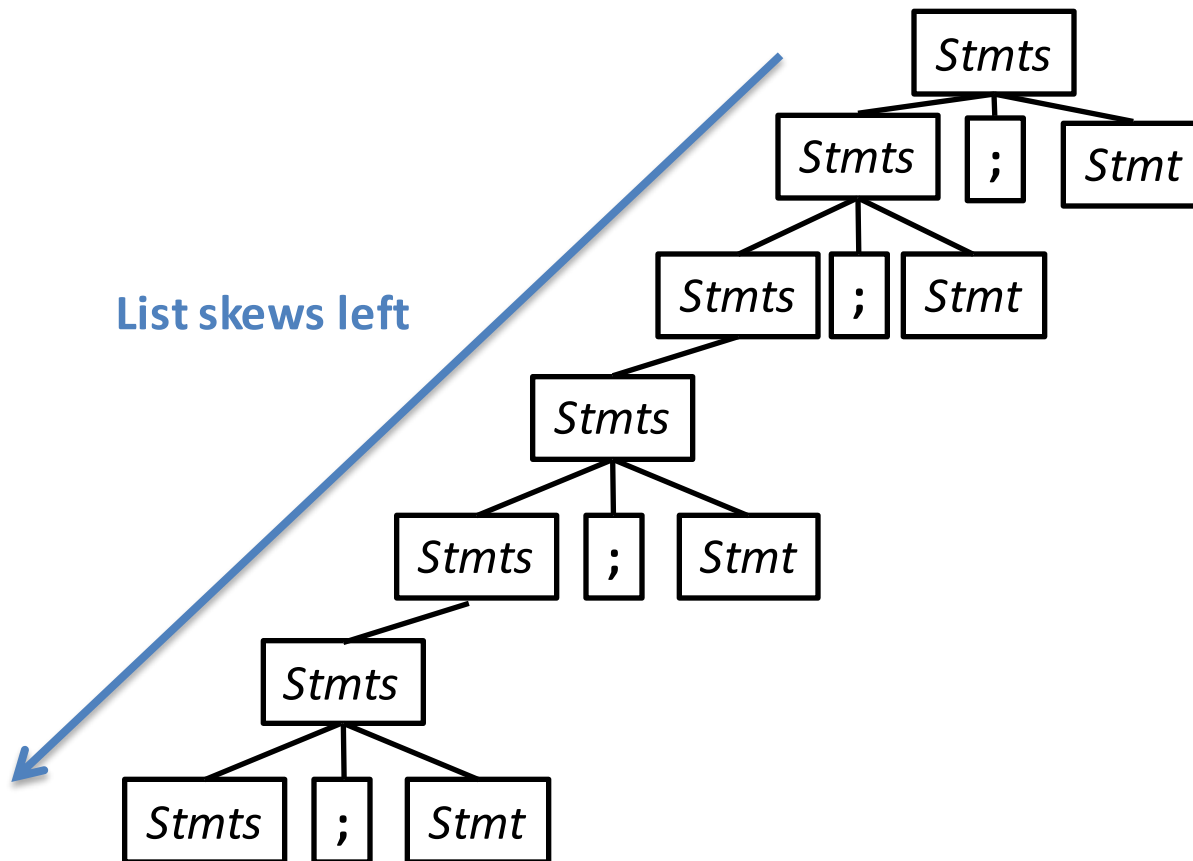
Backus-Naur form

- We call this notation “BNF” or “*enhanced BNF*”
- HTTP grammar using BNF:
 - <http://www.w3.org/Protocols/rfc2616/rfc2616-sec2.html>

List Grammars

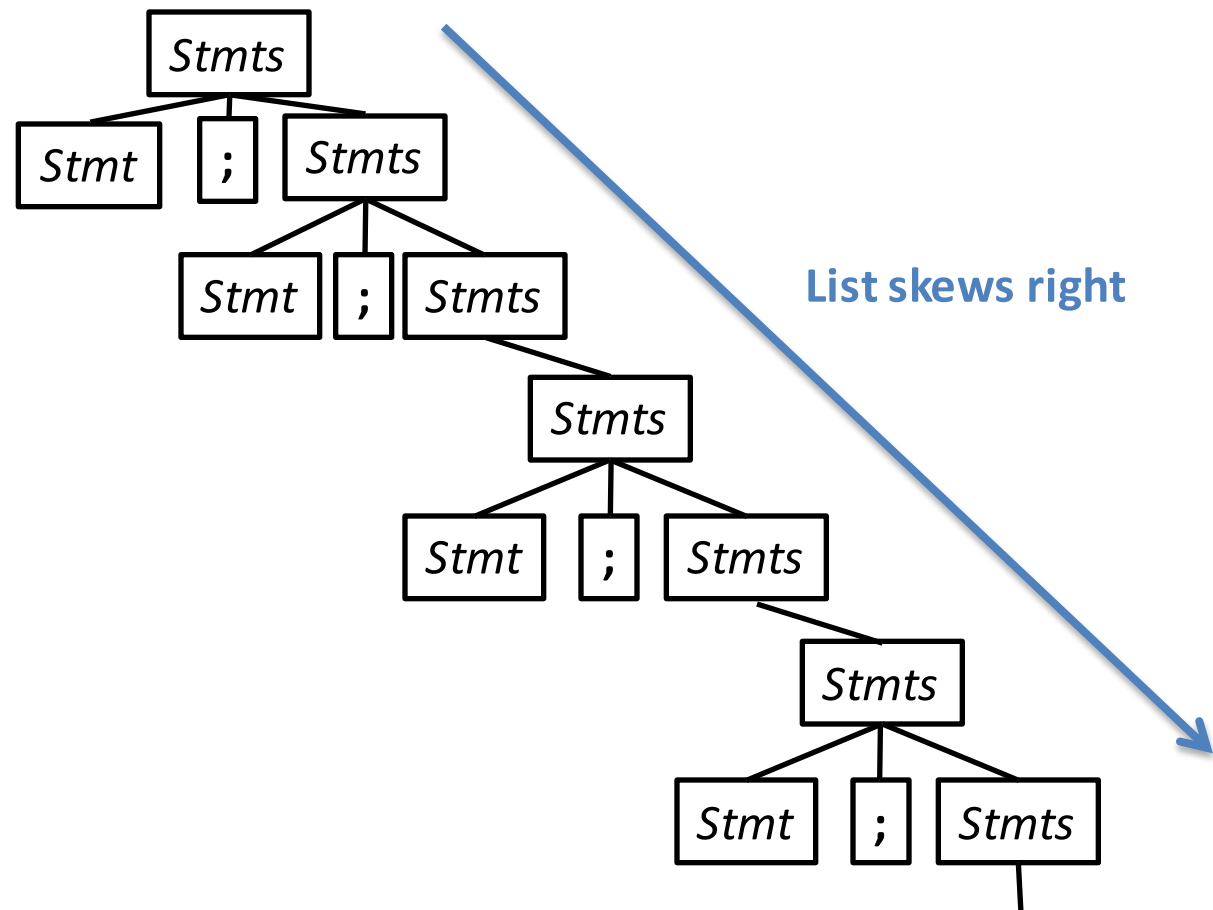
- Useful to repeat a structure arbitrarily often

$Stmts \rightarrow Stmts \text{ semicolon } Stmt \mid Stmt$



List Grammars

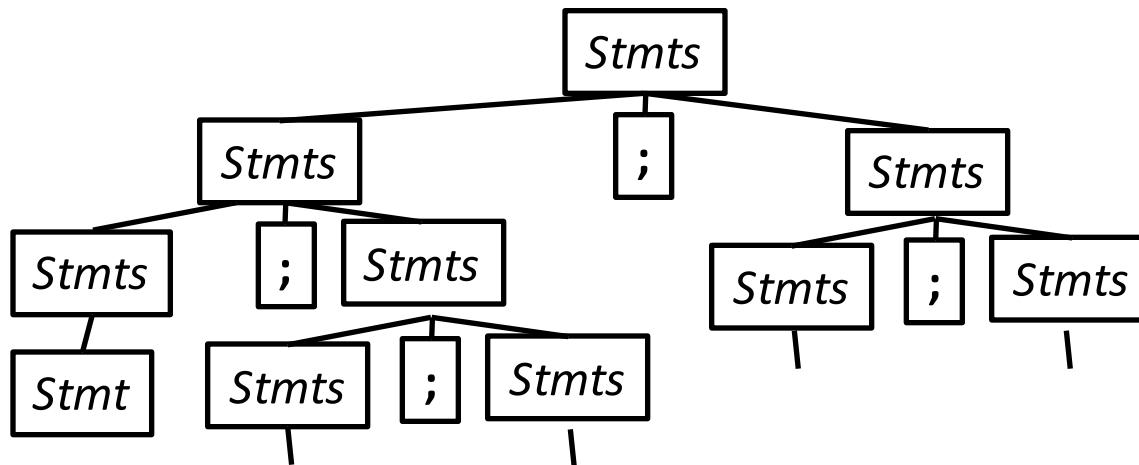
- Useful to repeat a structure arbitrarily often

$$Stmts \rightarrow Stmt \text{ semicolon } Stmts \mid Stmt$$


List Grammars

- What if we allowed both “skews”?

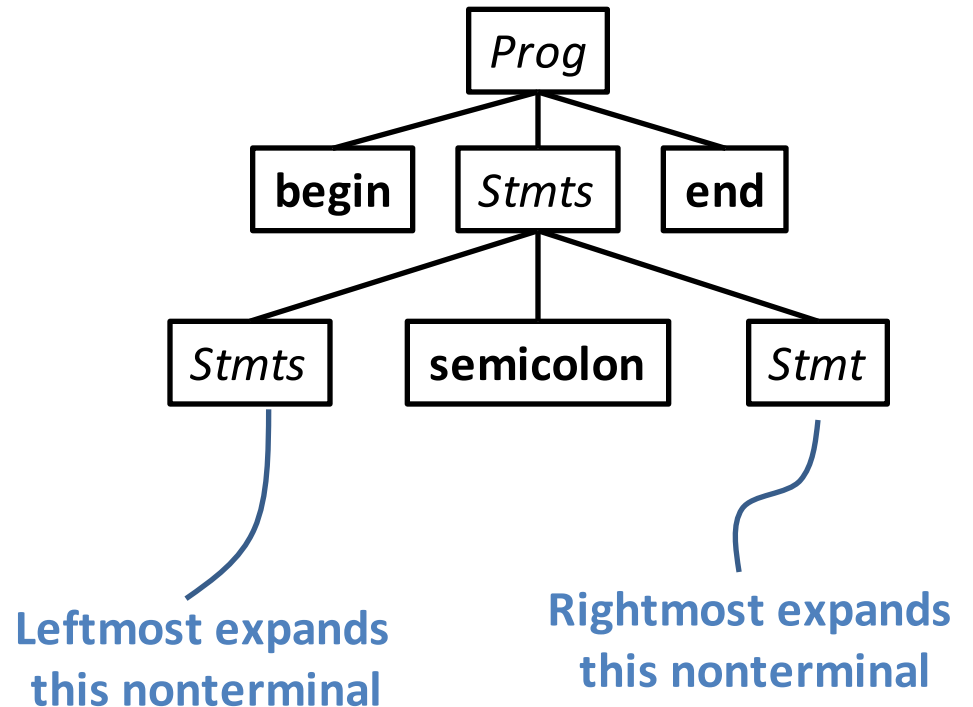
$Stmts \rightarrow Stmts \text{ semicolon } Stmts \mid Stmt$



Derivation Order

- Leftmost Derivation: always expand the leftmost nonterminal
- Rightmost Derivation: always expand the rightmost nonterminal

1. $Prog \rightarrow \mathbf{begin} \textit{Stmts} \mathbf{end}$
2. $Stmts \rightarrow \textit{Stmts} \mathbf{semicolon} \textit{Stmt}$
3. | \textit{Stmt}
4. $Stmt \rightarrow \mathbf{id} \mathbf{assign} \textit{Expr}$
5. $Expr \rightarrow \mathbf{id}$
6. | $\textit{Expr} \mathbf{plus id}$



Ambiguity

- Even with a fixed derivation order, it is possible to derive the same string in multiple ways
- For Grammar G and string w
 - G is ambiguous if
 - >1 leftmost derivation of w
 - >1 rightmost derivation of w
 - > 1 parse tree for w

Example: Ambiguous Grammars

$Expr \rightarrow \text{intlit}$

| $Expr \text{ minus } Expr$

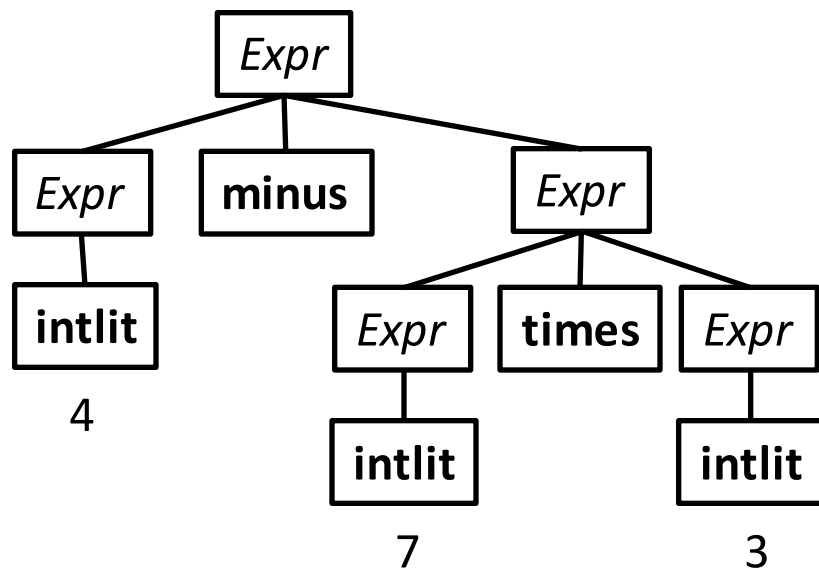
| $Expr \text{ times } Expr$

| $\text{lparen } Expr \text{ rparen}$

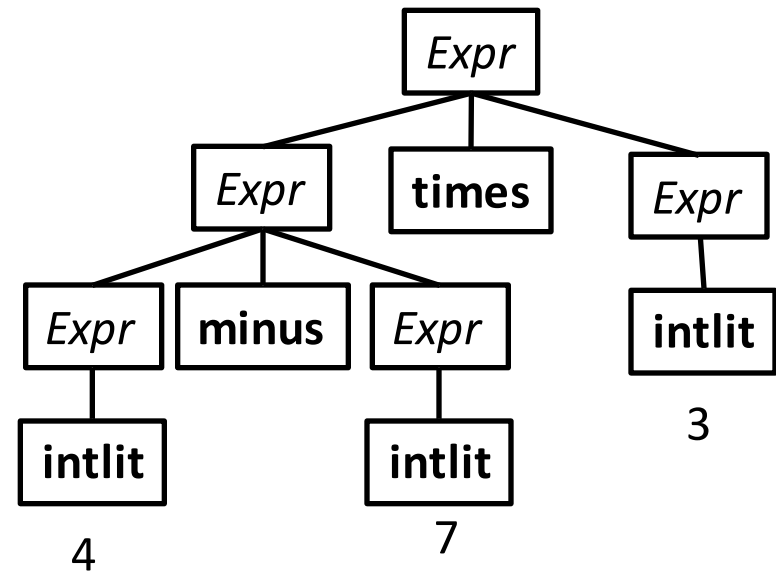
Derive the string $4 - 7 * 3$

(assume tokenization)

Parse Tree 1

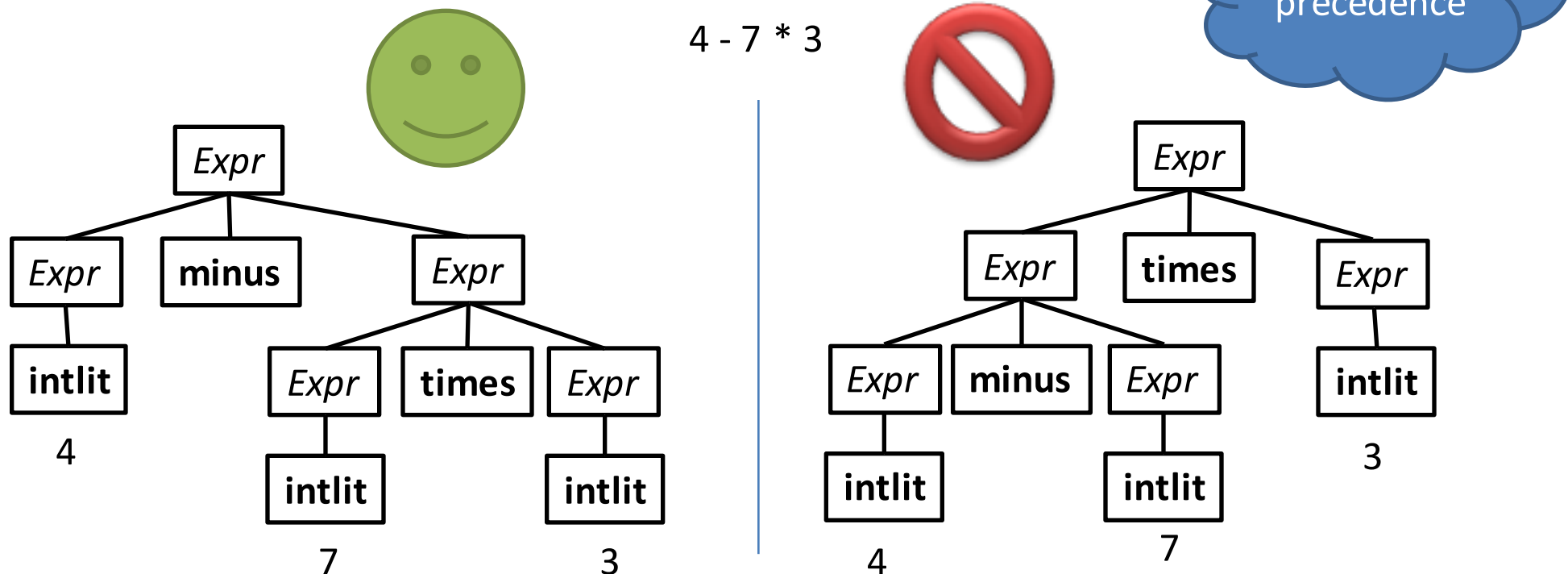


Parse Tree 2



Why is Ambiguity Bad?

- Eventually, we'll be using CFGs as the basis for our parser
 - Parsing is much easier when there is no ambiguity in the grammar
 - The parse tree may mismatch user understanding!



Resolving Grammar Ambiguity: Precedence

Expr → **intlit**
| *Expr* **minus** *Expr*
| *Expr* **times** *Expr*
| **lparen** *Expr* **rparen**

- Intuitive problem
 - “Context-freeness”
 - Nonterminals are the same for both operators
- To fix precedence
 - 1 nonterminal per precedence level
 - Parse lowest level first

Resolving Grammar Ambiguity: Precedence

$Expr \rightarrow \text{intlit}$

| $Expr \text{ minus } Expr$

| $Expr \text{ times } Expr$

| $\text{lparen } Expr \text{ rparen}$



$Expr \rightarrow Expr \text{ minus } Expr$

| $Term$

$Term \rightarrow Term \text{ times } Term$

| $Factor$

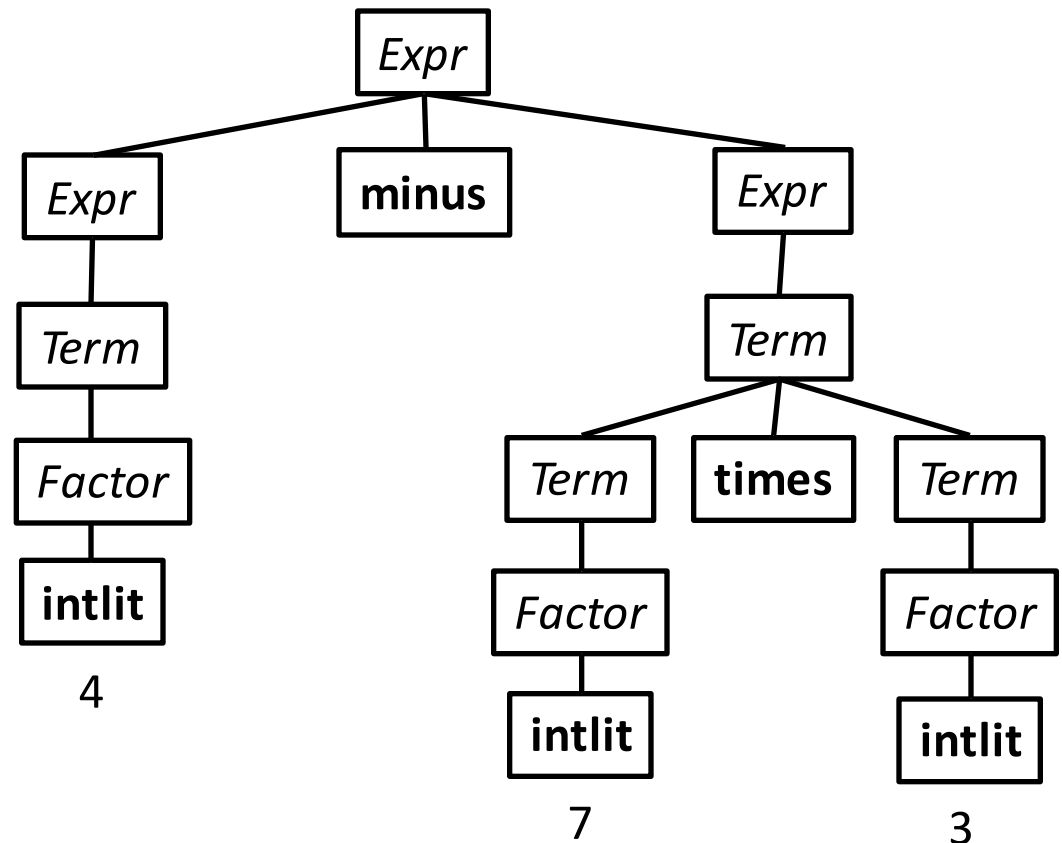
$Factor \rightarrow \text{intlit}$

| $\text{lparen } Expr \text{ rparen}$

lowest precedence level first

1 nonterm per precedence level

Derive the string $4 - 7 * 3$



Resolving Grammar Ambiguity: Precedence

Fixed Grammar

Expr → **expr** minus **expr**

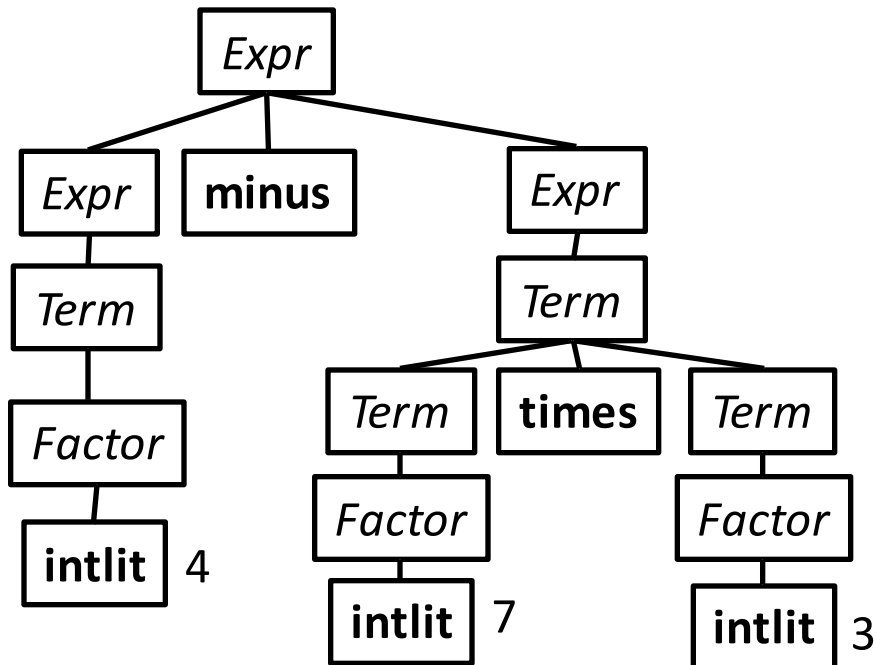
| *Term*

Term → *Term* **times** *Term*

| *Factor*

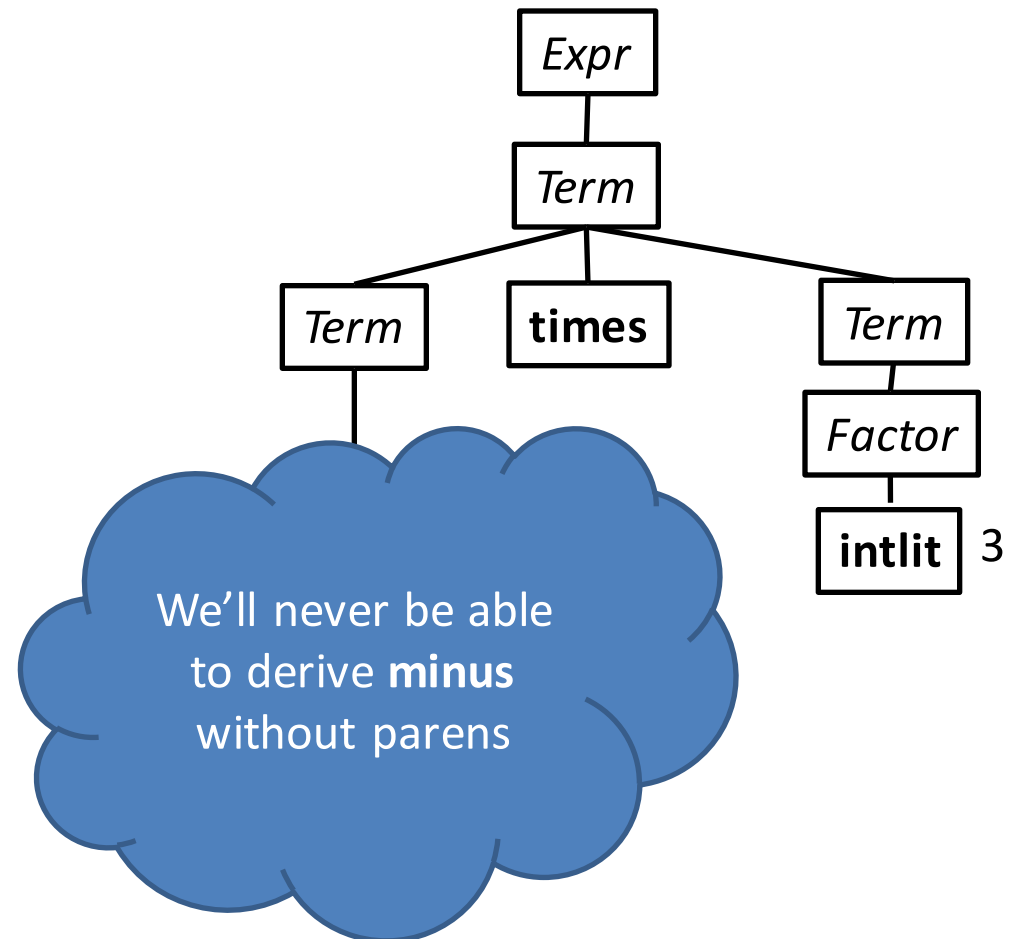
Factor → **intlit**

| **lparen** *Expr* **rparen**



Derive the string 4 - 7 * 3

Let's try to re-build the wrong parse tree



Did we fix all ambiguity?

Fixed Grammar

$Expr \rightarrow Expr \text{ minus } Expr$

| $Term$

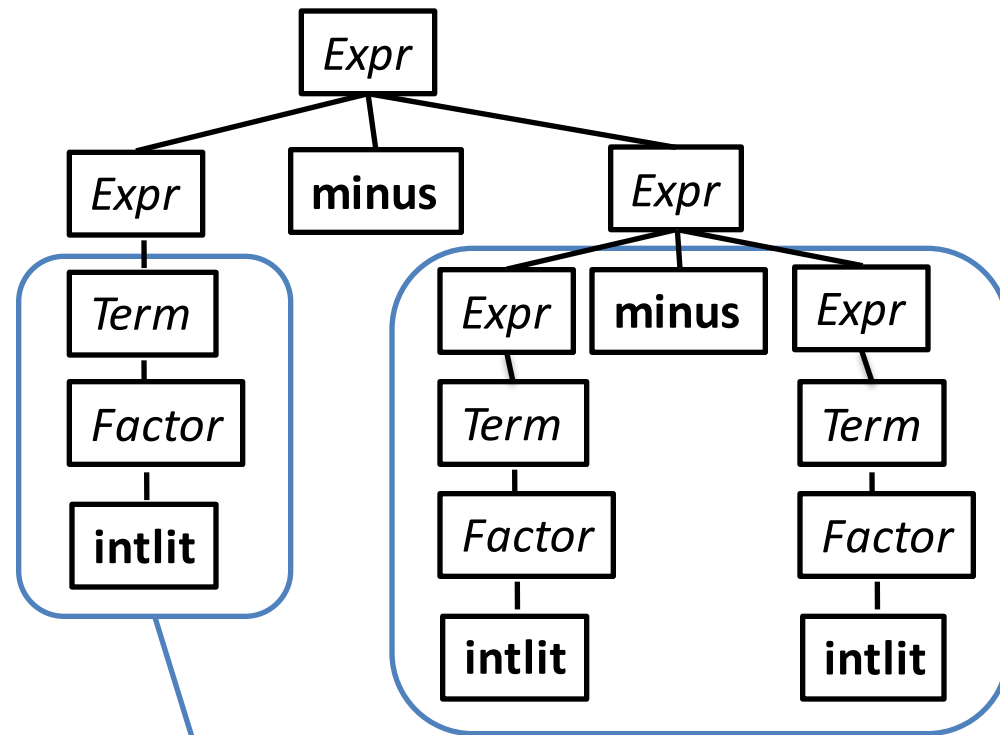
$Term \rightarrow Term \text{ times } Term$

| $Factor$

$Factor \rightarrow \text{intlit}$

| $\text{lparen } Expr \text{ rparen}$

Derive the string 4 - 7 - 3



NO!

These subtrees could have been swapped!

Where we are so far

- Precedence
 - We want correct behavior on $4 - 7 * 9$
 - A new nonterminal for each precedence level
- Associativity
 - We want correct behavior on $4 - 7 - 9$
 - Minus should be *left associative*: $a - b - c = (a - b) - c$
 - Problem: the *recursion* in a rule like

$Expr \rightarrow Expr \textbf{ minus } Expr$

Definition: Recursion in Grammars

- A grammar is *recursive* in (nonterminal) X if
$$X \stackrel{+}{\Rightarrow} \alpha X \gamma \text{ for non-empty strings of symbols } \alpha \text{ and } \gamma$$
- A grammar is *left-recursive* in X if
$$X \stackrel{+}{\Rightarrow} X \gamma \text{ for non-empty string of symbols } \gamma$$
- A grammar is *right-recursive* in X if
$$X \stackrel{+}{\Rightarrow} \alpha X \text{ for non-empty string of symbols } \alpha$$

Resolving Grammar Ambiguity: Associativity

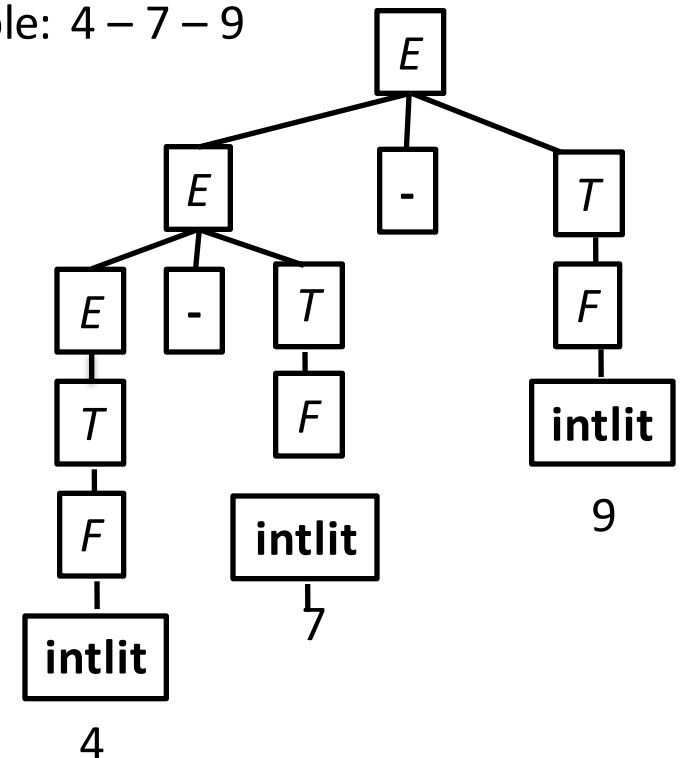
- We'll recognize left-associative operators with left-associative productions
- We'll recognize right-associative operators with right-associative productions

$Expr \rightarrow Expr \text{ minus } \cancel{Expr}$
 | *Term*

$Term \rightarrow Term \text{ times } \cancel{Term}$
 | *Factor*

$Factor \rightarrow \text{intlit} \mid \text{lparen } Expr \text{ rparen}$

Example: $4 - 7 - 9$



Resolving Grammar Ambiguity: Associativity

$Expr \rightarrow Expr \text{ minus } Term$

$| Term$

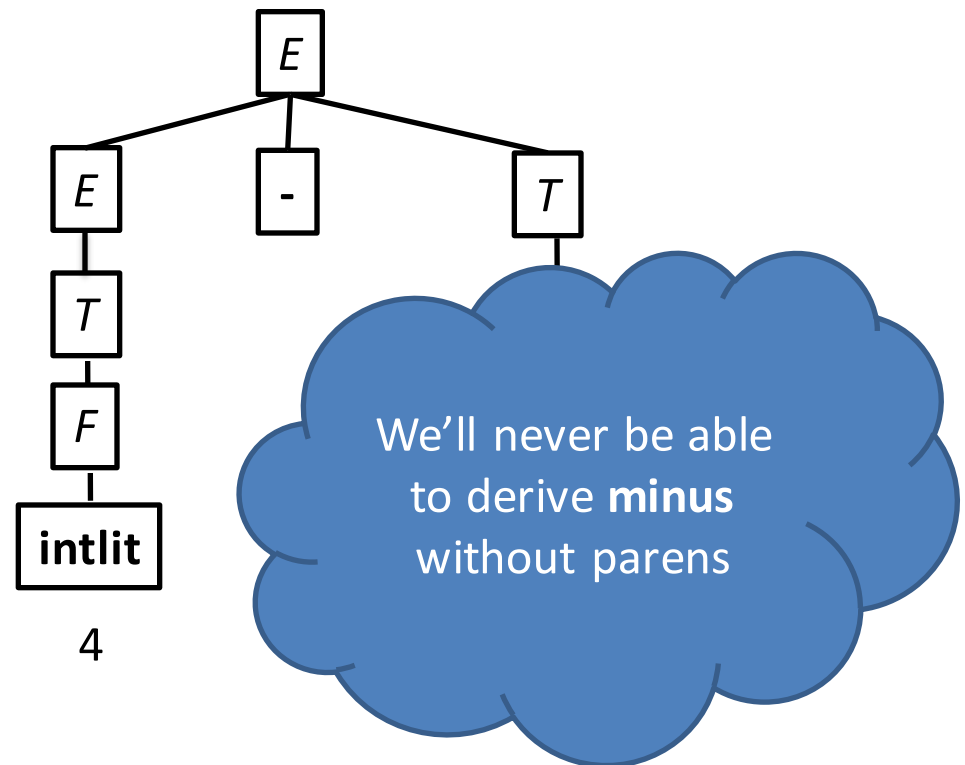
$Term \rightarrow Term \text{ times } Factor$

$| Factor$

$Factor \rightarrow \text{intlit} \mid \text{lparen } Expr \text{ rparen}$

Example: $4 - 7 - 9$

Let's try to re-build the wrong parse tree again



Example

- Language of Boolean expressions
 - $\text{bexp} \rightarrow \text{TRUE}$
 - $\text{bexp} \rightarrow \text{FALSE}$
 - $\text{bexp} \rightarrow \text{bexp OR bexp}$
 - $\text{bexp} \rightarrow \text{bexp AND bexp}$
 - $\text{bexp} \rightarrow \text{NOT bexp}$
 - $\text{bexp} \rightarrow \text{LPAREN bexp RPAREN}$
- Add nonterminals so that **OR** has lowest precedence, then **AND**, then **NOT**. Then change the grammar to reflect the fact that both **AND** and **OR** are left associative.
- Draw a parse tree for the expression:
 - true AND NOT true.

Another ambiguous example

Stmt \rightarrow

if Cond **then** Stmt |

if Cond **then** Stmt **else** Stmt | ...

Consider this word in this grammar:

if a **then** **if** b **then** s **else** s2

How would you derive it?

Summary

- To understand how a parser works, we start by understanding **context-free grammars**, which are used to define the language recognized by the parser.
 - terminal symbol
 - (non)terminal symbol
 - grammar rule (or production)
 - derivation (leftmost derivation, rightmost derivation)
 - parse (or derivation) tree
 - the language defined by a grammar
 - ambiguous grammar