Defining syntax using CFGs

Roadmap

Last time

- Defined context-free grammar

This time

- CFGs for syntax design
    - Language membership
    - List grammars
    - Resolving ambiguity

CFG Review

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

Example: Nested parens

N = { Q }

Σ = { **(** , **)** }

P = Q **→** **(** Q **)**

        | ε

S = Q

Formal CFG Language Definition

Let *G=* (N,Σ,P,S) be a CFG. Then

L(G) = $w \square S \Rightarrow + \perp w$ 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* → **begin** *Stmts* **end**
2. Stmts → *Stmts* **semicolon** *Stmt*
3. | *Stmt*
4. *Stmt* → **id assign** *Expr*
5. *Expr* → **id**
6. | *Expr* **plus id**

We call this notation "*BNF*" or "*extended BNF*"

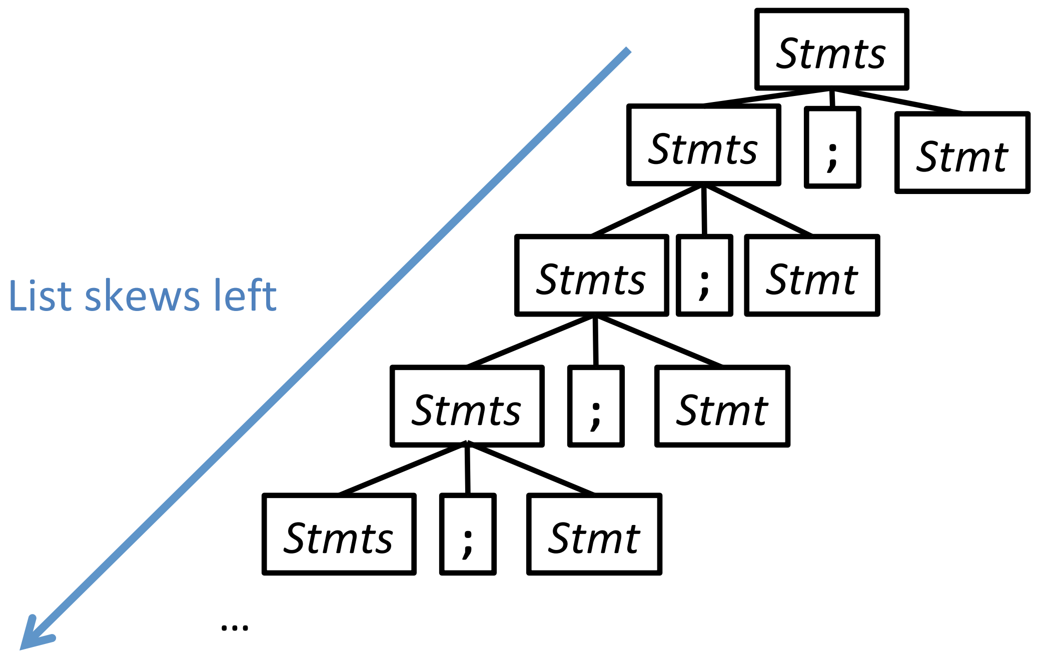
HTTP grammar using BNF:

– http://www.w3.org/Protocols/rfc2616/rfc2616-sec2.html

List Grammars

- Useful to repeat a structure arbitrarily often

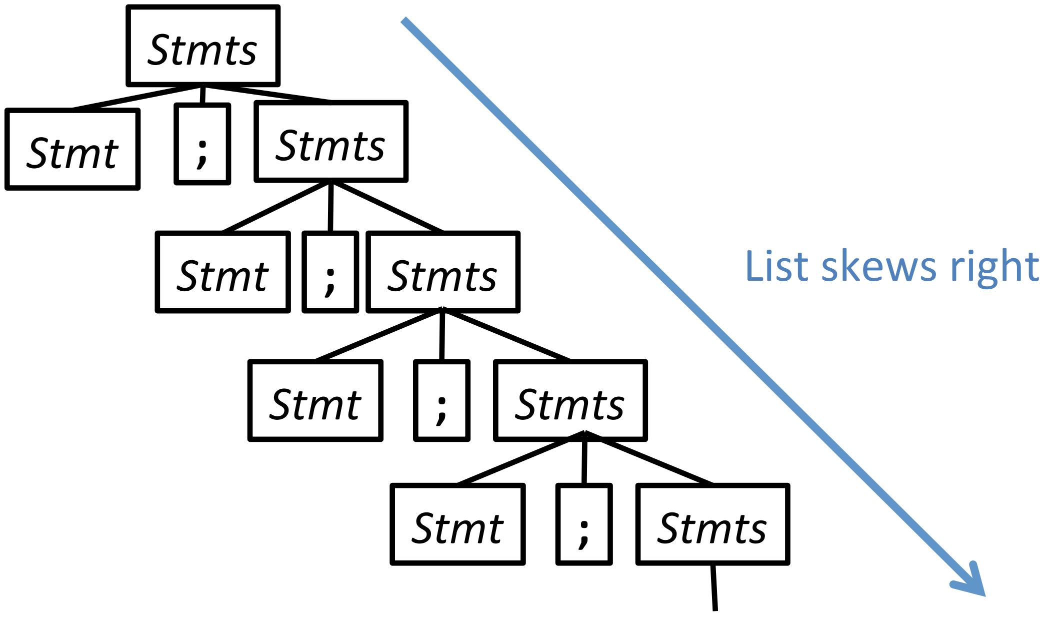
Stmts → *Stmts* **semicolon** *Stmt* | *Stmt*



List Grammars

- Useful to repeat a structure arbitrarily often

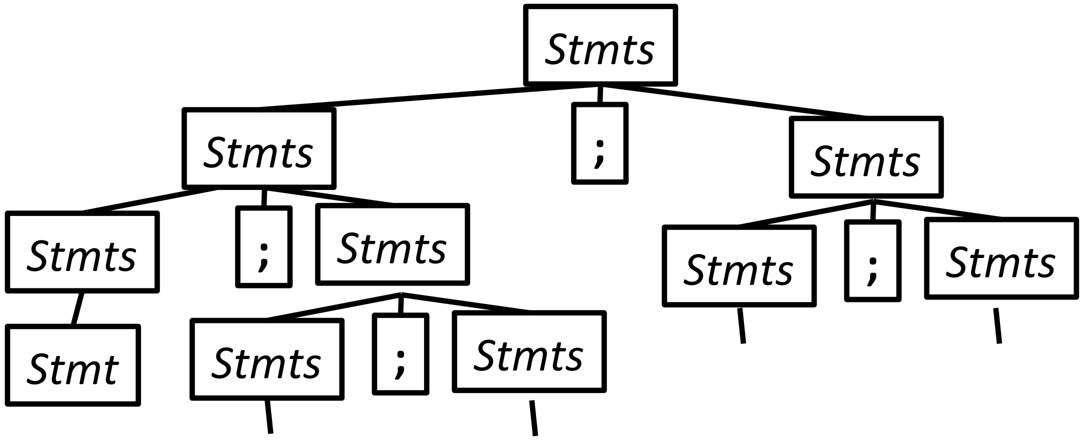
Stmts → *Stmt* **semicolon** *Stmts* | *Stmt*



List Grammars

- What if we allowed both "skews"?

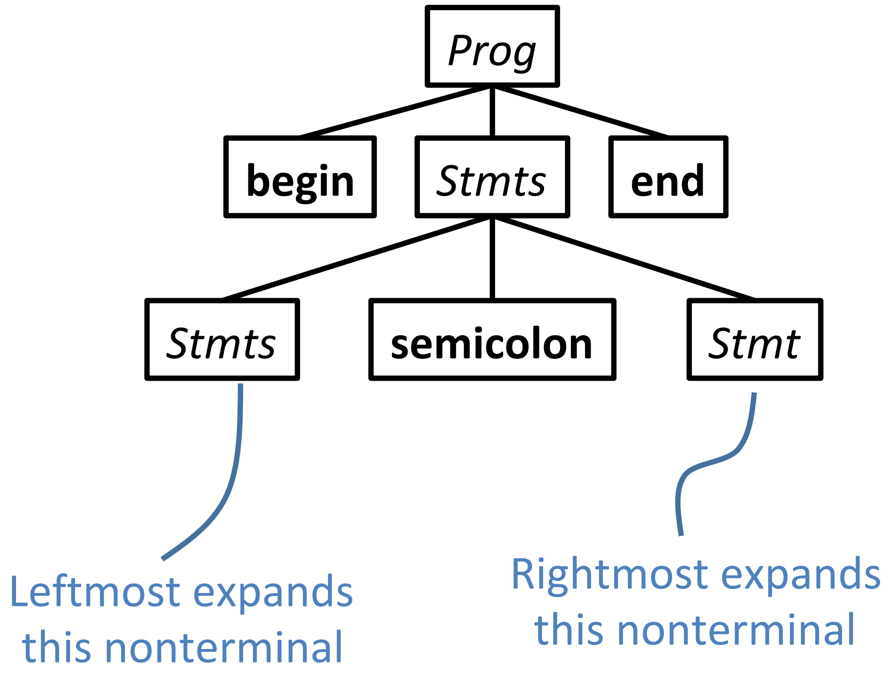
Stmts → *Stmts* **semicolon** *Stmts* | *Stmt*



Derivation Order

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

1. *Prog* → **begin** *Stmts* **end**
2. Stmts → *Stmts* **semicolon** *Stmt*
3. | *Stmt*
4. *Stmt* → **id assign** *Expr*
5. *Expr* → **id**
6. | *Expr* **plus id**


Prog
begin
Stmts
end
Stmts
semicolon
Stmt
Leftmost expands this nonterminal
Rightmost expands this nonterminal

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 → **intlit**

| *Expr* **minus** *Expr*

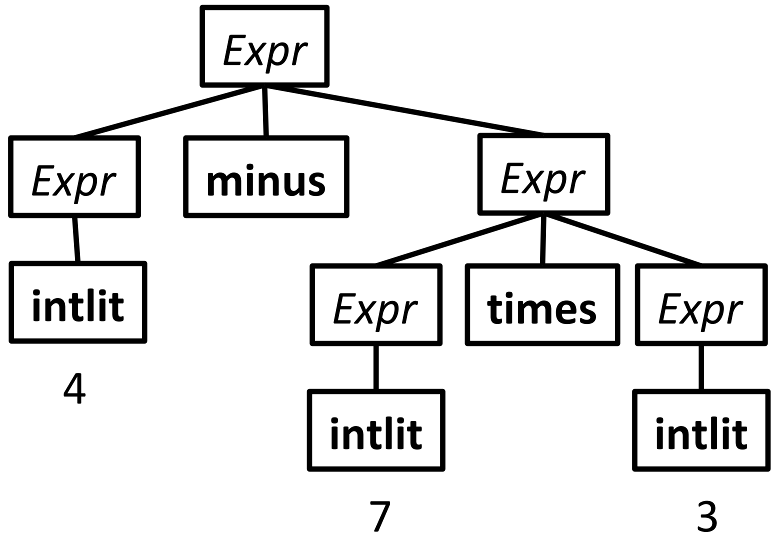
| *Expr* **times** *Expr*

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

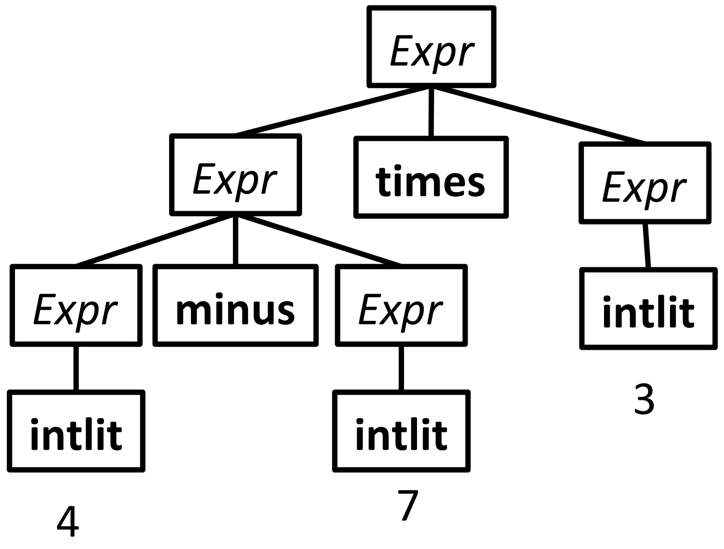
Derive the string 4 - 7 * 3

(assume tokenization)

Parse Tree 1



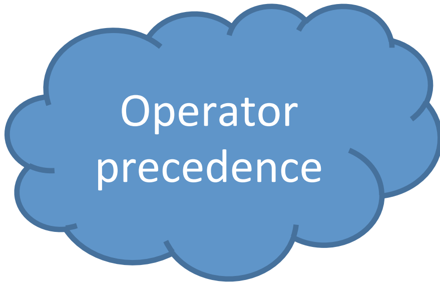
Parse Tree 2



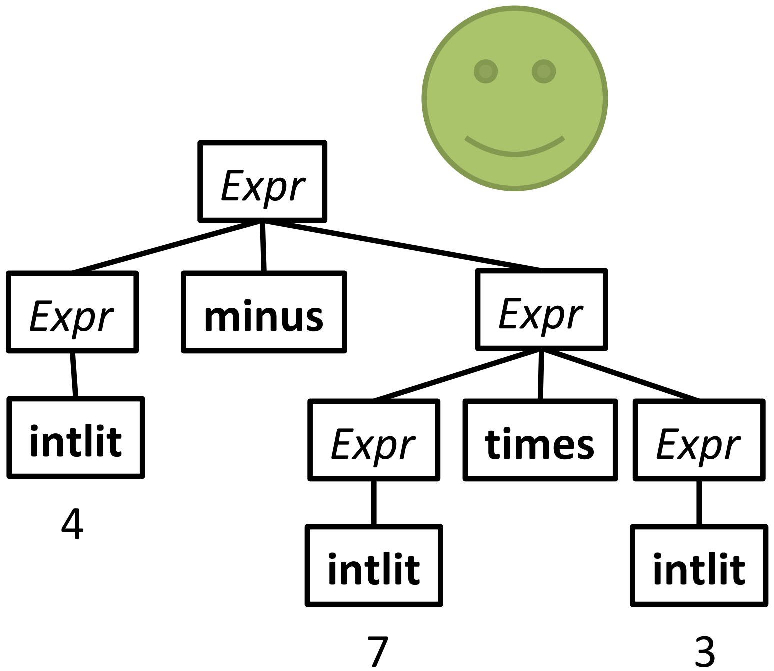
Why is Ambiguity Bad?

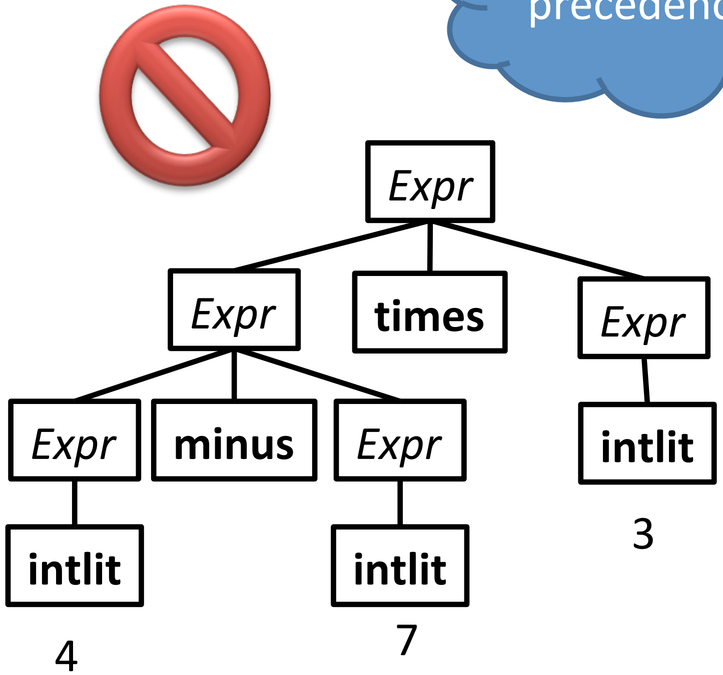
Eventually, we'll be using CFGs as the basis for our parser

- Parsing is much easier when there is no ambiguity in the grammar
- The parse tree may mismatch user understanding!



4 - 7 * 3





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 → **intlit**

| *Expr* **minus** *Expr*

| *Expr* **times** *Expr*

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

Expr → *Expr* **minus** *Expr*

| *Term*

Term → *Term* **times** *Term*

| *Factor*

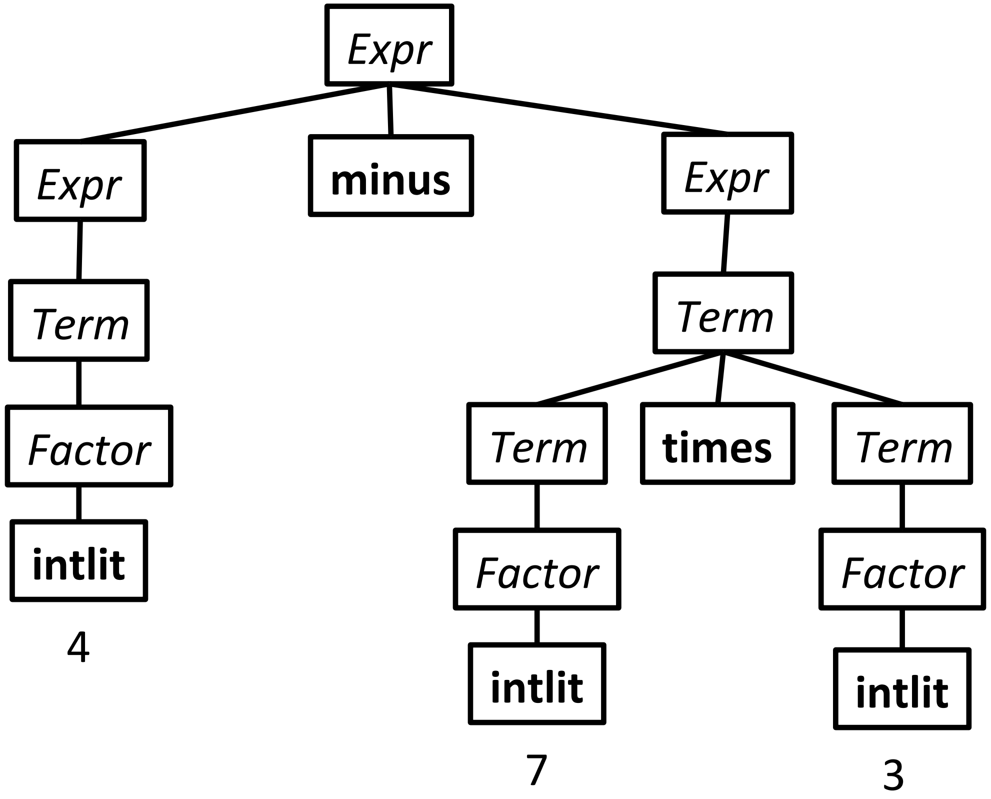
Factor → **intlit**

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

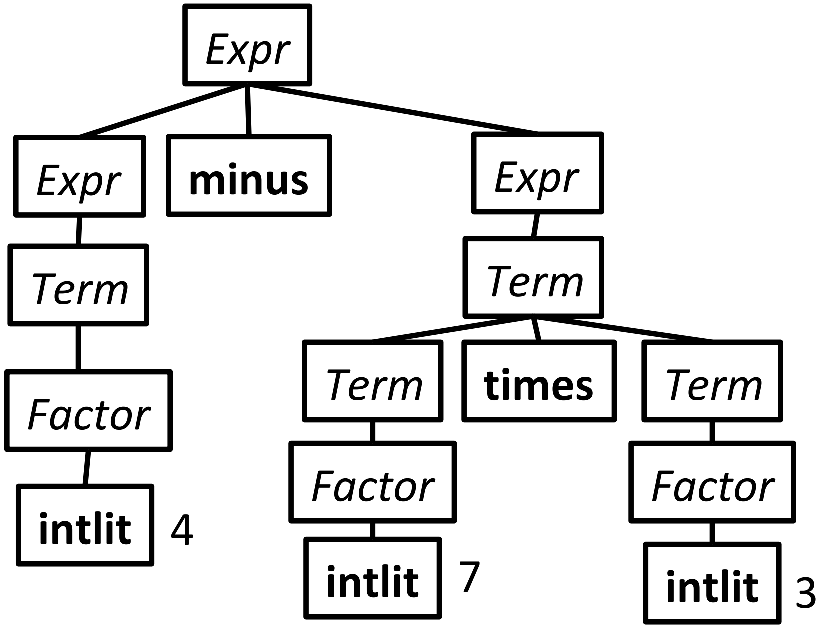
| *Factor*

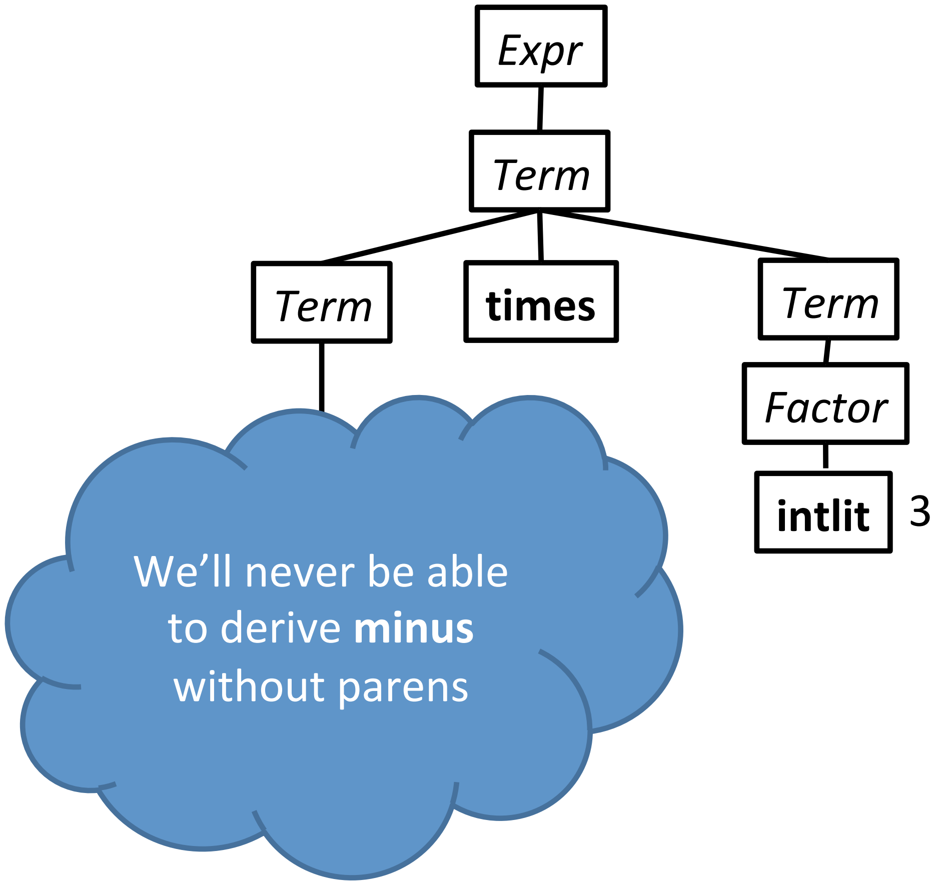
Factor → **intlit**

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

Derive the string 4 - 7 * 3

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





Did we fix all ambiguity?

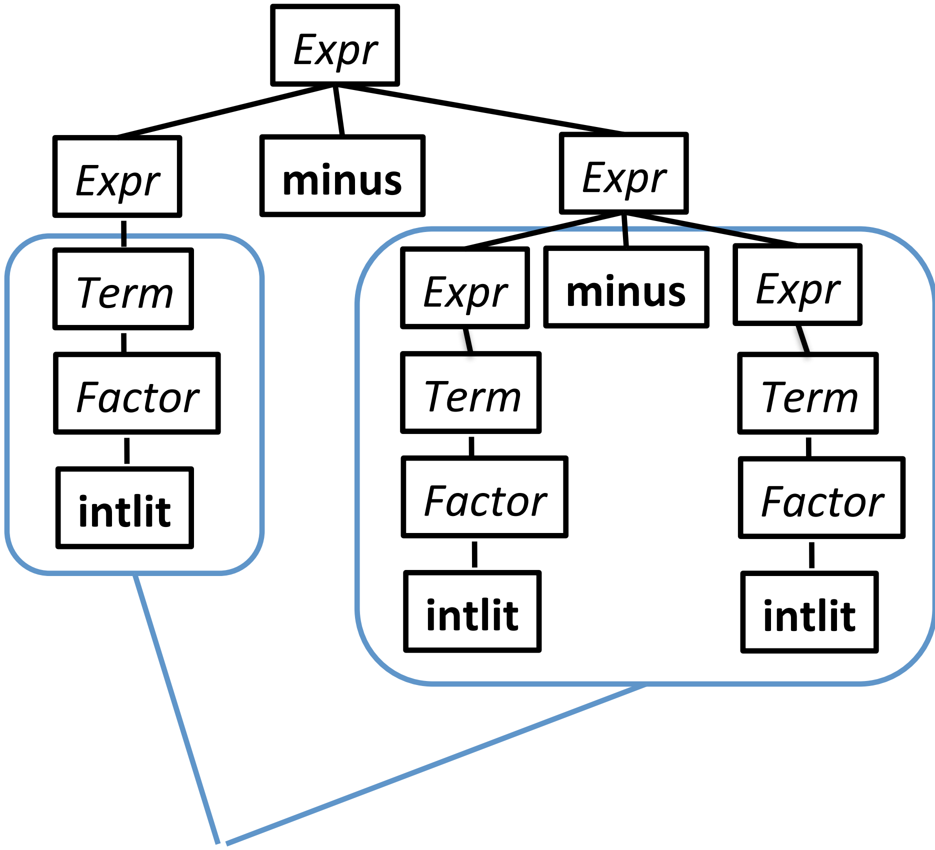
<u>Fixed Grammar</u>

Expr → *Expr* **minus** *Expr*
| *Term*

Term → *Term* **times** *Term*
| *Factor*

Factor → **intlit**
| **lparen** *Expr* **rparen**

Derive the string 4 - 7 - 3



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

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

Definition: Recursion in Grammars

- A grammar is *recursive* in (nonterminal) *X* if

  $X \Rightarrow^{+} \alpha X \gamma$ for non-empty strings of symbols α and γ
- A grammar is *left-recursive* in *X* if

  $X \Rightarrow^{+} X \gamma$ for non-empty string of symbols γ
- A grammar is *right-recursive* in *X* if

  $X \Rightarrow^{+} \alpha X$ for non-empty string of symbols α

Resolving Grammar Ambiguity: Associativity

Recognize left-assoc operators with left-associative productions

Recognize right-assoc operators with right-associative productions

Term

Expr → *Expr* **minus** ~~*Expr*~~

| *Term*

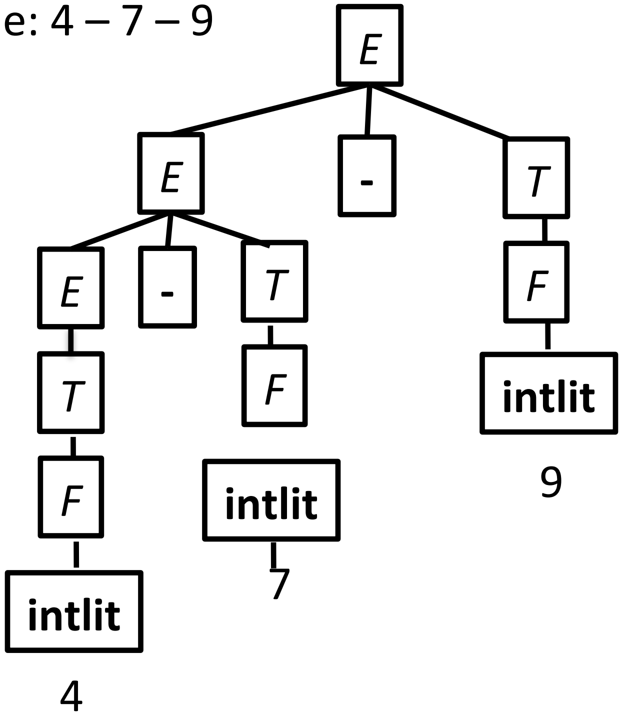
Factor

Term → *Term* **times** ~~*Term*~~

| *Factor*

Factor → **intlit** | **lparen** *Expr* **rparen**

Example: 4 – 7 – 9



Resolving Grammar Ambiguity: Associativity

Expr → *Expr* **minus** *Term*

| *Term*

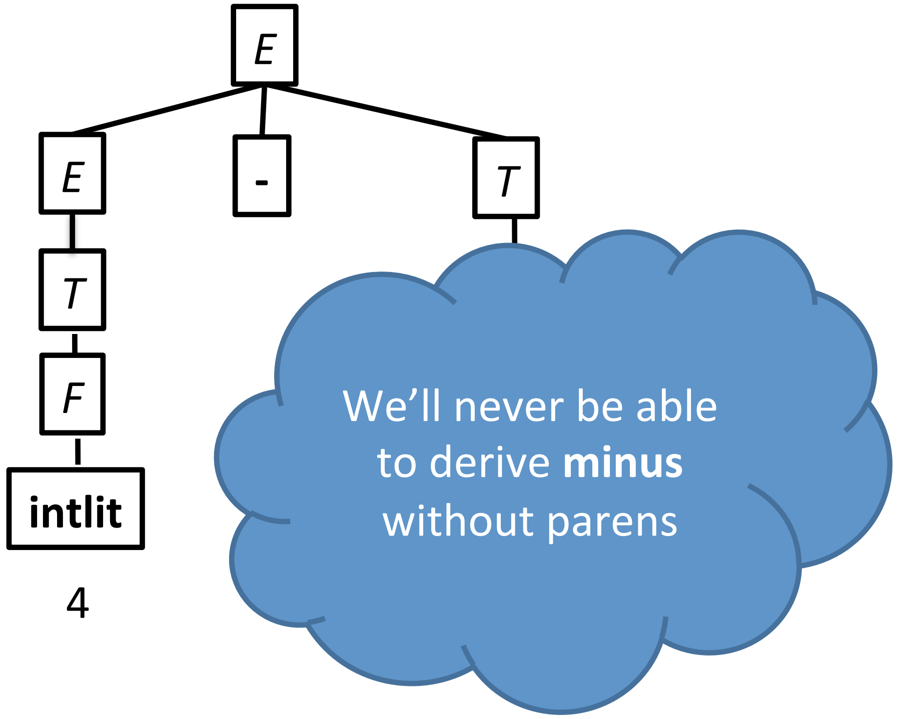
Term → *Term* **times** *Factor*

| *Factor*

Factor → **intlit** | **lparen** *Expr* **rparen**

Example: 4 – 7 – 9

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



Example

- Language of Boolean expressions

```
bexp → TRUE
     | FALSE
     | bexp OR bexp
     | bexp AND bexp
     | NOT bexp
     | 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