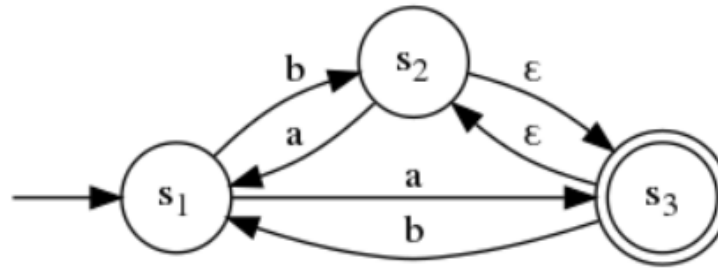


Check-in

Review: Tokenization

These questions use the following state machine:



1. Draw out an equivalent finite automaton to the given one after applying the ϵ -elimination technique described in class.
2. Is the state machine an NFA (non-deterministic finite automaton) or DFA (deterministic finite automaton)?
3. If the machine is a DFA, create an NFA that recognizes the same language. If the machine is an NFA, create a DFA that recognizes the same language.

Administrivia

Housekeeping

- Quiz 1 NEXT Friday
- Project 1 due **NEXT** Wednesday (or that Thursday for 1 late mark or that Friday for 2 late marks)

• • • The University of Kansas • • •

THE ASSOCIATION FOR COMPUTING MACHINERY

ACM @ KU

• • •
• • •
• • •
• • •
• • •

Join Our Discord



• • •
• • •
• • •
• • •
• • •

Visit Our Website



What we do

- ✓ Free Tutoring
- ✓ Company Speakers
- ✓ Career Fair Prep
- ✓ HACKKU
- ✓ Computer Science Talks

August 27th
6:00 - 7:00 PM
G415



Stay up to date

@acm.ku 
<https://kuacm.club> 

ECCS 665
COMPILER
CONSTRUCTION

The title is presented in a stylized, multi-line format. At the top, 'ECCS 665' is written in a pink, cursive font. Below it, the word 'COMPILER' is in large, bold, metallic 3D block letters with a blue and purple glow. Underneath 'COMPILER', the word 'CONSTRUCTION' is written in a white, italicized, sans-serif font with a blue outline. The background is a dark space scene with stars and a purple grid floor.

4 – Syntactic Ambiguity

I *Think* We're Mostly Done with Review

Lecture 4 – Syntactic Ambiguity

COMPILER

LAND

Code
Generation

Execution

Runtime
Environment

Intermediate
Representation

Optimization

SDD

Semantics

Parsing

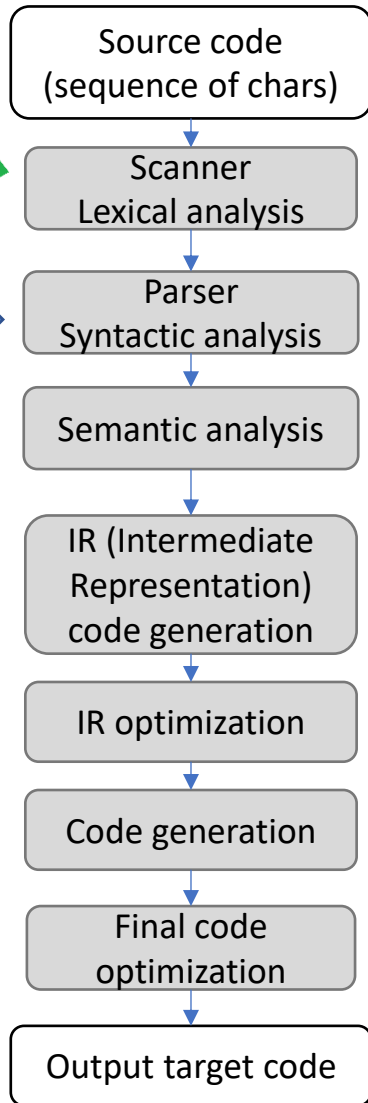
Lexical
Analysis

Syntactic
Definition



Compiler Construction

Progress Pics



Quick Comparison

Lexer **VS** Parser

Lexer Parser

Definition: RegExes

Input: char stream

Output: token stream

Recognizer: DFA

Definition: CF Grammar

Input: token stream

Output: parse tree

Recognizer: PDA



**Let's breeze through this
(I bet you've seen it already)**

Lexer Parser

Definition: RegExes

Input: char stream

Output: token stream

Recognizer: DFA

Translator: Tokenizer

Definition: CF Grammar

Input: token stream

Output: parse tree

Recognizer: PDA

Translator: Parser

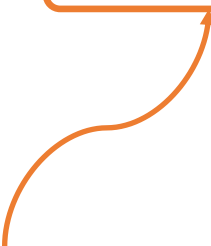
Last Time

Review Lecture 3 – Defining Syntax

Reviewed Context-Free Grammars

- Language Definition via grammar
- BNF notation
- Parse trees

CFGs - usable for syntactic **definition**



*BNF is a good way for
programmers to
communicate syntax*

Derivation Sequence

Prog

Prod. 1

⇒ begin Stmts end

Prod. 2

⇒ begin Stmts semi Stmt end

Prod. 3

⇒ begin Stmt semi Stmt end

Prod. 4

⇒ begin id assign Expr semi Stmt end

Prod. 4

⇒ begin id assign Expr semi id assign Expr end

Prod. 5

⇒ begin id assign id semi id assign Expr end

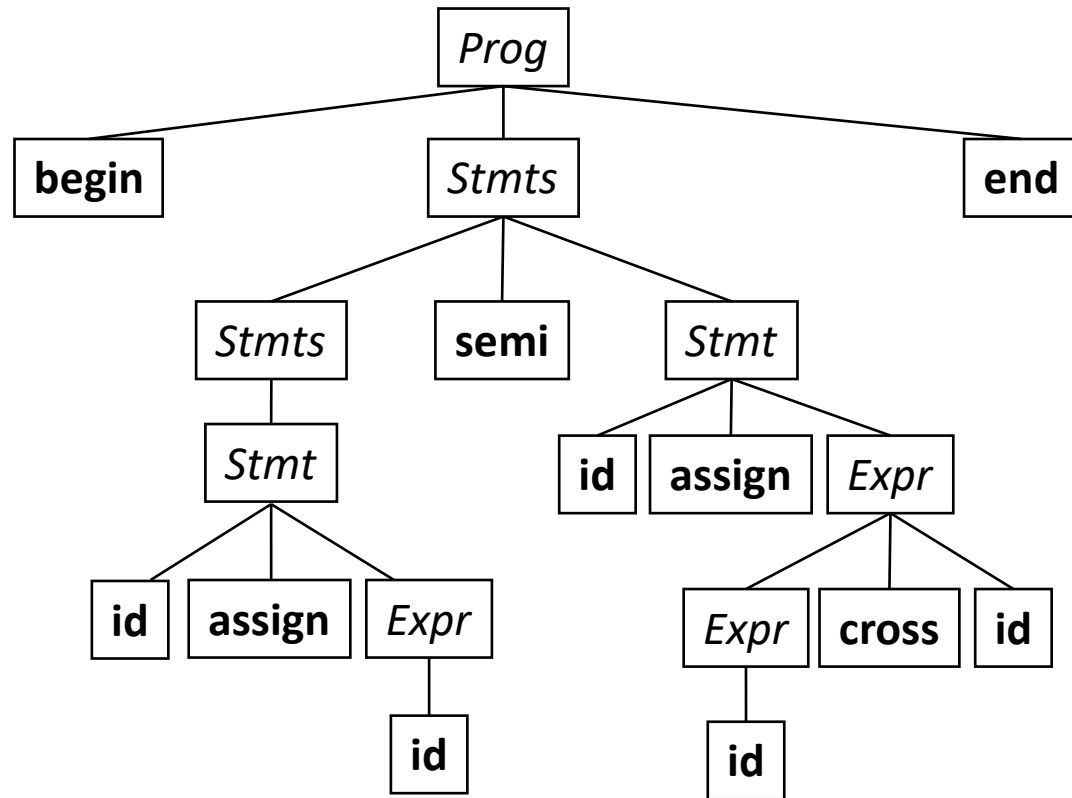
Prod. 6

⇒ begin id assign id semi id assign Expr cross id end

Prod. 5

⇒ begin id assign id semi id assign id cross id end

Parse Tree



Productions

1. $Prog ::= \text{begin } Stmts \text{ end}$
2. $Stmts ::= Stmts \text{ semi } Stmt$
3. $\quad \quad \quad | \text{ Stmt}$
4. $Stmt ::= \text{id assign Expr}$
5. $Expr ::= \text{id}$
6. $\quad \quad \quad | \text{ Expr cross id}$

Lecture Outline

Lecture 4 –Syntactic Ambiguity

Compiler Concerns in Syntactic Analysis

- Recognizing CF-Languages
- Constraining Derivations
- Ambiguous Syntax
 - What it is
 - How we resolve it

Constraining Derivations

Lecture 4 –Syntactic Ambiguity

Goal: 1-1 map derivation sequences to sentences

- Reason #1: predictable, reliable algorithms
- Reason #2: we care about the structure of sentences



Consistency is key

Derivation Order

Syntactic Ambiguity: Constraining Derivations

Grammar

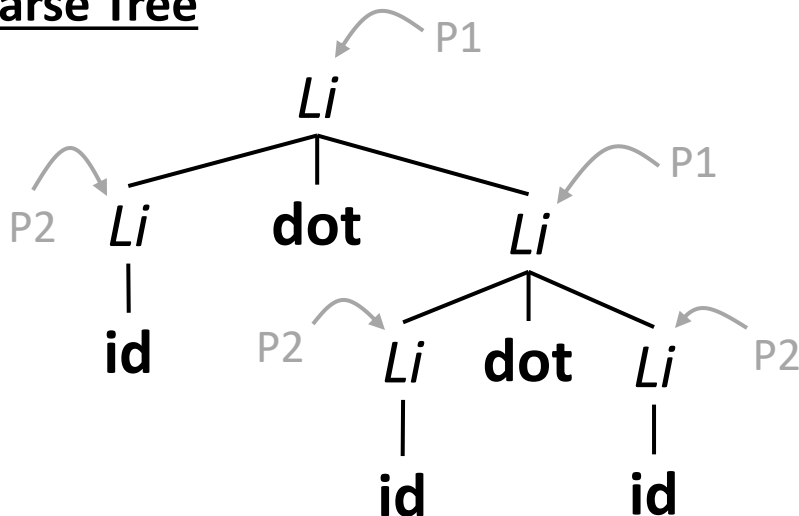
P1: $Li ::= Li \text{ dot } Li$

P2: $\mid id$

Token stream

id dot id dot id

Parse Tree



How can we derive the sentence?

- Multiple orderings of the same productions to generate sentence
 - That's bad for consistency!
- We usually commit to a single ordering
 - *Leftmost derivation*: expand the leftmost nonterminal
 - *Rightmost derivation*: expand the rightmost nonterminal

Derivation Order

Syntactic Ambiguity: Constraining Derivations

Grammar

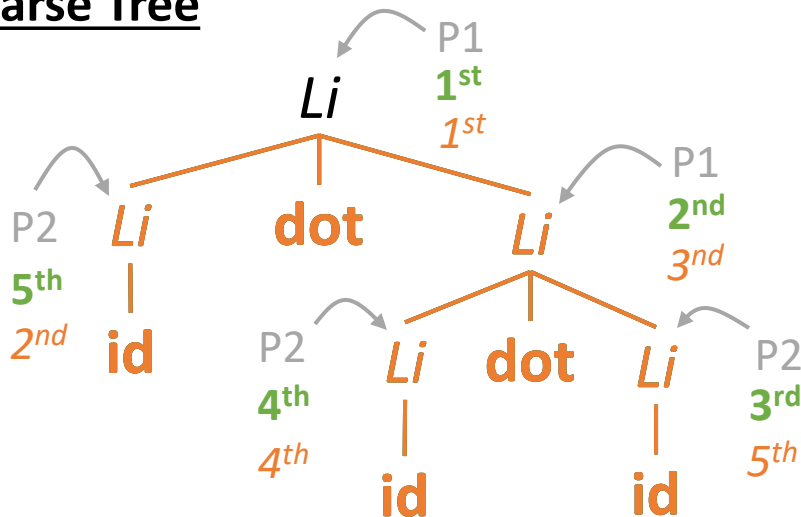
P1: $Li ::= Li \text{ dot } Li$

P2: $\quad | \text{ id}$

Token stream

id dot id dot id

Parse Tree



How can we derive the sentence?

- Multiple orderings of the same productions to generate sentence
 - That's bad for consistency!
- We usually commit to a single ordering
 - *Leftmost derivation*: expand the leftmost nonterminal
 - *Rightmost derivation*: expand the rightmost nonterminal

Leftmost derivation

$Li \Rightarrow Li \text{ dot } Li$
 $\Rightarrow id \text{ dot } Li$
 $\Rightarrow id \text{ dot } Li \text{ dot } Li$
 $\Rightarrow id \text{ dot } id \text{ dot } Li$
 $\Rightarrow id \text{ dot } id \text{ dot } id$

Rightmost derivation

$Li \Rightarrow Li \text{ dot } Li$
 $\Rightarrow Li \text{ dot } Li \text{ dot } Li$
 $\Rightarrow Li \text{ dot } Li \text{ dot } id$
 $\Rightarrow Li \text{ dot } id \text{ dot } id$
 $\Rightarrow id \text{ dot } id \text{ dot } id$

Bigger Problems On the Horizon

Syntactic Ambiguity: Constraining Derivations



**May be able to produce different parse trees
even under the same derivation order!**

The Real Problem

Syntactic Ambiguity: Constraining Derivations

Grammar

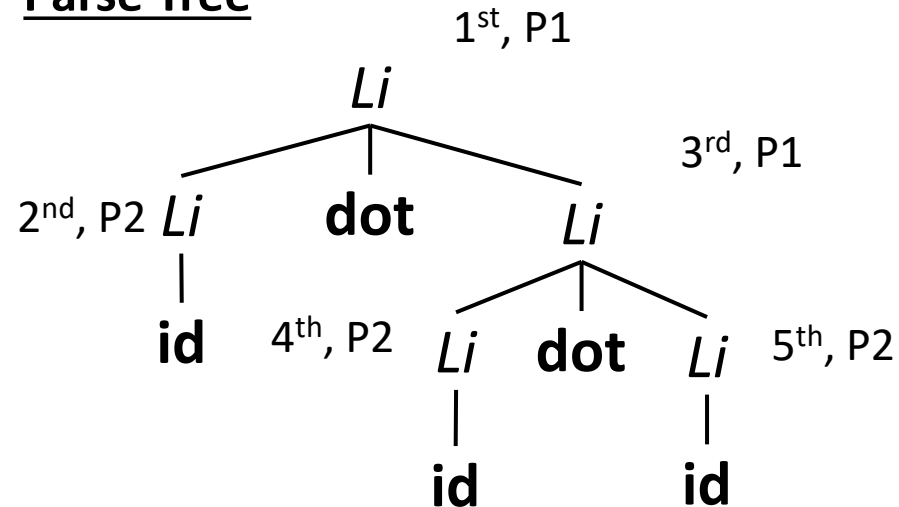
P1. $Li ::= Li \text{ dot } Li$

P2. $\mid id$

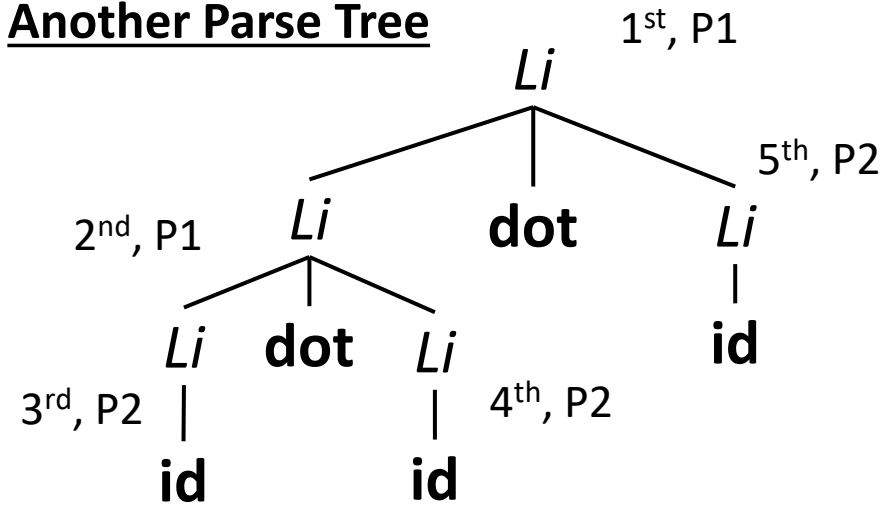
Input

id dot id dot id

Parse Tree



Another Parse Tree



Same (leftmost) derivation order, different productions

The Real Problem

Syntactic Ambiguity: Constraining Derivations

- Even with a fixed derivation order, possible to derive the same sentence in multiple ways

Syntactic Ambiguity (Definition):

For Grammar G and string w

- G is ambiguous if
 - >1 leftmost derivation of w
 $\equiv$
>1 rightmost derivation of w
 $\equiv$
> 1 parse tree for w

Why is syntactic ambiguity a problem?

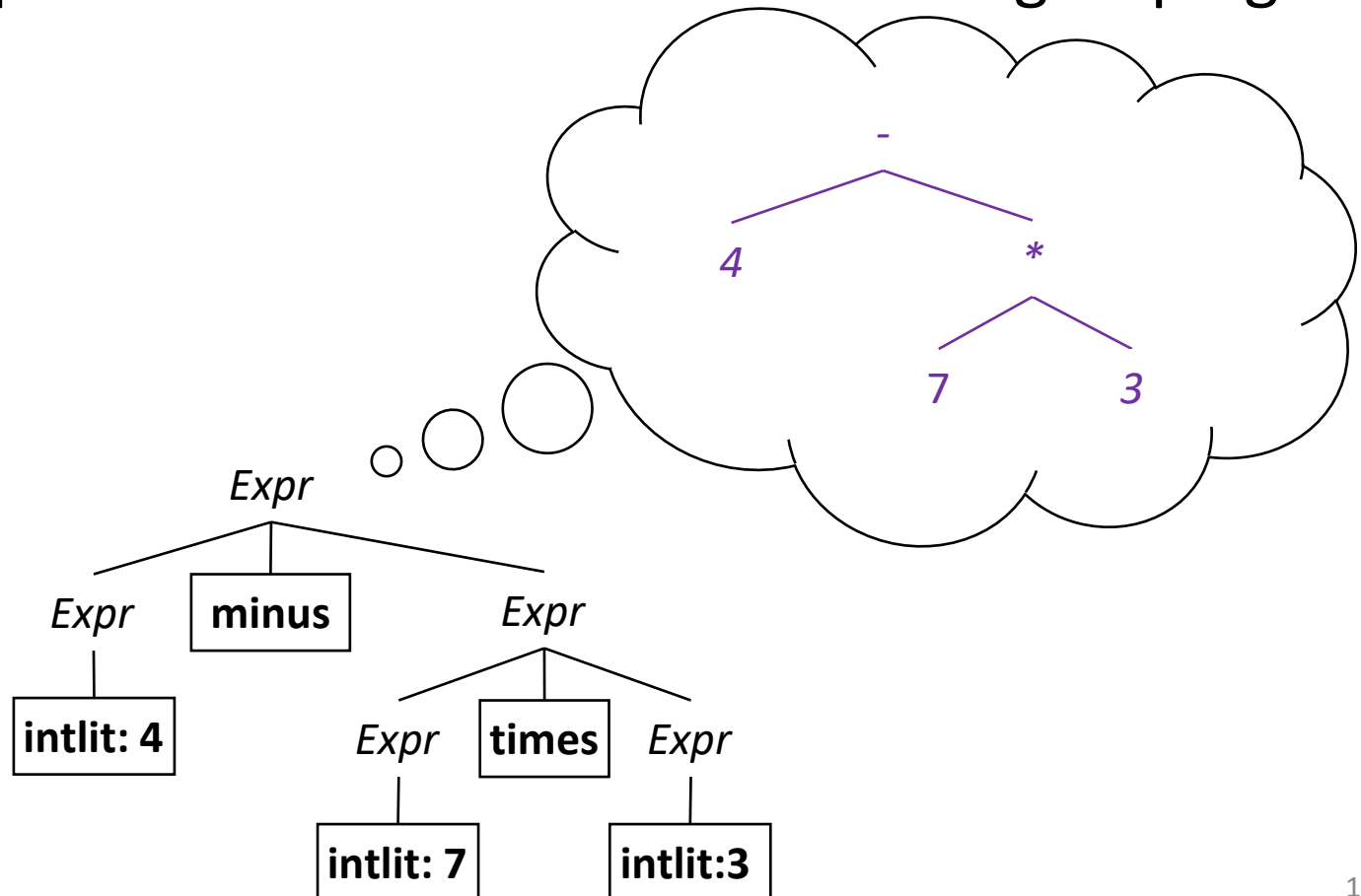
- Allows inconsistency
- Unreliable tree structure

Particularly troubling

Parse Tree Structure

Syntactic Ambiguity: Constraining Derivations

- Maintain the integrity of the underlying operations
- Intuitively, parse tree should mimic semantic grouping



Parse Tree Structure

Syntactic Ambiguity: Constraining Derivations

Grammar

- P1. *Expr* ::= **intlit**
P2. | *Expr* **minus** *Expr*
P3. | *Expr* **times** *Expr*

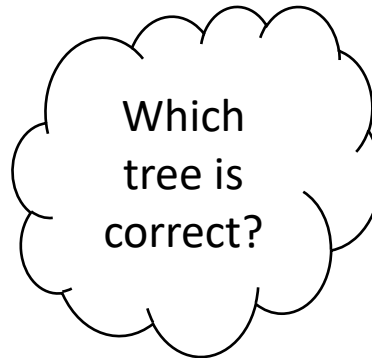
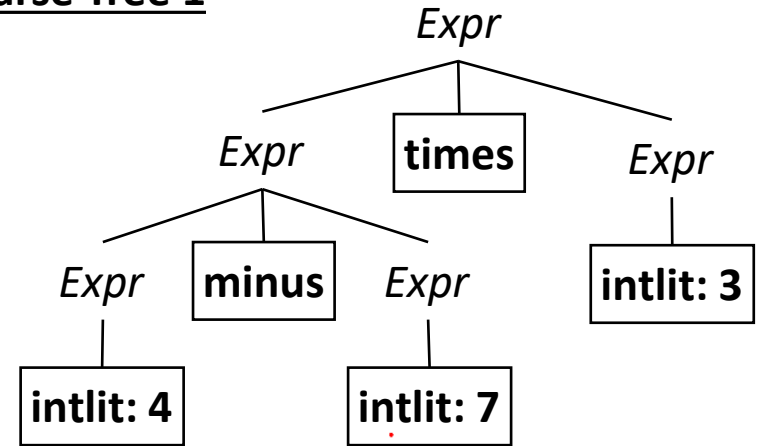
Char Stream:

4 – 7 * 3

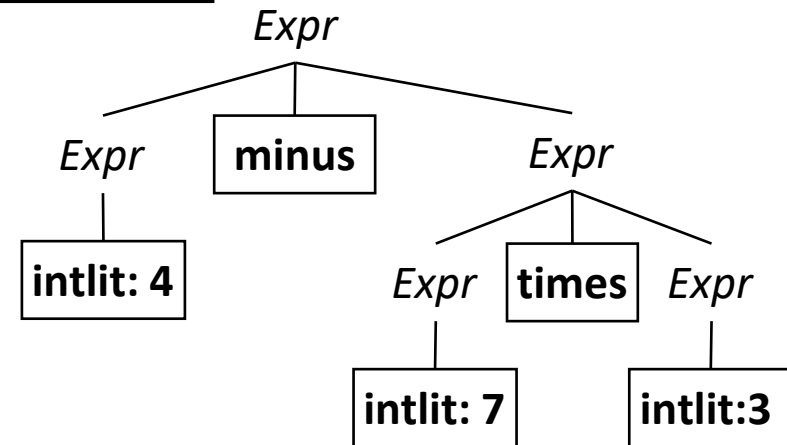
Token Stream:

intlit minus intlit times intlit

Parse Tree 1



Parse Tree 2



Parse Tree Structure

Syntactic Ambiguity: Constraining Derivations

Grammar

P1. $Expr ::= intlit$
P2. $| Expr \text{ times } Expr$
P3. $| Expr \text{ minus } Expr$

Char Stream:

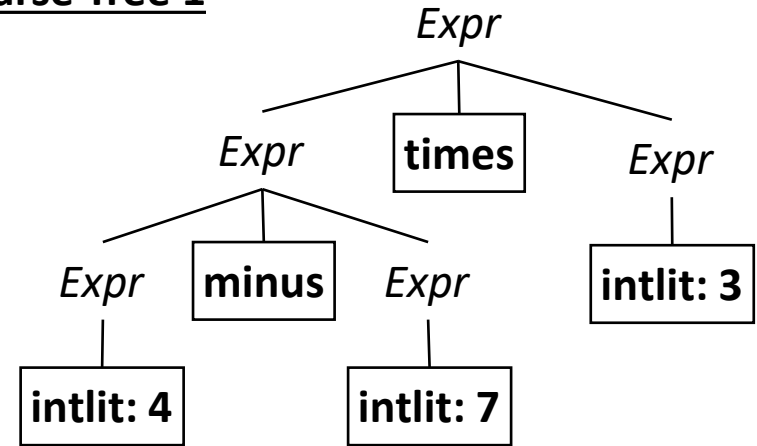
$(4 - 7) * 3$

Token Stream:

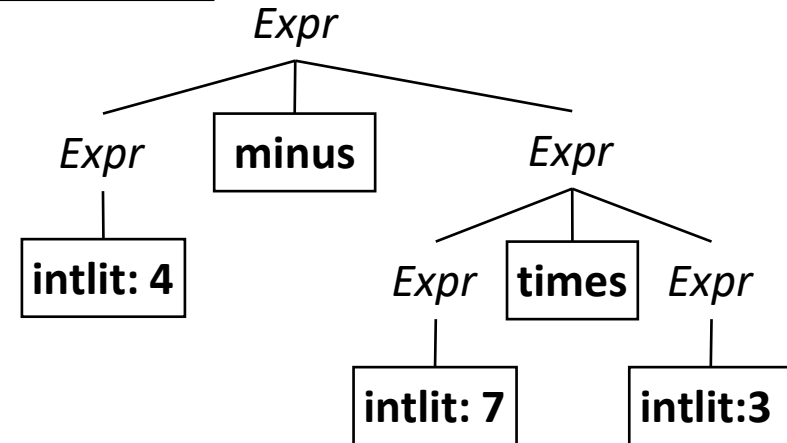
intlit minus intlit



Parse Tree 1



Parse Tree 2



Parse Tree Structure

Syntactic Ambiguity: Constraining Derivations

Grammar

- P1. $Expr ::= \text{intlit}$
P2. $| Expr \text{ minus } Expr$
P3. $| Expr \text{ times } Expr$

Char Stream:

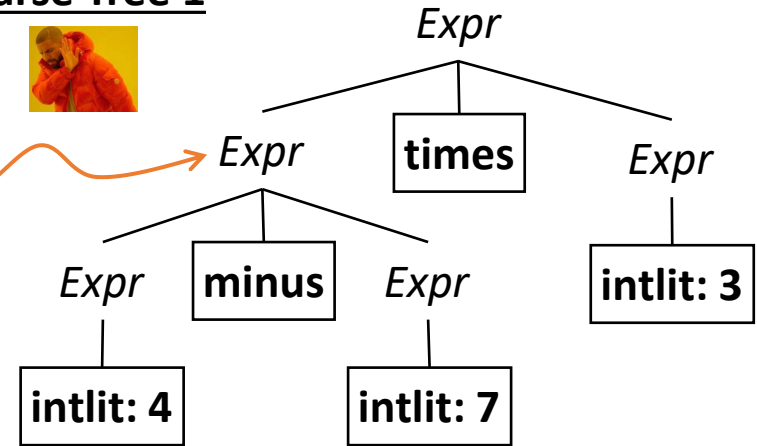
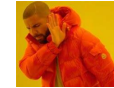
4 - 7 * 3

Token Stream:

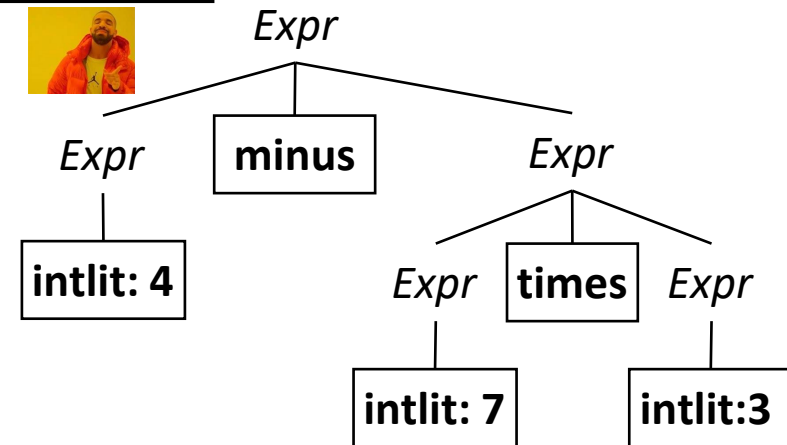
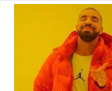
intlit minus intlit times intlit

Intuitive problem: Depth
Minus subtrees
(can be) produced
beneath times subtrees

Parse Tree 1



Parse Tree 2



Parse Tree Structure

Syntactic Ambiguity: Constraining Derivations

Grammar

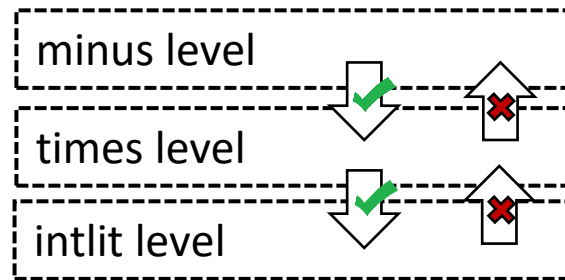
- P1. $Expr ::= \text{intlit}$
P2. | $Expr \text{ minus } Expr$
P3. | $Expr \text{ times } Expr$

*Intuitive problem: Depth
Minus subtrees
(can be) produced
beneath times subtrees*

Fixed Grammar

- P1. $Expr ::= Expr \text{ minus } Expr$
P2. | $Term$
P3. $Term ::= Term \text{ times } Term$
P4. | $Factor$
P5. $Factor ::= \text{intlit}$

Solution: Embed precedence “levels” into the grammar



- You can go down a level “for free”
- You can’t go up a level

Use a new nonterminal for each precedence level

Parse Tree Structure

Syntactic Ambiguity: Constraining Derivations

Fixed Grammar

- P1. $Expr ::= Expr \textbf{minus} Expr$
P2. | $Term$
P3. $Term ::= Term \textbf{times} Term$
P4. | $Factor$
P5. $Factor ::= \textbf{intlit}$

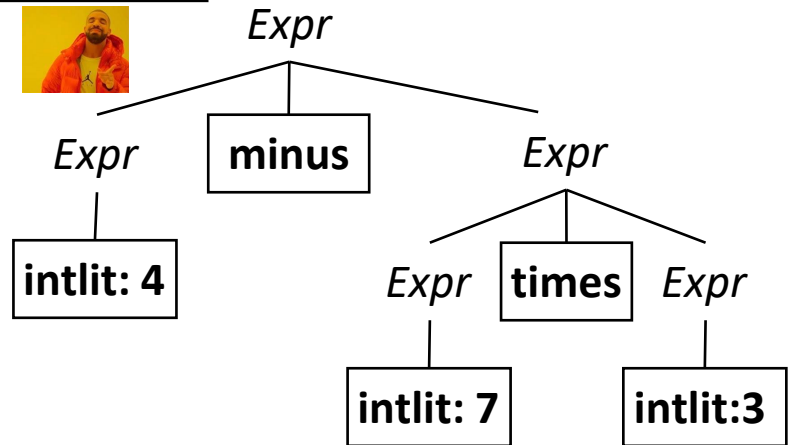
Char Stream:

4 – 7 * 3

Token Stream:

intlit minus intlit times intlit

Parse Tree 2



Parse Tree Structure

Syntactic Ambiguity: Constraining Derivations

Fixed Grammar

- P1. $Expr ::= Expr \text{ minus } Expr$
P2. | $Term$
P3. $Term ::= Term \text{ times } Term$
P4. | $Factor$
P5. $Factor ::= \text{intlit}$

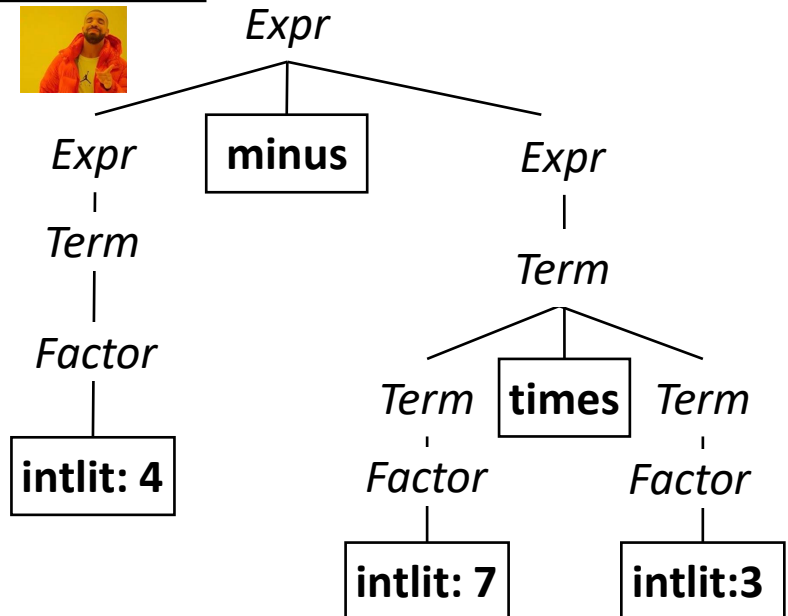
Char Stream:

4 – 7 * 3

Token Stream:

intlit minus intlit times intlit

Parse Tree 2



Parse Tree Structure

Syntactic Ambiguity: Constraining Derivations

Fixed Grammar

- P1. $Expr ::= Expr \textbf{minus} Expr$
P2. | $Term$
P3. $Term ::= Term \textbf{times} Term$
P4. | $Factor$
P5. $Factor ::= \textbf{intlit}$

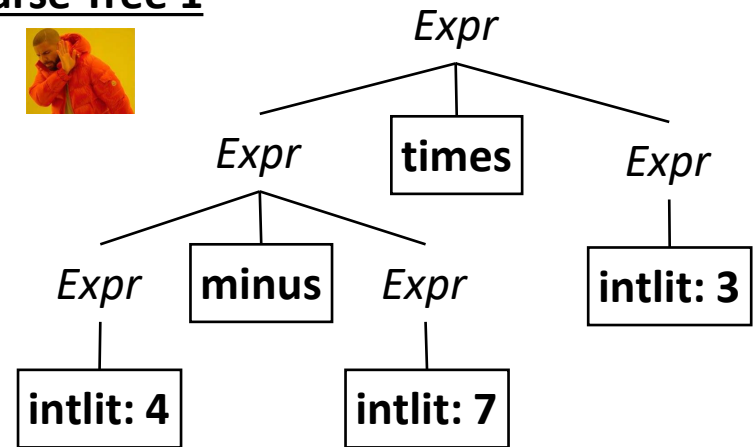
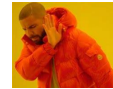
Char Stream:

4 – 7 * 3

Token Stream:

intlit minus intlit times intlit

Parse Tree 1



Parse Tree Structure

Syntactic Ambiguity: Constraining Derivations

Fixed Grammar

- P1. $Expr ::= Expr \text{ minus } Expr$
P2. | $Term$
P3. $Term ::= Term \text{ times } Term$
P4. | $Factor$
P5. $Factor ::= \text{intlit}$

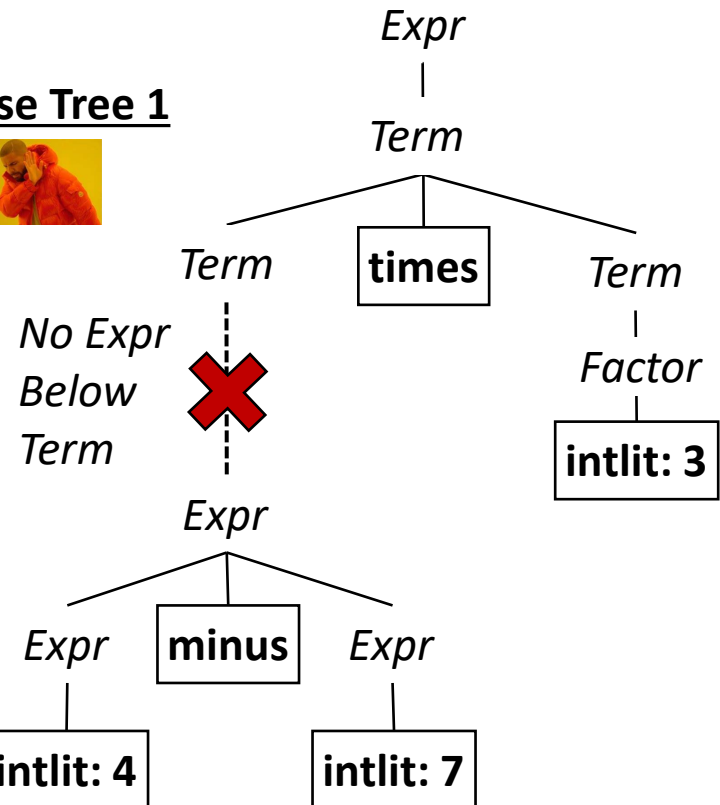
Char Stream:

4 – 7 * 3

Token Stream:

intlit minus intlit times intlit

Parse Tree 1



Did We Fix All Ambiguity?

Resolving Syntactic Ambiguity

Fixed Grammar

- P1. $Expr ::= Expr \text{ minus } Expr$
P2. $\quad \quad | Term$
P3. $Term ::= Term \text{ times } Term$
P4. $\quad \quad | Factor$
P5. $Factor ::= \text{intlit}$

Nope!

Grammar
is still
ambiguous!

Char Stream:

3 – 4 – 7



Did We Fix All Ambiguity?

Resolving Syntactic Ambiguity

Fixed Grammar

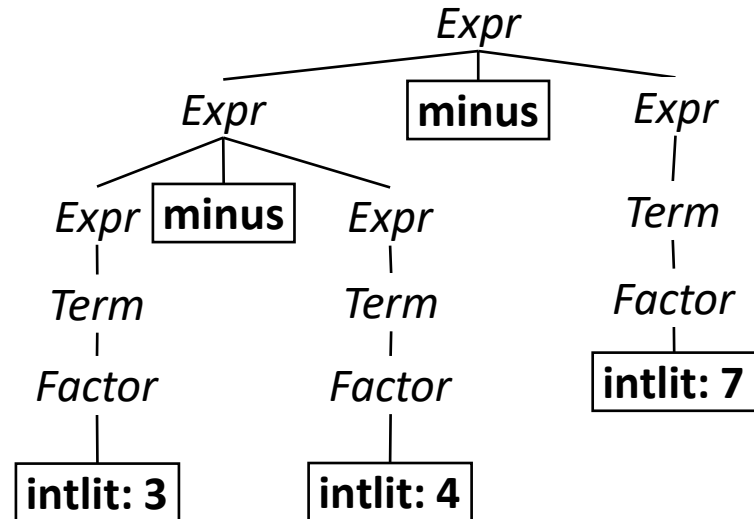
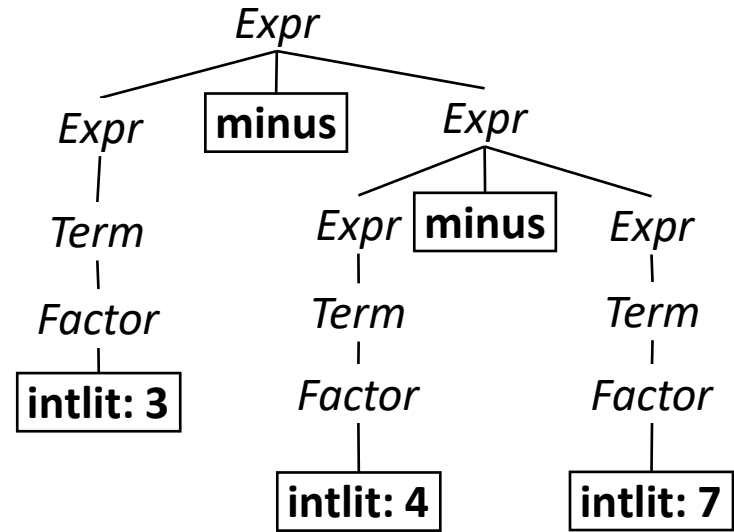
- P1. $Expr ::= Expr \text{ minus } Expr$
P2. | $Term$
P3. $Term ::= Term \text{ times } Term$
P4. | $Factor$
P5. $Factor ::= \text{intlit}$

Nope!

Grammar
is still
ambiguous!

Char Stream:

3 – 4 – 7



A Second Ambiguity Issue

Syntactic Ambiguity: Constraining Parse Trees

Precedence

Solved!
enforce depth with
nonterminal “levels”

- We want correct behavior on

$4 - 7 * 9$

- High-precedence operators
deeper in the tree

Associativity

Idea...
enforce with
production “skew”

- We want correct behavior on

$4 - 7 - 9$

- Minus should be *left associative*:

$a - b - c - d = ((a - b) - c) - d$

A Second Ambiguity Issue

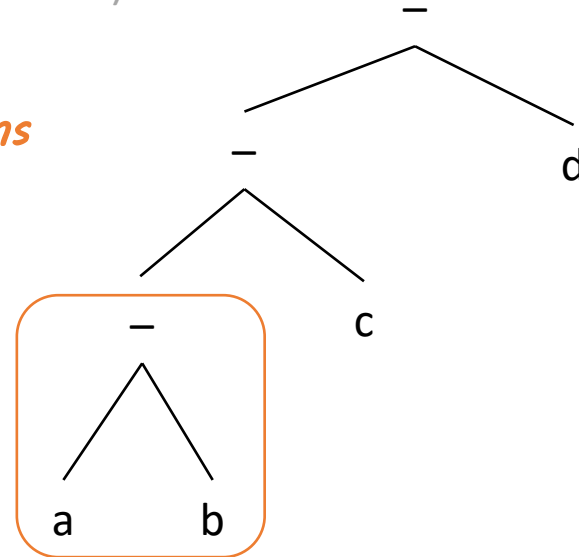
Syntactic Ambiguity: Constraining Parse Trees

Associativity

- We want correct behavior on
4 - 7 - 9
- Minus should be *left associative*:

$$a - b - c - d = ((a - b) - c) - d$$

Deeply nested parens
=
Deep in the tree



Idea...
enforce with
production "skew"

Enforcing Associativity

Resolving Syntactic Ambiguity

- Recognize left-associative operators with *left-recursive* productions
- Recognize right-associative operators with *right-recursive* productions
- **A grammar is *recursive* in (nonterminal) X if**
$$X \stackrel{+}{\Rightarrow} \alpha X \gamma \text{ for arbitrary strings of symbols } \alpha \text{ and } \gamma$$

Enforcing Associativity

Resolving Syntactic Ambiguity

- Recognize left-associative operators with *left-recursive* productions
- Recognize right-associative operators with *right-recursive* productions
- **A grammar is *recursive* in (nonterminal) X if**
$$X \xRightarrow{+} \alpha X \gamma \text{ for arbitrary strings of symbols } \alpha \text{ and } \gamma$$
- **A grammar is *left-recursive* in X if**
$$X \xRightarrow{+} X \gamma \text{ for non-empty string of symbols } \gamma$$

Enforcing Associativity

Resolving Syntactic Ambiguity

- Recognize left-associative operators with *left-recursive* productions
- Recognize right-associative operators with *right-recursive* productions
- **A grammar is *recursive* in (nonterminal) X if**
$$X \xRightarrow{+} \alpha X \gamma \text{ for arbitrary strings of symbols } \alpha \text{ and } \gamma$$
- **A grammar is *left-recursive* in X if**
$$X \xRightarrow{+} X \gamma \text{ for non-empty string of symbols } \gamma$$
- **A grammar is *right-recursive* in X if**
$$X \xRightarrow{+} \alpha X \text{ for non-empty string of symbols } \alpha$$

Enforcing Associativity

Resolving Syntactic Ambiguity

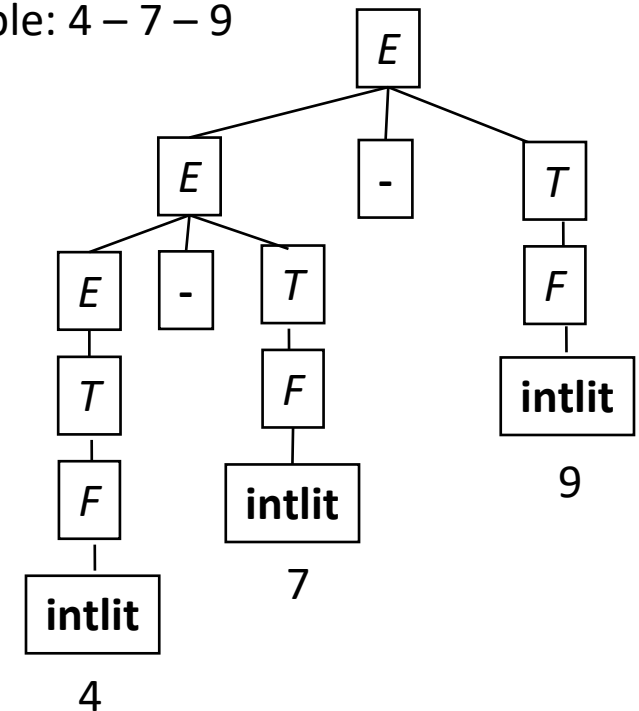
- Recognize left-associative operators with *left-recursive* productions
- Recognize right-associative operators with *right-recursive* productions

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

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

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

Example: 4 - 7 - 9



Enforcing Associativity

Resolving Syntactic Ambiguity

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

| $Term$

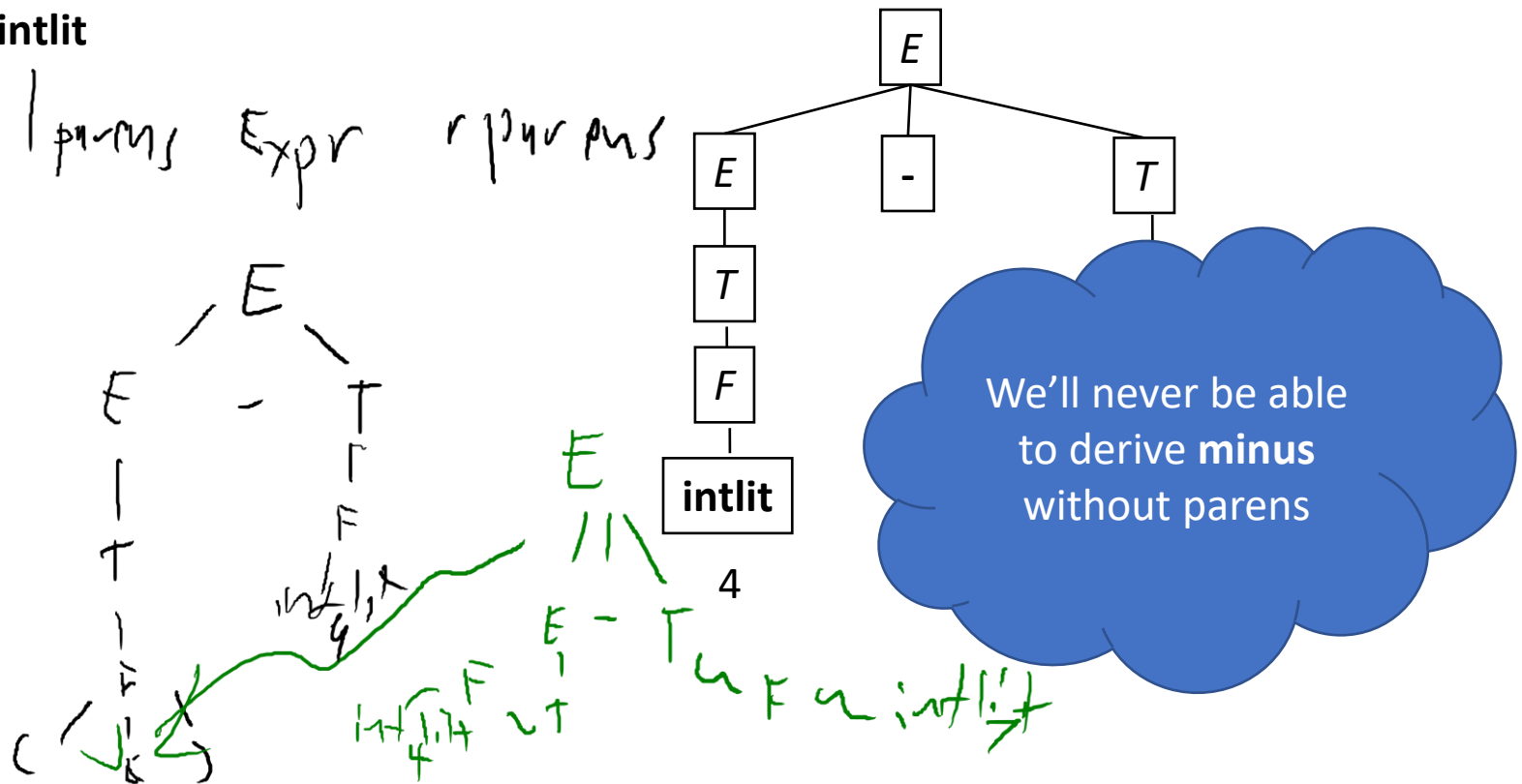
$Term \rightarrow Term \text{ times } Factor$

| $Factor$

$Factor \rightarrow \text{intlit}$

Example $\{4 - 7\} - 9$

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



End of Lecture

Syntactic Ambiguity

Summary

- We set out to make derivations and parse trees map 1-1 with any string in a grammar
- We note several forms of ambiguity and consider fixes:
 - Derivation order: use a fixed strategy
 - Syntactic ambiguity (precedence) – “stratify” the grammar
 - Syntactic ambiguity (associativity) – “lean” the grammar

Next Time

Parse tree translation: ascribing meaning to parse trees

