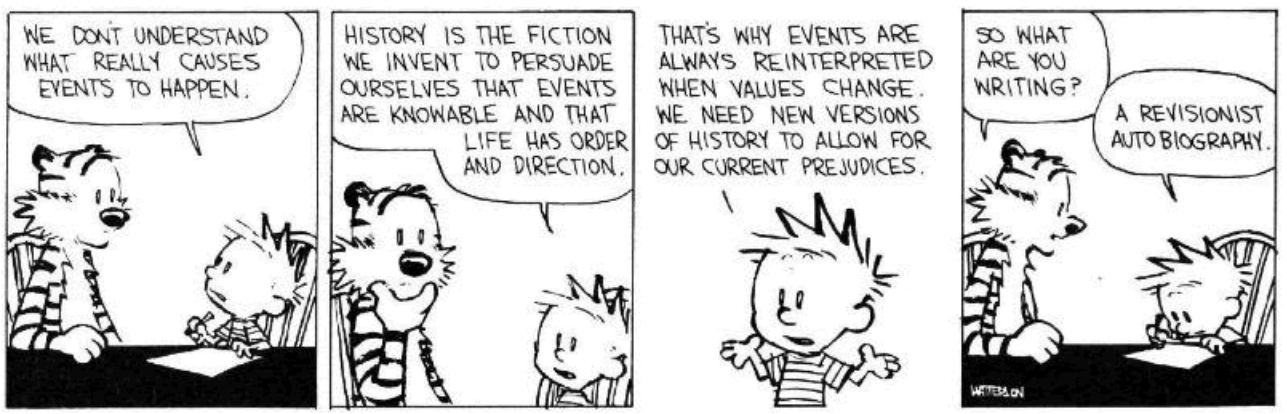
Meeting 07 - Concrete Syntax

Bor-Yuh Evan Chang

Tuesday, September 17, 2024

Meeting 07 - Concrete Syntax



What questions does your neighbor have?

In-Class Slides

Book Chapter

Announcements

- HW 2 due this ~~Friday~~ Monday 9/20 6pm
  - Jupyter assignment
- My OH → tomorrow (Wed) 3:30 – 4:30 CSEL
- Demo on coding.csel.io
- Grading update — Graders hired!

Today

- Preview HW 2
- Concrete Syntax
- Triage Your Questions
  - Lab 1?
- Revisit and Go Deeper On:
  - Inductive Data Types (Meeting 06), if time permits

Questions?

- Review:
  - What's *an inductive data type*?

- Recursive Data Types / Stack Frames
- Tail Recursion + Tree Problems
- Eval — What's the point

Questions?

What is the purpose of a programming language specification?

Who uses a language specification?

parser — language implementer (Olivia)

textfile? — language programmer / client / user (Oscar)

"mathy" grammars — language designer

linter — language checker

What is the difference between syntax and semantics?

Can a given syntax have two different semantics (in different languages)?

→ syntax — how you write

semantics — how code gets executed

What is the difference between concrete and abstract syntax?

concrete — lower-level
— string

parsing

abstract — important things only
— tree

Concrete Syntax

Formal Language Terminology

In formal language theory, a *language* L is ...

a string

L a set of strings

'aa' $\in L$

Formal Language Terminology

A *sentence* is …

a string is in a given language

Grammars

A (context-free) grammar describes …

languages — context free languages

BNF (Backus-Naur Form) is …

particular notation of grammars

Grammars

A grammar consists *terminals*, *non-terminals*, and *productions*:

$$N ::= \alpha_1 \mid \alpha_2 \mid \cdots$$

alphabet = terminals

where $N \in \mathcal{N}$ is a non-terminal and $\alpha \in (\Sigma \cup \mathcal{N})^*$.

Lexical versus Syntactic

A *lexeme* is …

num('134')

A *token* is …

= terminal that consists of a string

Generators versus Recognizers

Grammar "(. ::= . | .)*"

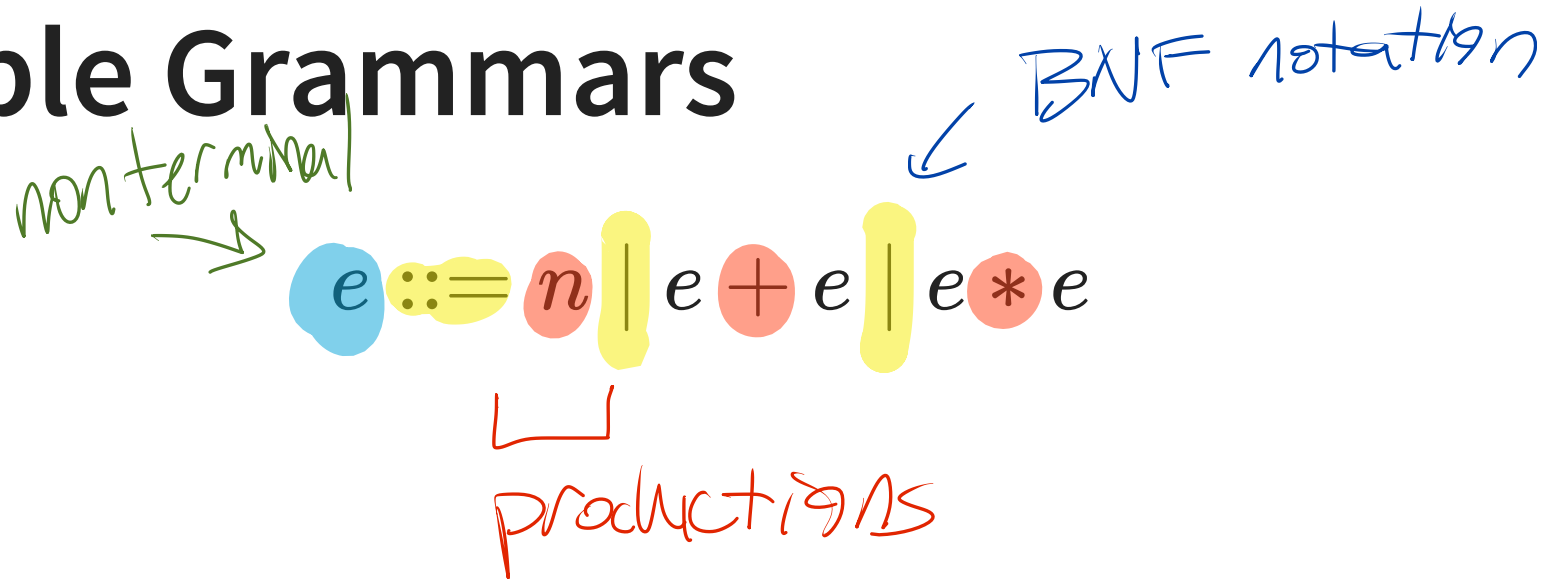
(derivation) "top-down" generates strings in a language L

parse tree "bottom up"

recognizer

string → "reads it" boolean (in L or not)

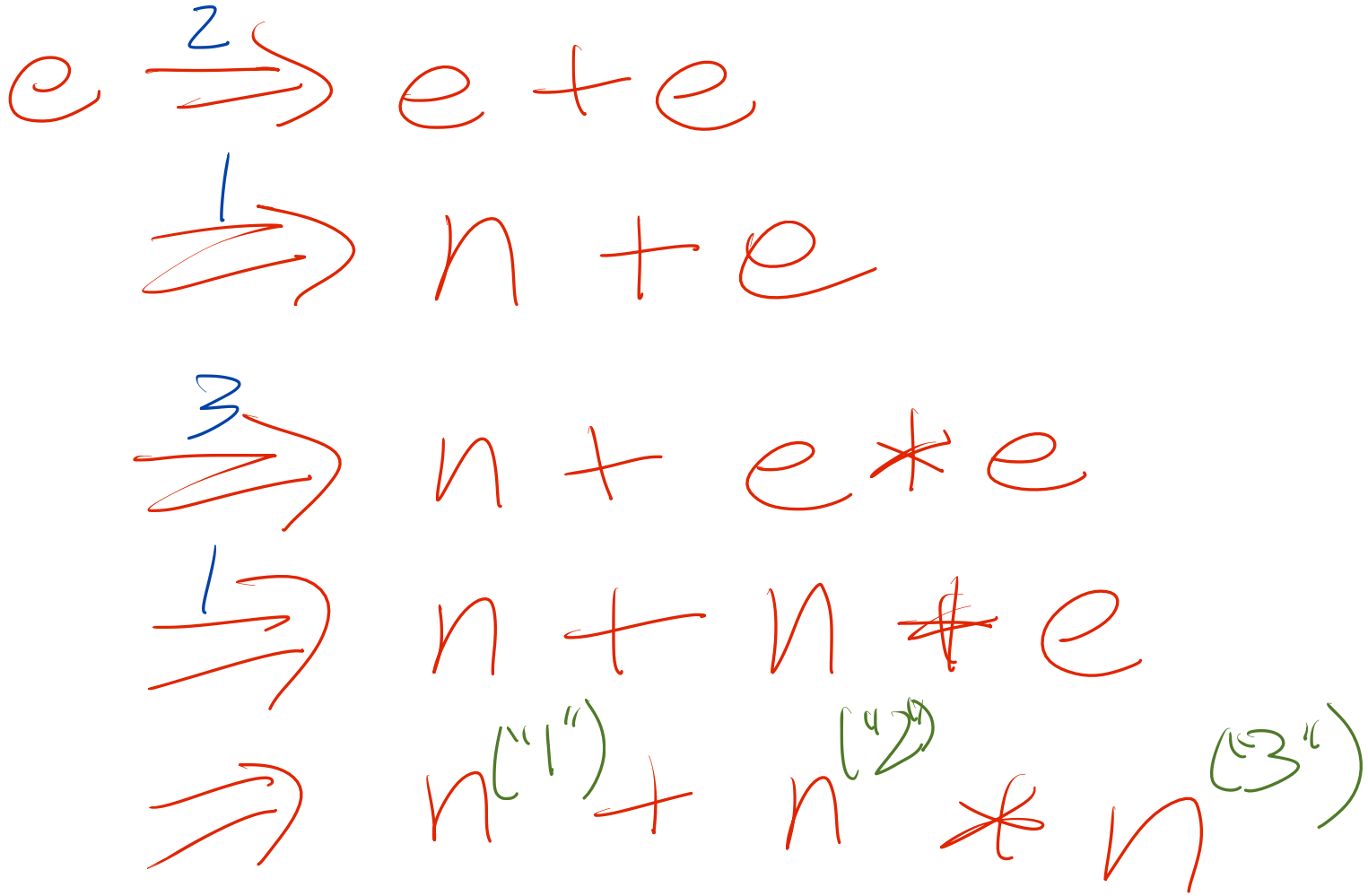
Example Grammars



Parsing Derivations

ambiguity
ambiguous
grammar

Give a *parsing derivation* for 1 + 2 * 3 with
$e ::= n \mid e + e \mid e * e.$

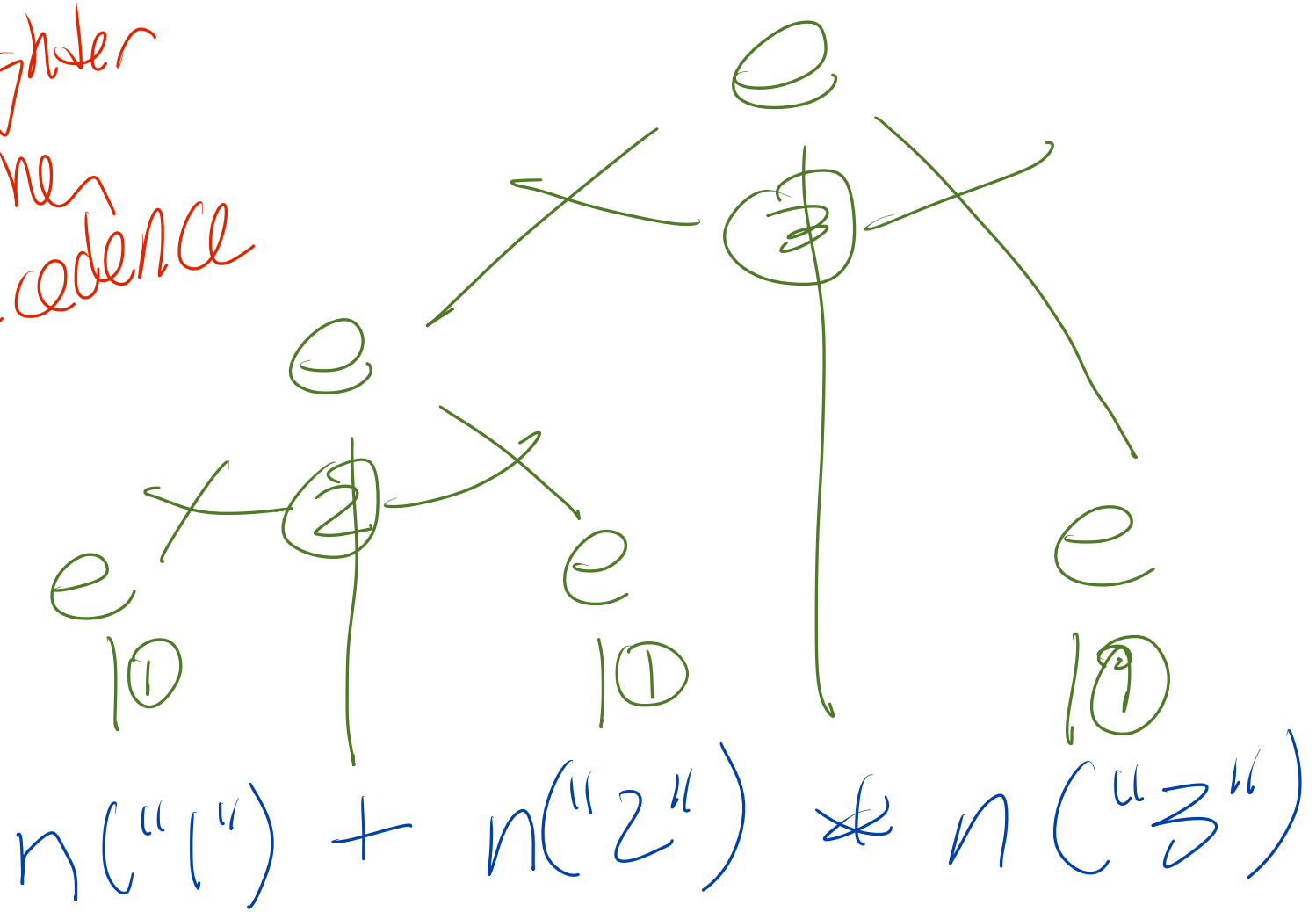


Parse Trees

precedence rules

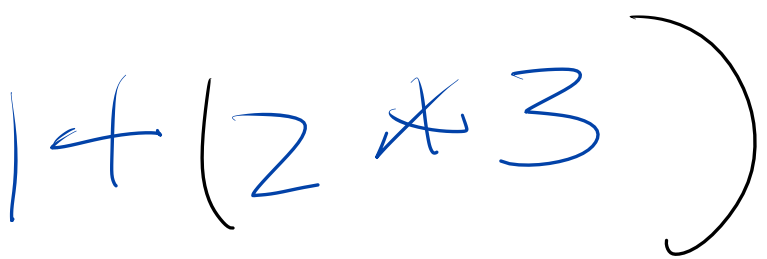
Give a *parse tree* for 1 + 2 * 3 with $e ::= \overset{1}{n} \mid e \overset{2}{+} e \mid e \overset{3}{*} e$.

binding tighter is higher precedence



Ambiguity

What should `1 + 2 * 3` evaluate to (i.e., what is the semantics of `1 + 2 * 3`)?



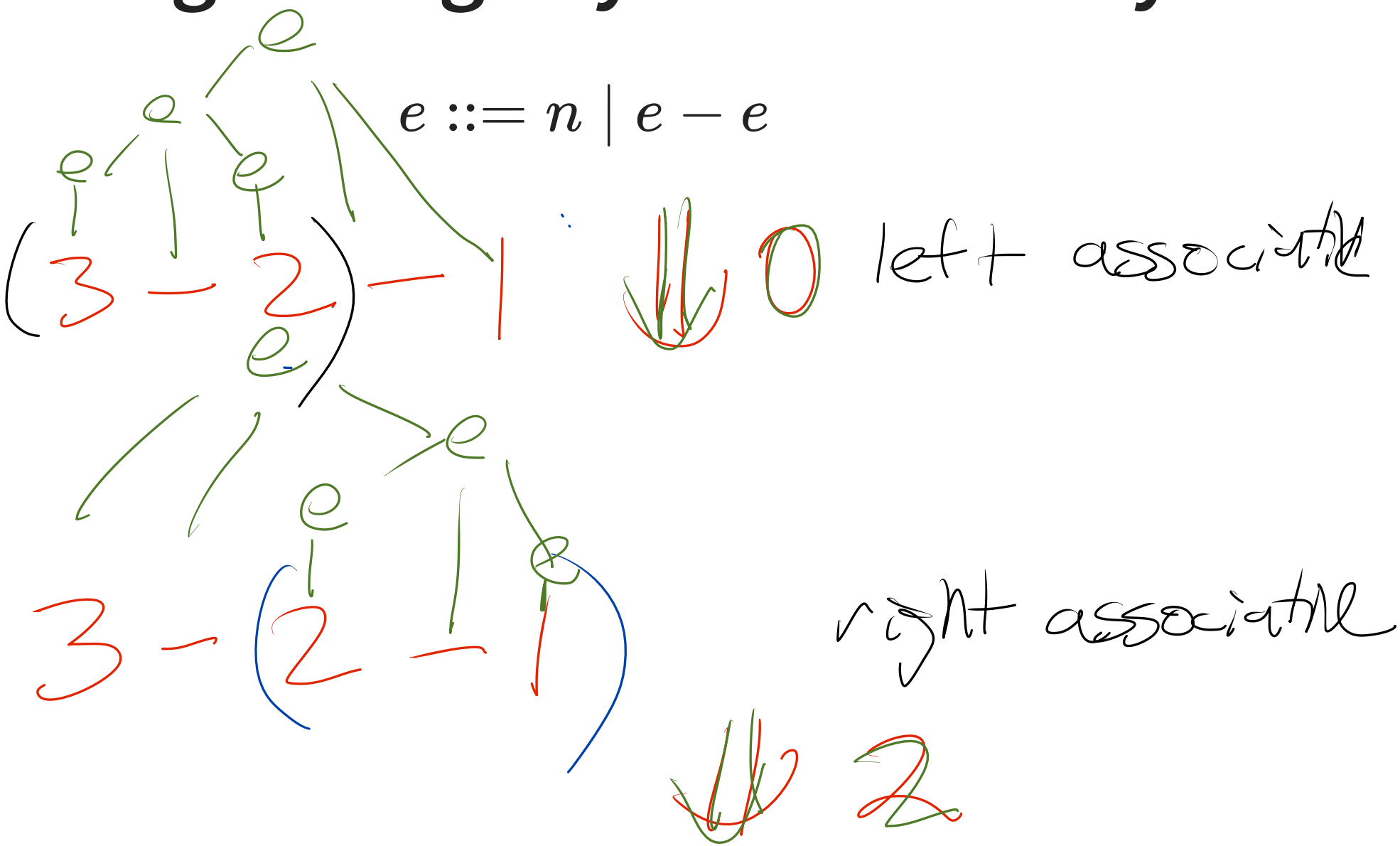
Ambiguity

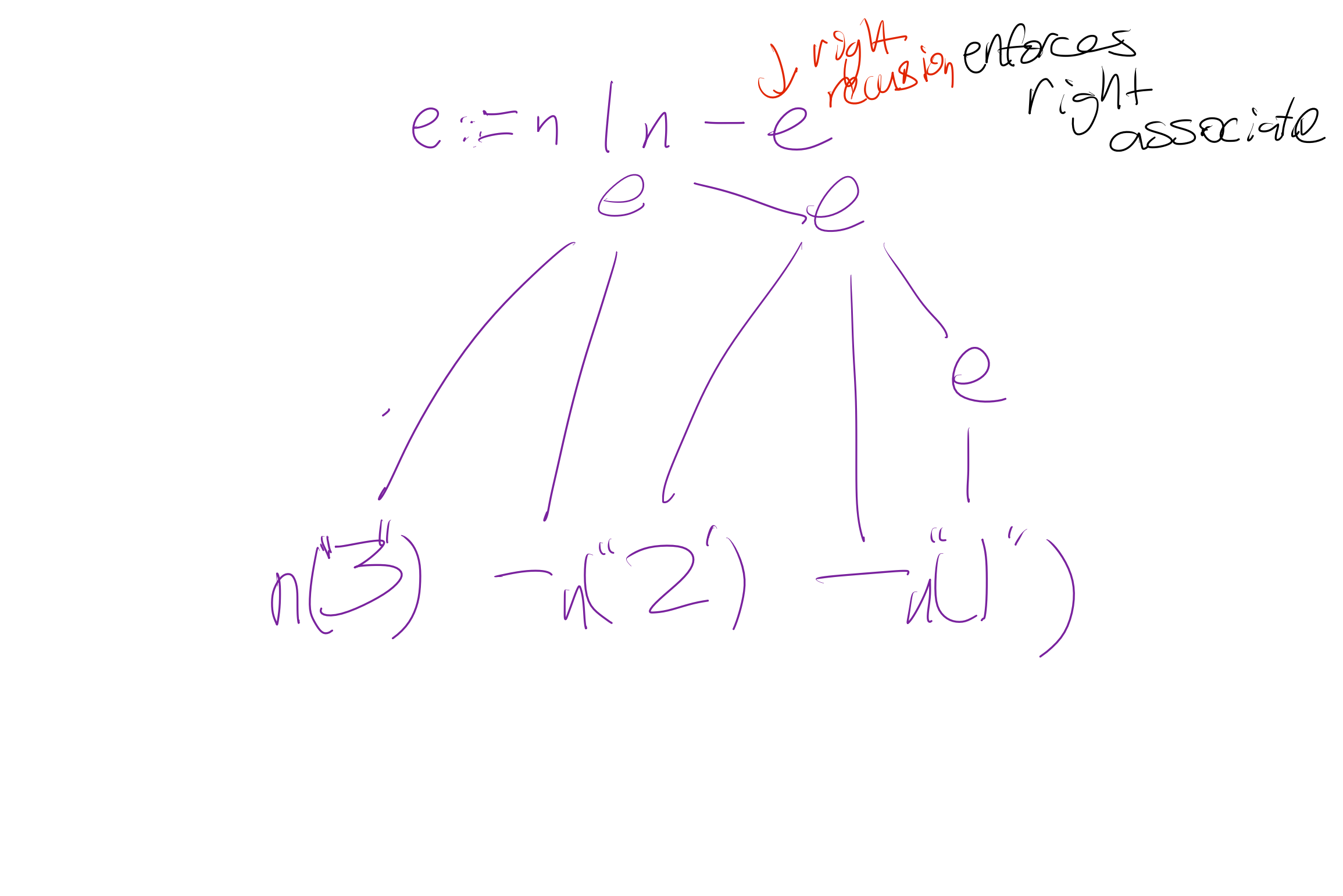
Give another parsing derivation for `1 + 2 * 3` with $e ::= n \mid e + e \mid e * e$.

Ambiguity

Give the corresponding parse tree as the previous derivation for `1 + 2 * 3` with $e ::= n \mid e + e \mid e * e$.

Resolving Ambiguity: Associativity




right recursion enforces right associate
e ::= n | n - e
e
e
e
n("3")
n("2")
n("1")

Resolving Ambiguity: Precedence

$$e ::= n \mid n + e \mid n * e$$

e+n e&n

unambiguous

right associativity

left

(4+1)&2

Resolving Ambiguity: Precedence

Consider the ambiguous grammar again:

$$e ::= n \mid e + e \mid e * e$$

$e + t$

$e ::= t \mid t + e$

$t ::= n \mid n * t$

$t * n$

Bonus: Resolving Ambiguity

$$
\begin{array}{rcl}
s & ::= & ifstmt \mid \cdots \\
ifstmt & ::= & \texttt{if } e \texttt{ then } s \\
& \mid & \texttt{if } e \texttt{ then } s \texttt{ else } s
\end{array}
$$

Ambiguous?

Write Grammars

Write a grammar for a language that has the string **`aa`**.

Write Grammars

Write a grammar for a language that has an even number of **a** ’s.

Write Grammars

Write a grammar for a language that has an odd number of **a**'s.

Write Grammars

Write a grammar that generates matching open and close braces { }.