CS 152

Programming Paradigms

BNF and Parsing
Ambiguity, Precedence & Associativity

Today

- BNF and Parsing
- Ambiguity
- Precedence
- Associativity

Course Learning Outcomes

5. Read and produce context-free grammars
6. Write recursive-descent parsers for simple languages, by hand or with a parser generator.

BNF

1)<sentence>→<noun-phrase><verb-phrase>.
2)<noun-phrase> → <article> <noun>
3)<article> → a | the
4)<noun> → girl | dog
5)<verb-phrase> → <verb> <noun-phrase>
6)<verb> → sees | pets

Context-free?

- Each production rule has a single non-terminal on the left, then a → metasymbol, followed by a sequence of terminals/tokens or other non-terminals on the right
- There is no context under which only certain replacements can occur
- Typically there are as many productions in a context-free grammar as there are non-terminals
- Terminals never appear on the left hand side of a rule

The Language

- The language defined by a context free grammar is the set of all strings of terminals for which there is a derivation beginning with the start symbol and ending with the strings of terminals.

iClicker: The Language

How many sentences are there in this language?

A. 5
B. 12
C. 16
D. 32
E. Infinity

1) <sentence>→<noun-phrase><verb-phrase>.
2) <noun-phrase> → <article> <noun>
3) <article> → a | the
4) <noun> → girl | dog
5) <verb-phrase> → <verb> <noun-phrase>
6) <verb> → sees | pets

Why BNF?

- BNF notation makes it easier to write translators
- Parsing stage can be automated

BNF for Arithmetic Expressions

<expr> → <expr> PLUS <expr> | <expr> TIMES <expr> |
LPAREN <expr> RPAREN | <number>

<number> → INT | FLOAT

UPPERCASE indicates it is a token whose structure is determined by the scanner

Rules can express recursion

The Language

<expr> → <expr> PLUS <expr> | <expr> TIMES <expr> |
LPAREN <expr> RPAREN | <number>

<number> → INT | FLOAT

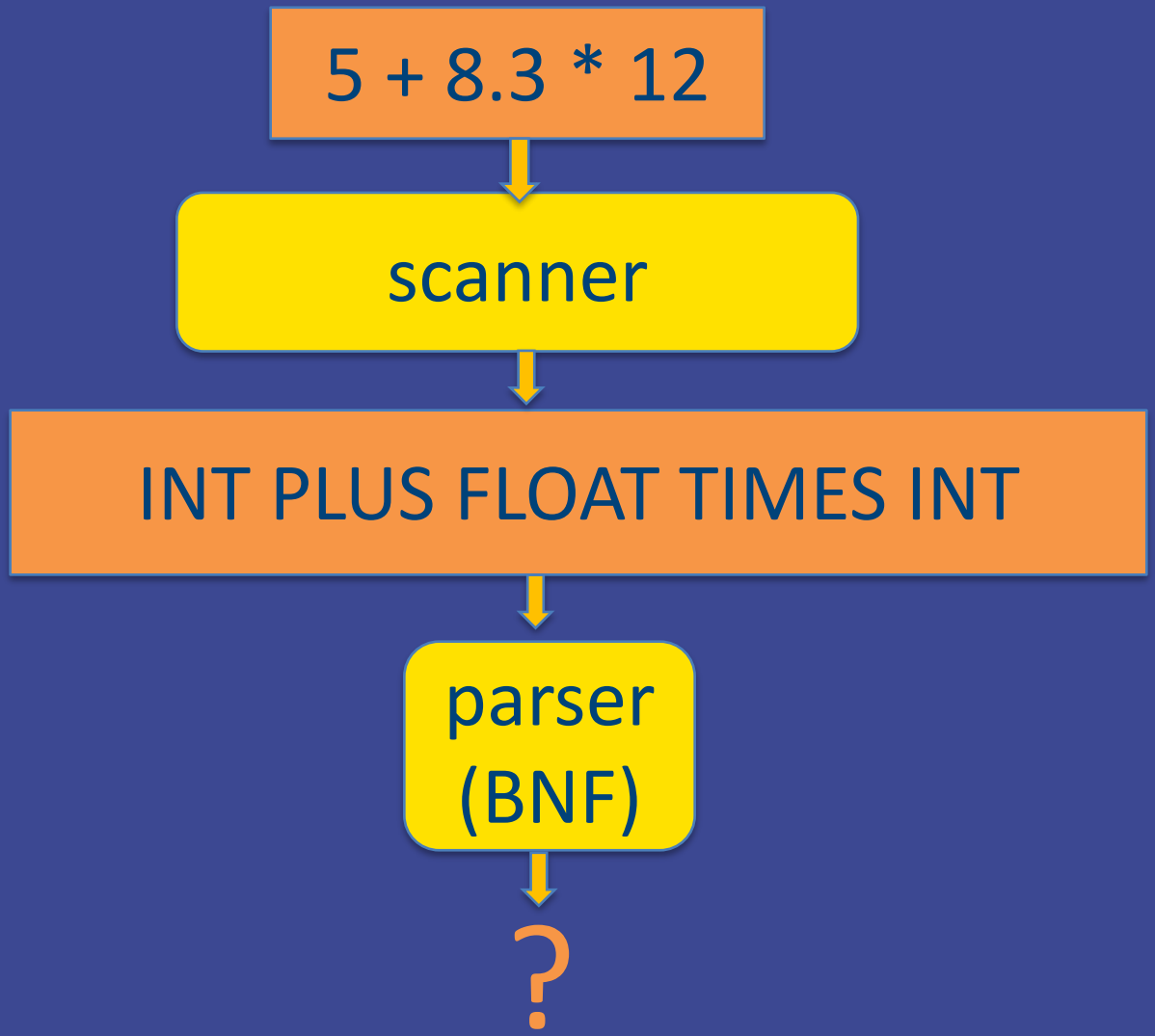
How many sentences are there in this language?

A. 5
B. 12
C. 16
D. 32
E. Infinity

Grammar and Parsing

- A grammar written in BNF describes the strings of tokens that are syntactically legal
- It also describes how a parser must act to parse correctly
- Recognizer: a program that accepts or rejects strings based on whether they are legal strings in the language

Question



Arithmetic Expressions

<expr> → <expr> PLUS <expr> | <expr> TIMES <expr> |
LPAREN <expr> RPAREN | <number>
<number> → INT | FLOAT

INT PLUS FLOAT TIMES INT

5 + 8.3 * 12

Valid expression?

Arithmetic Expressions

<expr> → <expr> PLUS <expr> | <expr> TIMES <expr> |
LPAREN <expr> RPAREN | <number>
<number> → INT | FLOAT

INT PLUS FLOAT TIMES INT

Start symbol?

<expr>

⇒ <expr> PLUS <expr>

⇒ <number> PLUS <expr>

⇒ INT PLUS <expr>

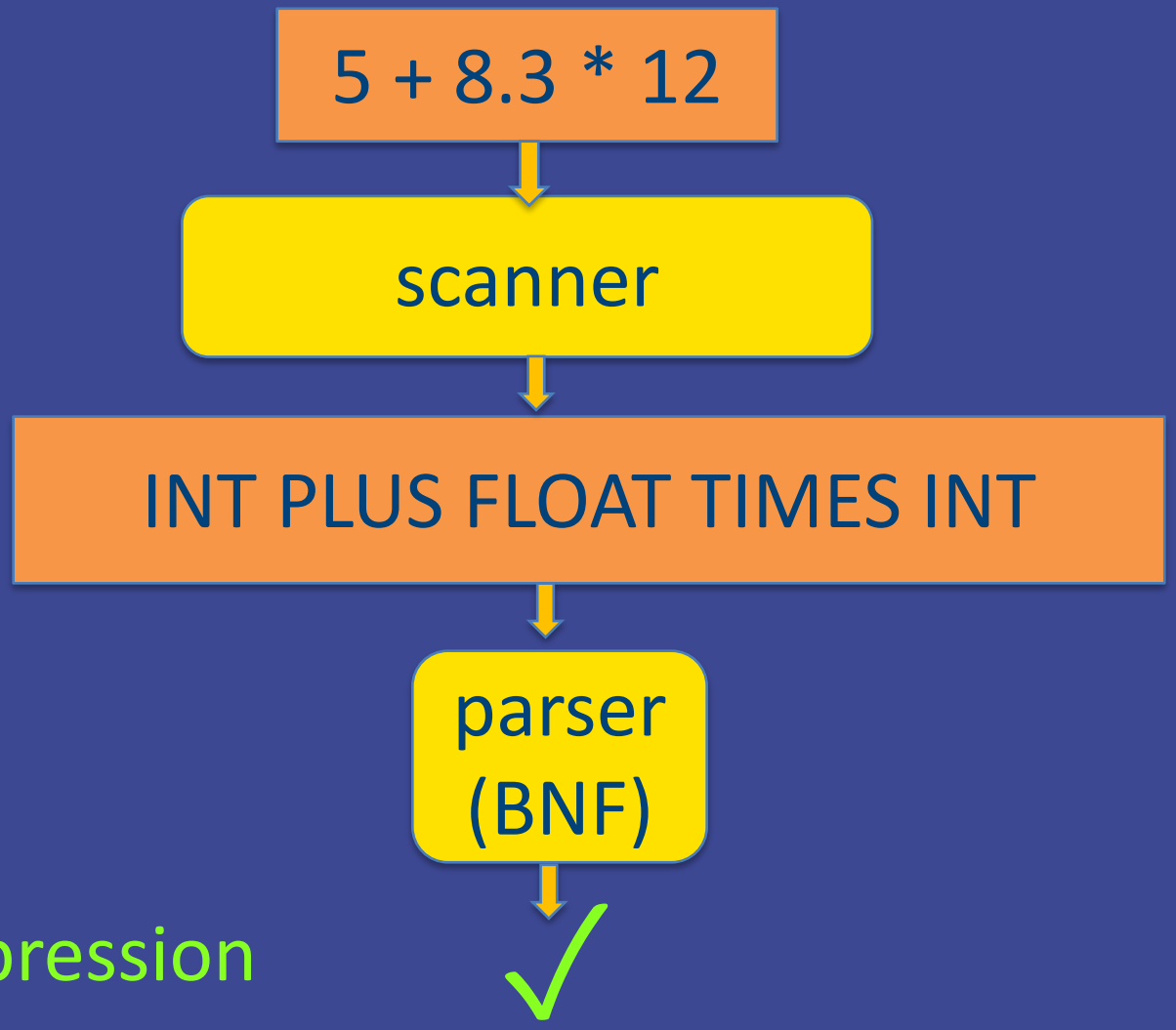
Arithmetic Expressions

<expr> → <expr> PLUS <expr> | <expr> TIMES <expr> |
LPAREN <expr> RPAREN | <number>
<number> → INT | FLOAT

INT PLUS FLOAT TIMES INT

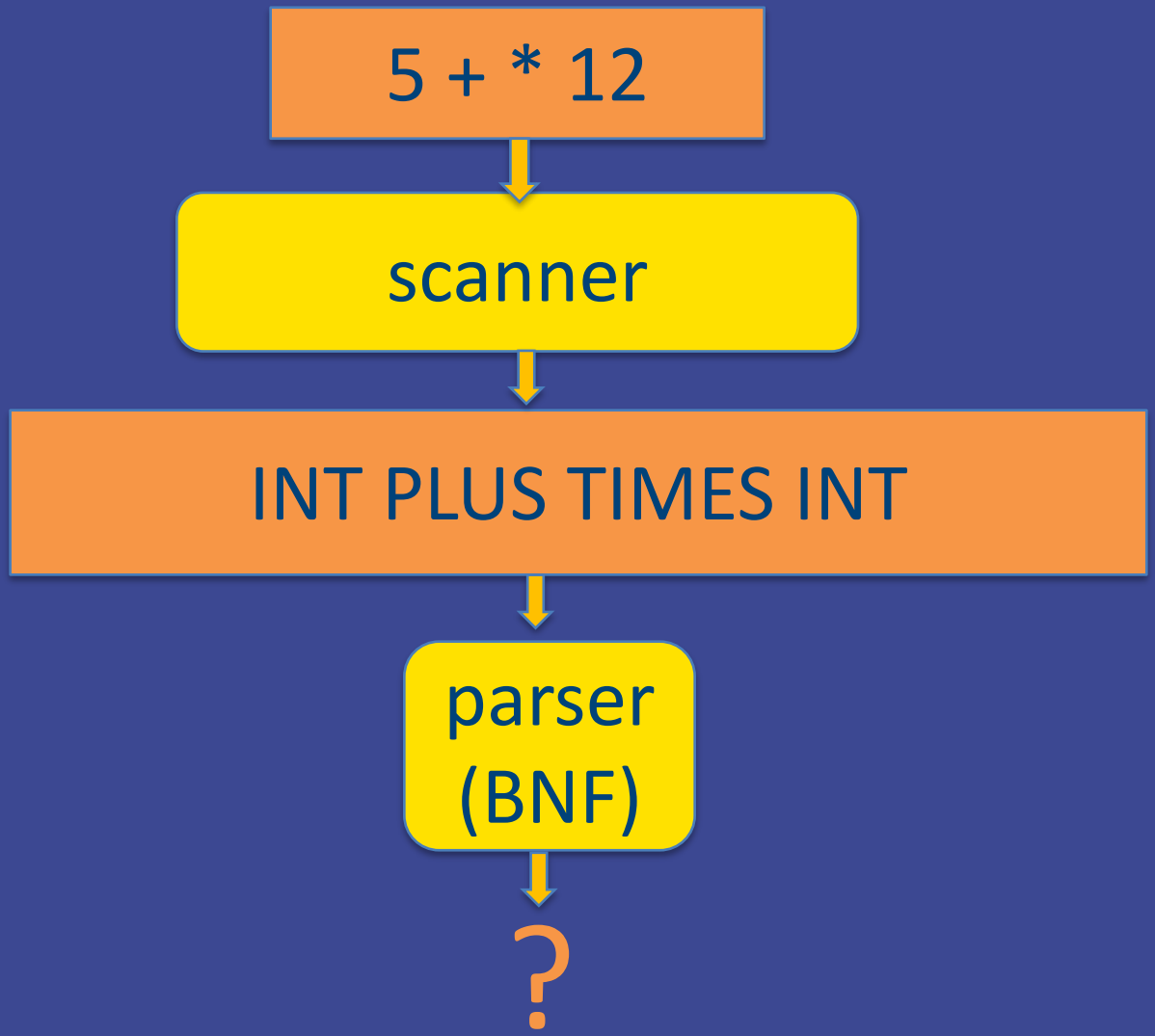
⇒ INT PLUS <expr>
⇒ INT PLUS <expr> TIMES <expr>
⇒ INT PLUS <number> TIMES <expr>
⇒ INT PLUS FLOAT TIMES <expr>
⇒ INT PLUS FLOAT TIMES <number>
⇒ INT PLUS FLOAT TIMES INT ✓

Answer



Yes. Valid expression

Question



Arithmetic Expressions

<expr> → <expr> PLUS <expr> | <expr> TIMES <expr> |
LPAREN <expr> RPAREN | <number>
<number> → INT | FLOAT

INT PLUS TIMES INT

<expr>
⇒ <expr> PLUS <expr>
⇒ <number> PLUS <expr>
⇒ INT PLUS <expr>
⇒ INT PLUS ?

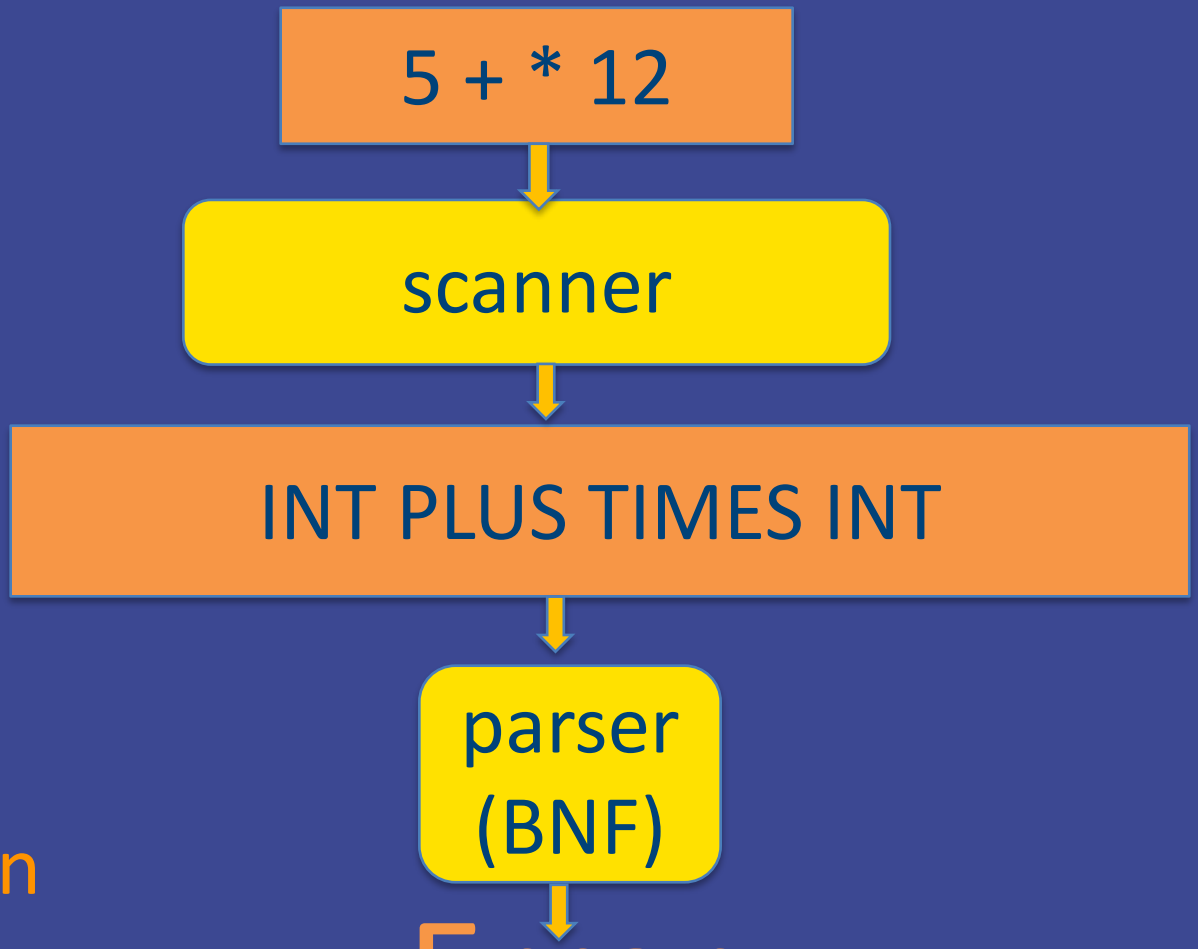
Arithmetic Expressions

<expr> → <expr> PLUS <expr> | <expr> TIMES <expr> |
LPAREN <expr> RPAREN | <number>
<number> → INT | FLOAT

INT PLUS TIMES INT

<expr>
⇒ <expr> TIMES <expr>
⇒ <expr> TIMES INT
⇒ ?

The Big Picture



No valid derivation
=> invalid syntax

BNF for Prefix Calculator?

Write a BNF for a prefix calculator.

Use non terminals: <expr>, <operator>, <operands> and <number>

Use tokens: PLUS, MINUS, TIMES, DIVIDE, INT and FLOAT

The expressions below are included in the language:

+ 2

* 7 8

+ 6 8 5 34

What is the start symbol?

BNF for Prefix Calculator?

<expr> →

<operator> → PLUS | MINUS | TIMES | DIVIDE

<operands> →

<number> → INT | FLOAT

iClicker: BNF for Prefix Calculator?

A. <expr> → <operator> <number>
B. <expr> → <operator> INT | <operator> FLOAT
C. <expr> → PLUS <number> | MINUS <number> | TIMES <number> | DIVIDE <number>
D. <expr> → <operator> <operands>
E. <expr> → <operands>

```
+ 2
* 7 8
+ 6 8 5  34
```

BNF for Prefix Calculator?

<expr> → <operator> <operands>

<operator> → PLUS | MINUS | TIMES | DIVIDE

<operands> → <number> | <number> <operands>

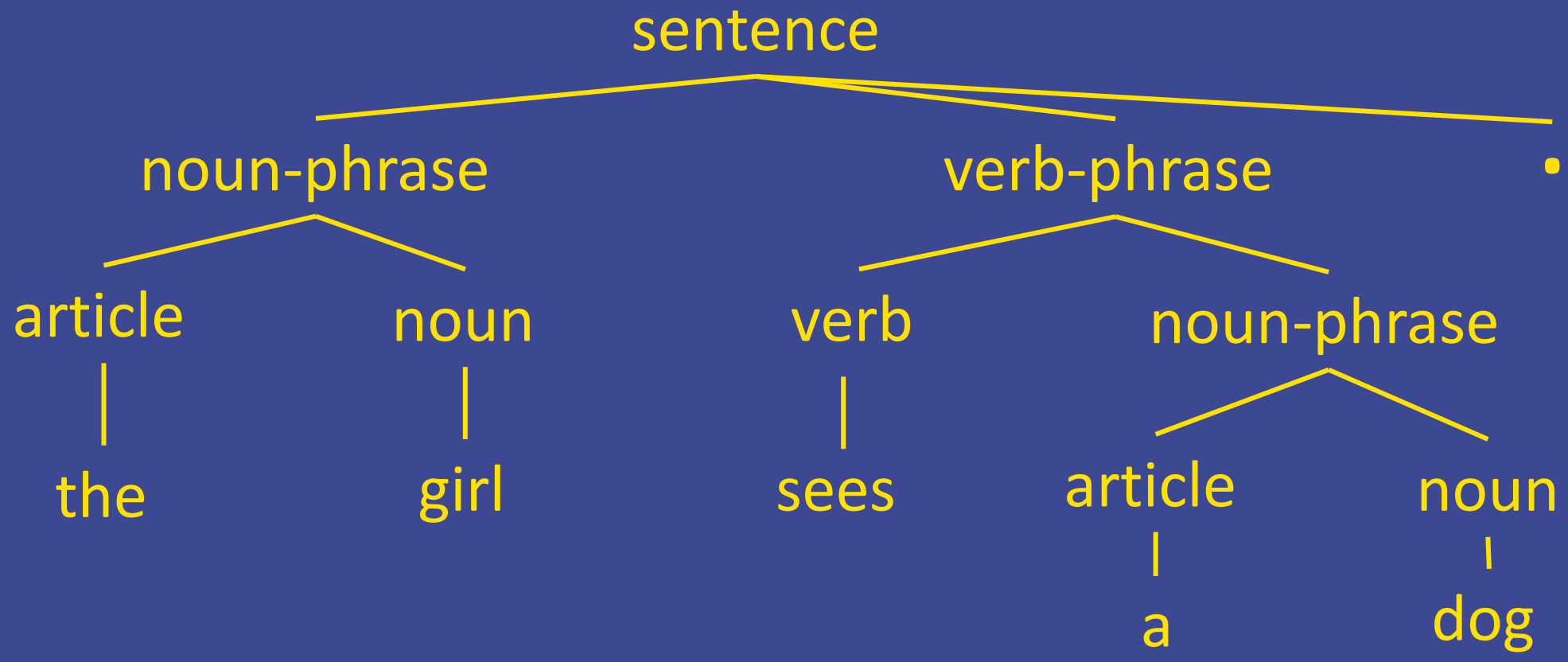
<number> → INT | FLOAT

Syntax vs Semantics

- Syntax establishes structure, not meaning
  - But meaning is related to syntax
- Syntax-directed semantics: process of associating the semantics of a construct to its syntactic structure
  - Must construct the syntax so that it reflects the semantics to be attached later

Parse Tree

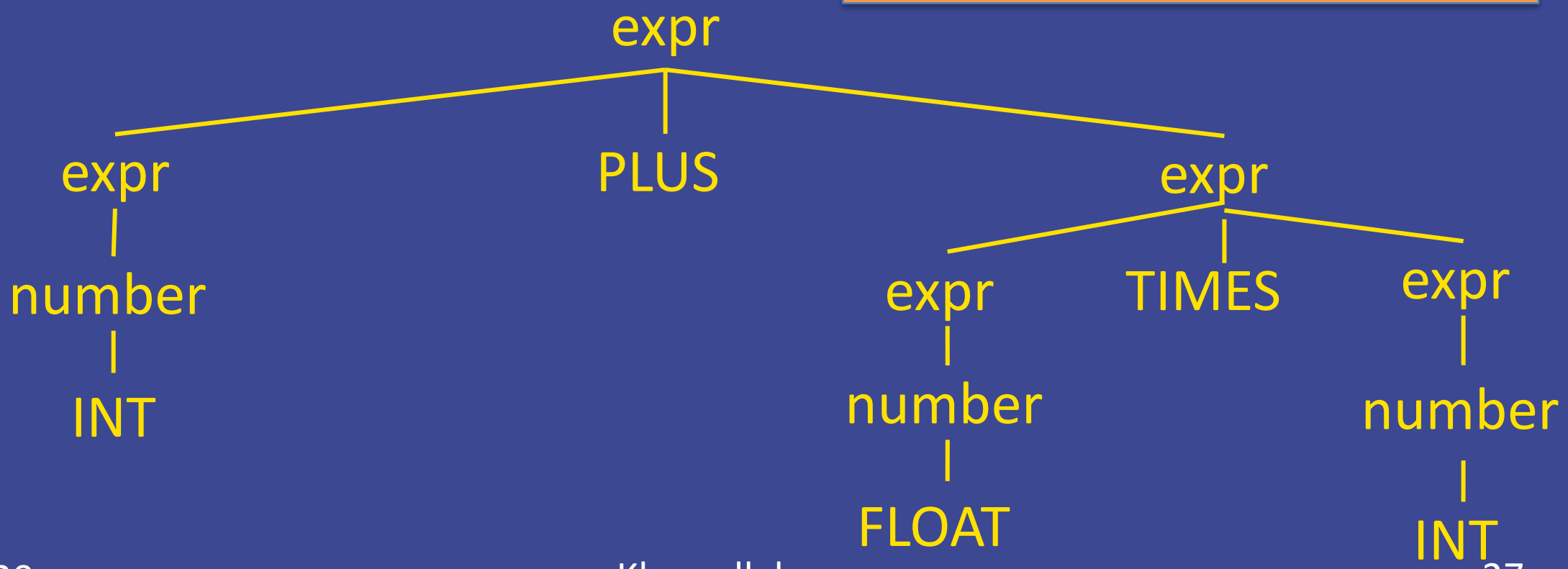
Parse tree: graphical depiction of the replacement process in a derivation



Parse Tree

<expr> → <expr> PLUS <expr> | <expr> TIMES <expr> |
LPAREN <expr> RPAREN | <number>
<number> → INT | FLOAT

INT PLUS FLOAT TIMES INT

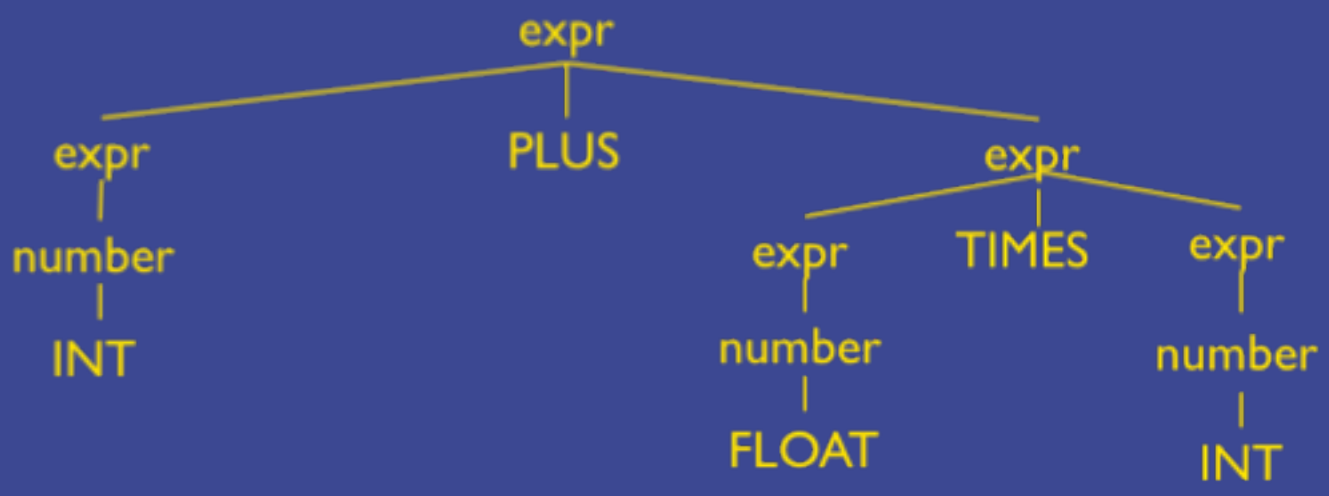


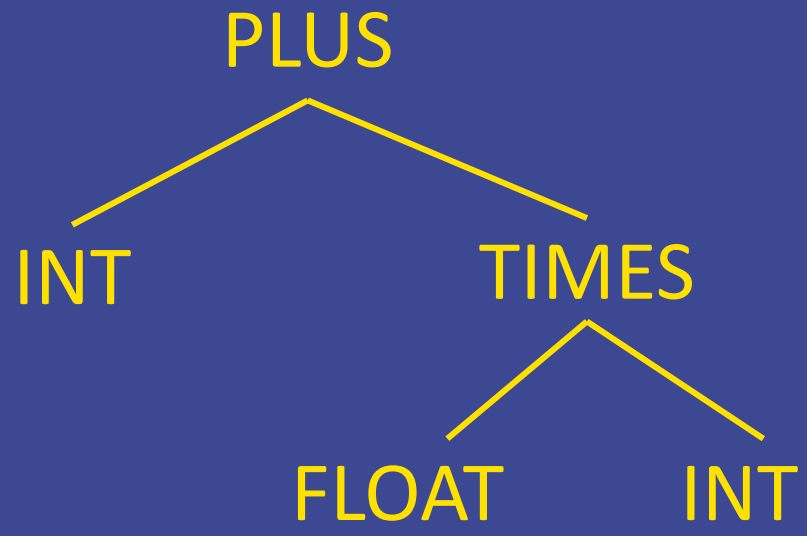
Parse Trees

- Nodes that have at least one child are labeled with non-terminals
- Leaves (nodes with no children) are labeled with terminals/tokens
- The structure of a parse tree is completely specified by the grammar rules of the language and a derivation of the sequence of terminals
- All terminals and non-terminals in a derivation are included in the parse tree

Parse Trees vs Abstract Syntax Trees

We do not need all the nodes to determine the syntactic structure of an expression

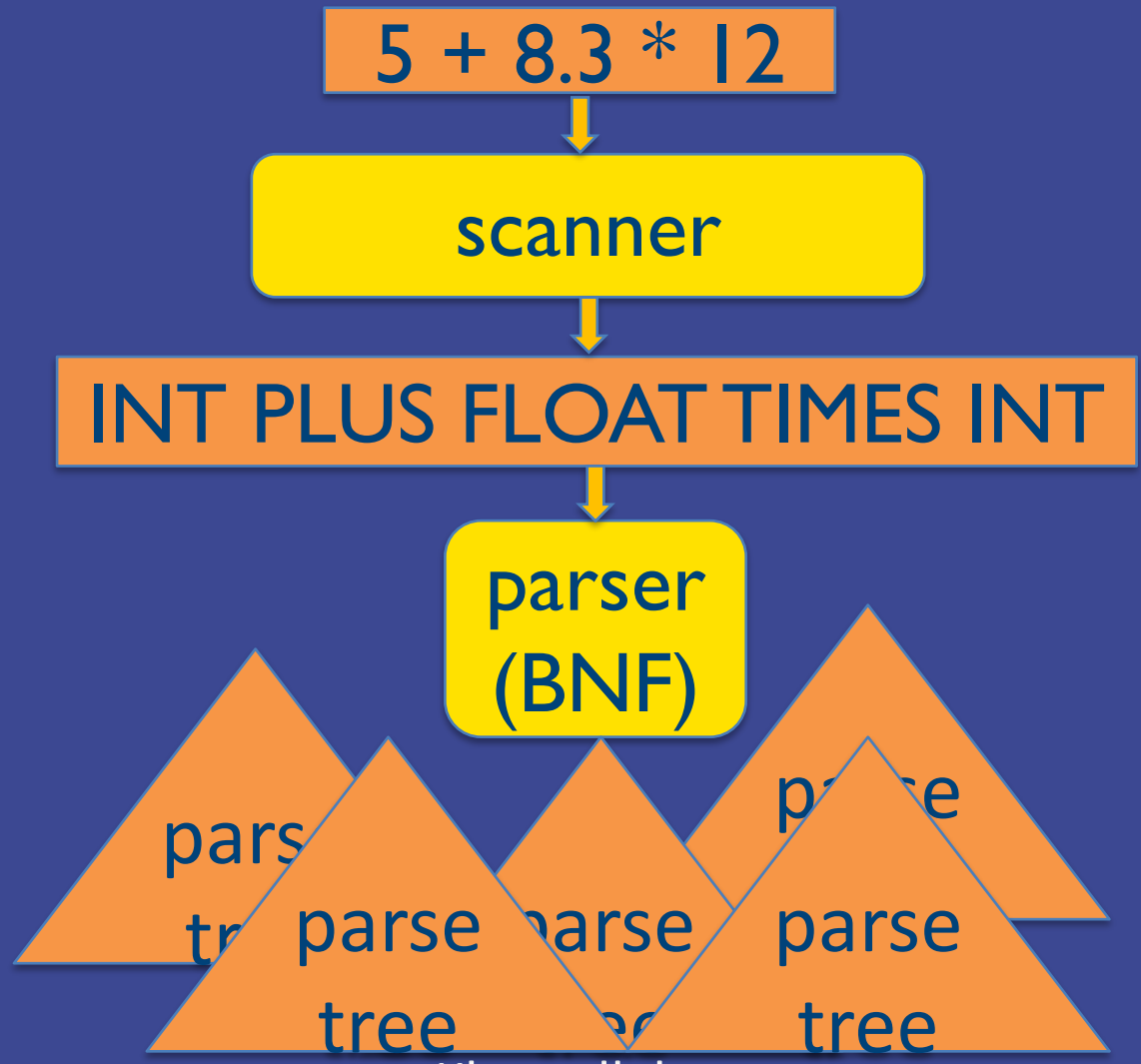




Parse Tree (Concrete Syntax Tree)

Abstract Syntax Tree (AST)

The Big Picture



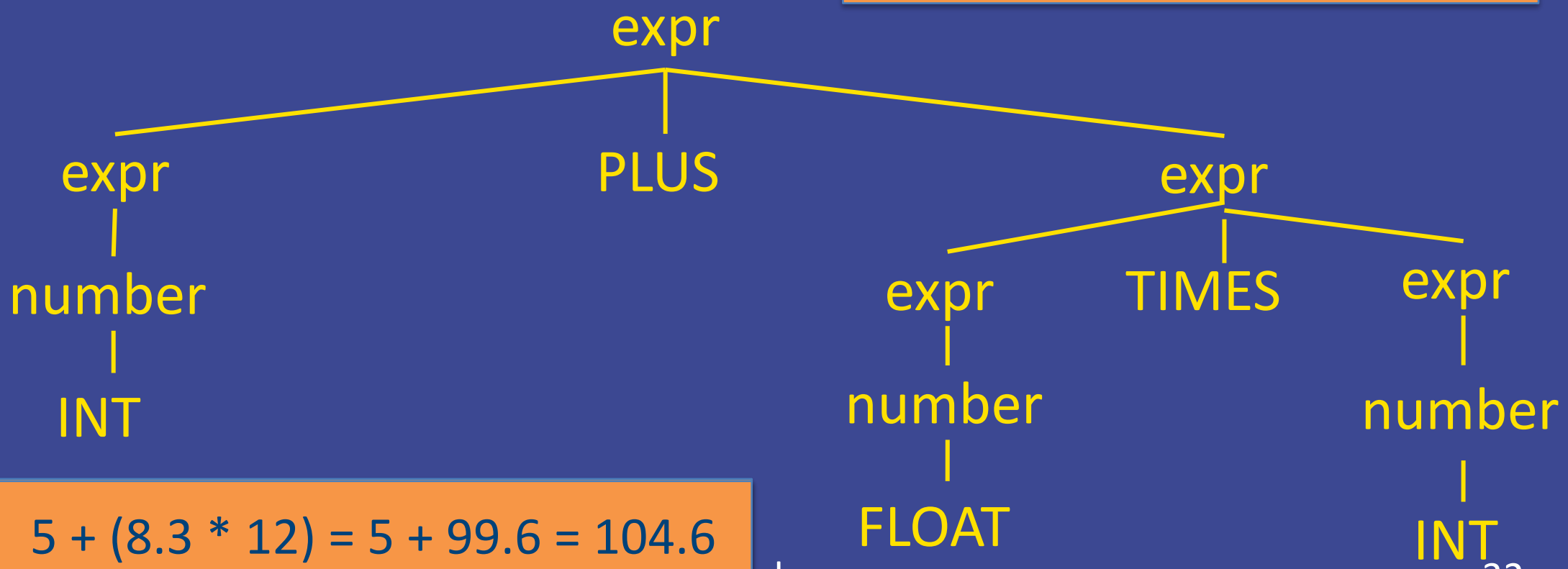
Ambiguity

- Ambiguous grammar: one for which two distinct parse trees are possible for a given valid string

First Parse Tree

<expr> → <expr> PLUS <expr> | <expr> TIMES <expr> |
LPAREN <expr> RPAREN | <number>
<number> → INT | FLOAT

INT PLUS FLOAT TIMES INT

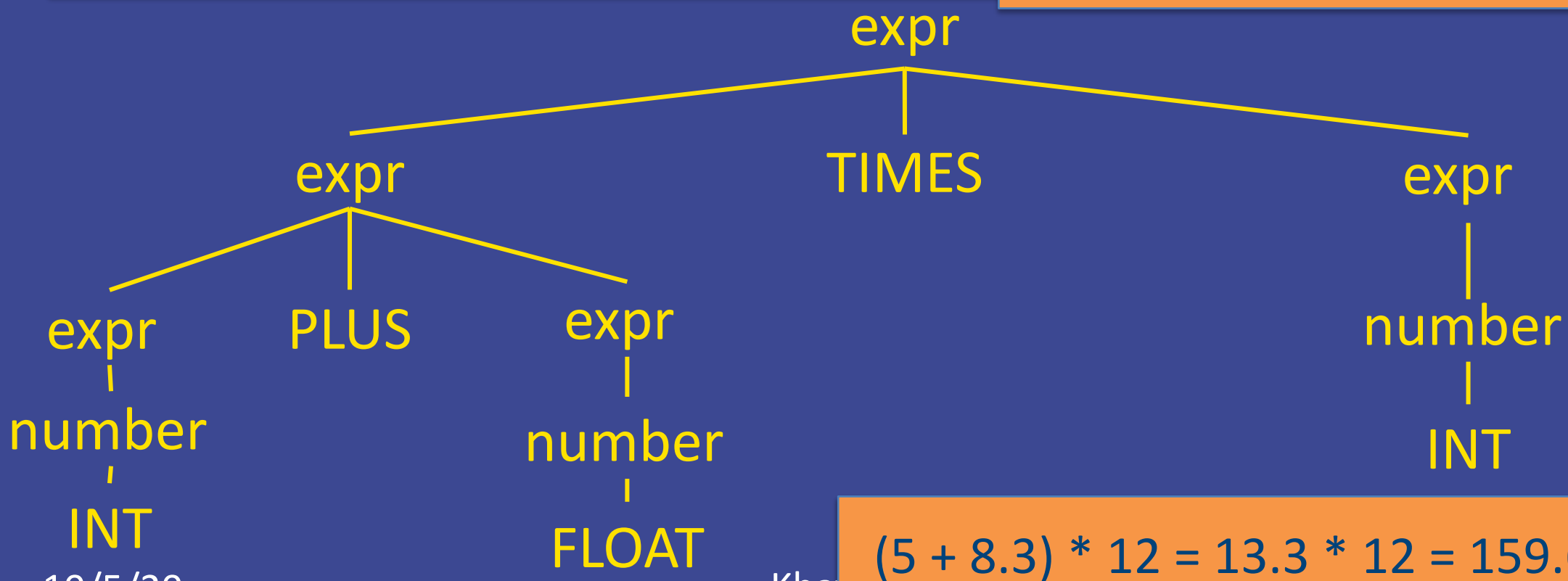


5 + (8.3 * 12) = 5 + 99.6 = 104.6

Second Parse Tree

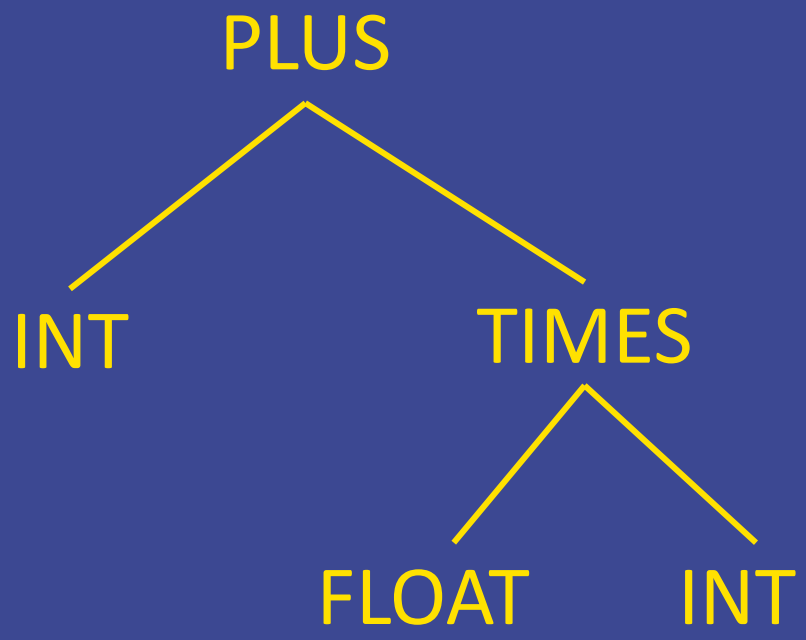
<expr> → <expr> PLUS <expr> | <expr> TIMES <expr> |
LPAREN <expr> RPAREN | <number>
<number> → INT | FLOAT

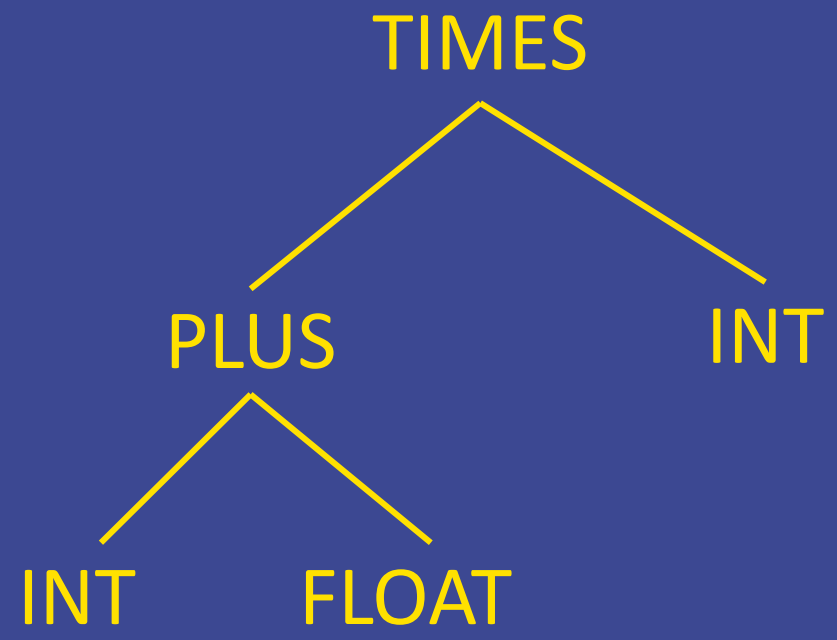
INT PLUS FLOAT TIMES INT



(5 + 8.3) * 12 = 13.3 * 12 = 159.6

Abstract Syntax Trees





5 + (8.3 * 12) = 5 + 99.6 = 104.6

(5 + 8.3) * 12 = 13.3 * 12 = 159.6

=> Ambiguous grammar

Leftmost Derivation

- Leftmost derivation: the leftmost remaining nonterminal is singled out for replacement at each step
- Each parse tree has a unique leftmost derivation
- Ambiguity of a grammar can be tested by searching for two different leftmost derivations
- Unambiguous grammar is a context-free grammar for which every valid string has a unique leftmost derivation

Two Leftmost Derivations?

<expr>
⇒ <expr> PLUS <expr>
⇒ <number> PLUS <expr>
⇒ INT PLUS <expr>
⇒ INT PLUS <expr> TIMES <expr>
⇒ INT PLUS <number> TIMES <expr>
⇒ INT PLUS FLOAT TIMES <expr>
⇒ INT PLUS FLOAT TIMES <number>
⇒ INT PLUS FLOAT TIMES INT

<expr>
⇒ <expr> TIMES <expr>
⇒ <expr> PLUS <expr> TIMES <expr>
⇒ <number> PLUS <expr> TIMES <expr>
⇒ INT PLUS <expr> TIMES <expr>
⇒ INT PLUS <number> TIMES <expr>
⇒ INT PLUS FLOAT TIMES <expr>
⇒ INT PLUS FLOAT TIMES <number>
⇒ INT PLUS FLOAT TIMES INT

INT PLUS FLOAT TIMES INT

Leftmost Derivation 1

<expr>

⇒ <expr> PLUS <expr>

⇒ <number> PLUS <expr>

⇒ INT PLUS <expr>

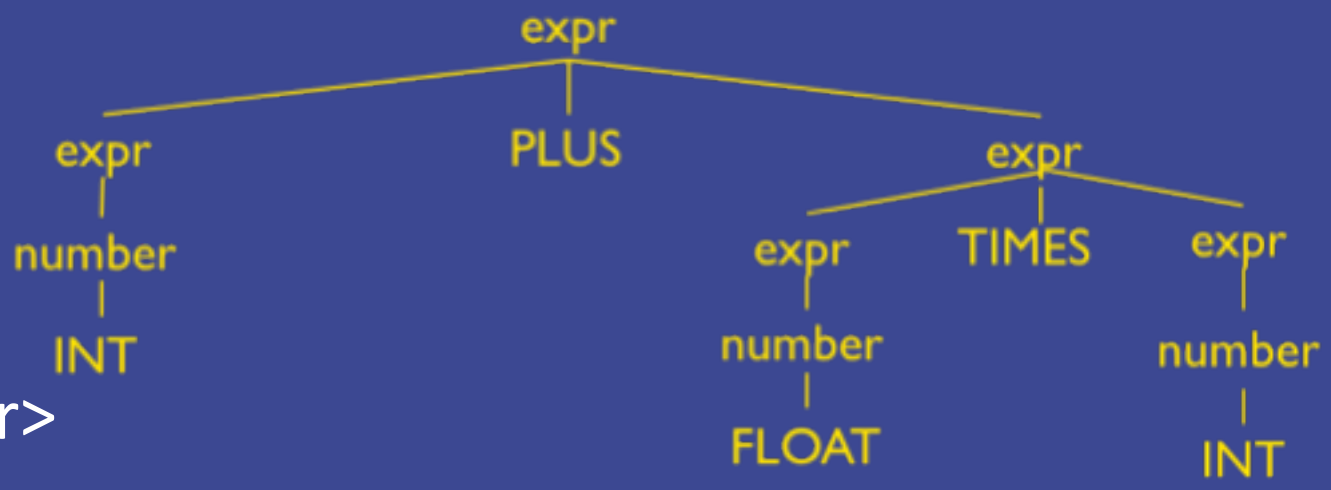
⇒ INT PLUS <expr> TIMES <expr>

⇒ INT PLUS <number> TIMES <expr>

⇒ INT PLUS FLOAT TIMES <expr>

⇒ INT PLUS FLOAT TIMES <number>

⇒ INT PLUS FLOAT TIMES INT



INT PLUS FLOAT TIMES INT

Leftmost Derivation 2

<expr>

⇒ <expr> TIMES <expr>

⇒ <expr> PLUS <expr> TIMES <expr>

⇒ <number> PLUS <expr> TIMES <expr>

⇒ INT PLUS <expr> TIMES <expr>

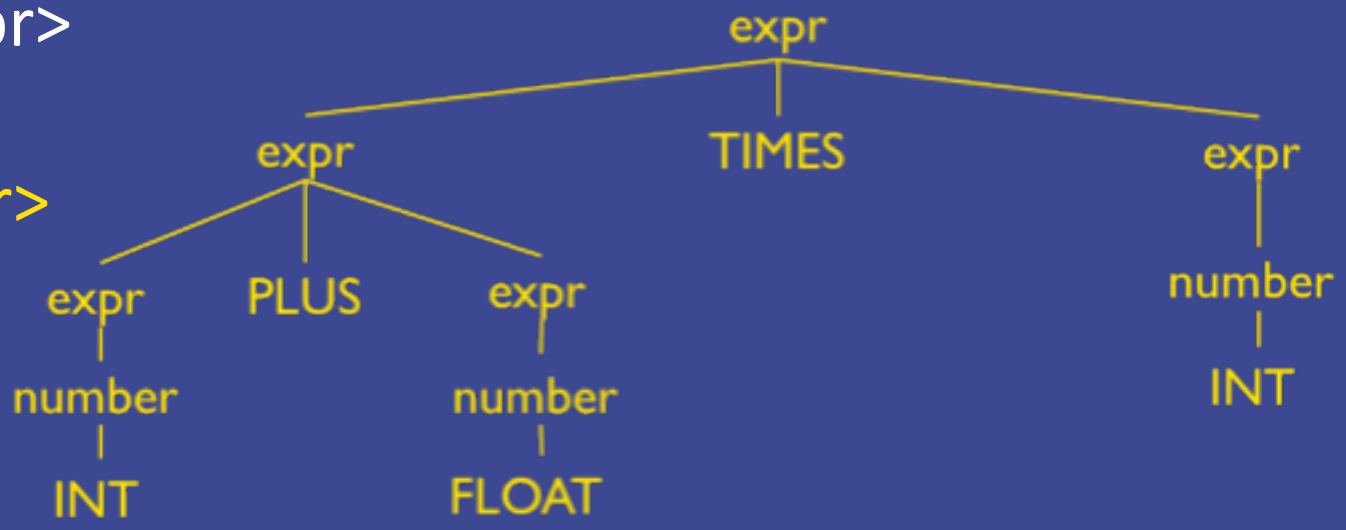
⇒ INT PLUS <number> TIMES <expr>

⇒ INT PLUS FLOAT TIMES <expr>

⇒ INT PLUS FLOAT TIMES <number>

⇒ INT PLUS FLOAT TIMES INT

INT PLUS FLOAT TIMES INT



Precedence

<expr> → <expr> PLUS <expr> | <expr> TIMES <expr> |
LPAREN <expr> RPAREN | <number>
<number> → INT | FLOAT

- The problem is that the grammar above does not specify the precedence of the PLUS and TIMES operators

Precedence

- Ambiguous grammars present difficulties
- We must either revise them to remove ambiguity or state a disambiguating rule
- The usual way to revise the grammar is to write a new grammar rule that establishes a precedence cascade

Precedence

We can replace

<expr> → <expr> PLUS <expr> | <expr> TIMES <expr> |
LPAREN <expr> RPAREN | <number>

With:

<expr> → <expr> PLUS <expr> | <term>

<term> → <term> TIMES <term> | LPAREN <expr> RPAREN |
<number>

Precedence

<expr> → <expr> PLUS <expr> | <term>

<term> → <term> TIMES <term> | LPAREN <expr> RPAREN |
<number>

<number> → INT | FLOAT

Precedence

<expr> → <expr> PLUS <expr> | <term>
<term> → <term> TIMES <term> | LPAREN <expr> RPAREN |
<number>
<number> → INT | FLOAT

INT PLUS FLOAT TIMES INT

<expr>
⇒ <expr> PLUS <expr>
⇒ <term> PLUS <expr>
⇒ <number> PLUS <expr>
⇒ INT PLUS <expr>
⇒ INT PLUS <term>

Precedence

<expr> → <expr> PLUS <expr> | <term>
<term> → <term> TIMES <term> | LPAREN <expr> RPAREN |
<number>
<number> → INT | FLOAT

INT PLUS FLOAT TIMES INT

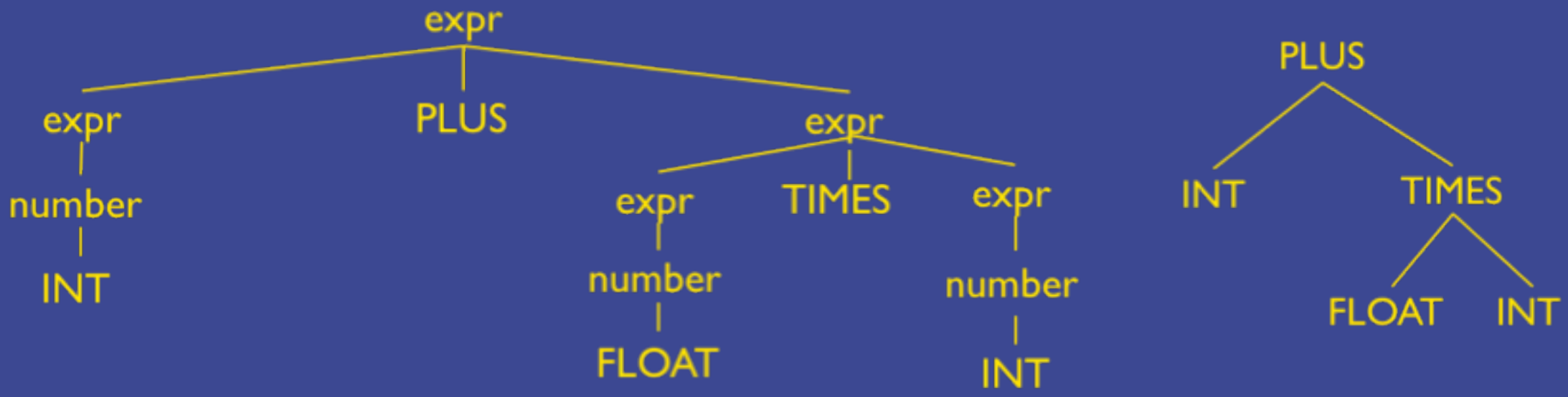
⇒ INT PLUS <term>
⇒ INT PLUS <term> TIMES <term>
⇒ INT PLUS <number> TIMES <term>
⇒ INT PLUS FLOAT TIMES <term>
⇒ INT PLUS FLOAT TIMES <number>
⇒ INT PLUS FLOAT TIMES INT

The expression
5 + 8.3 * 12
has now a unique
leftmost derivation.

Precedence

<expr> → <expr> PLUS <expr> | <term>
<term> → <term> TIMES <term> | LPAREN <expr> RPAREN |
<number>
<number> → INT | FLOAT

INT PLUS FLOAT TIMES INT



The expression 5 + 8.3 * 12 has now a unique parse tree.

Ambiguity?

- Unambiguous grammar is a context-free grammar for which every valid string has a unique leftmost derivation and hence a unique parse tree.

iClicker: Ambiguity?

<expr> → <expr> PLUS <expr> | <term>
<term> → <term> TIMES <term> | LPAREN <expr> RPAREN |
<number>
<number> → INT | FLOAT

7 + 3.2 + 2

Is the grammar above ambiguous?

A. Yes
B. No
C. It depends

Ambiguity?

<expr> → <expr> PLUS <expr> | <term>
<term> → <term> TIMES <term> | LPAREN <expr> RPAREN |
<number>
<number> → INT | FLOAT

7 + 3.2 + 2

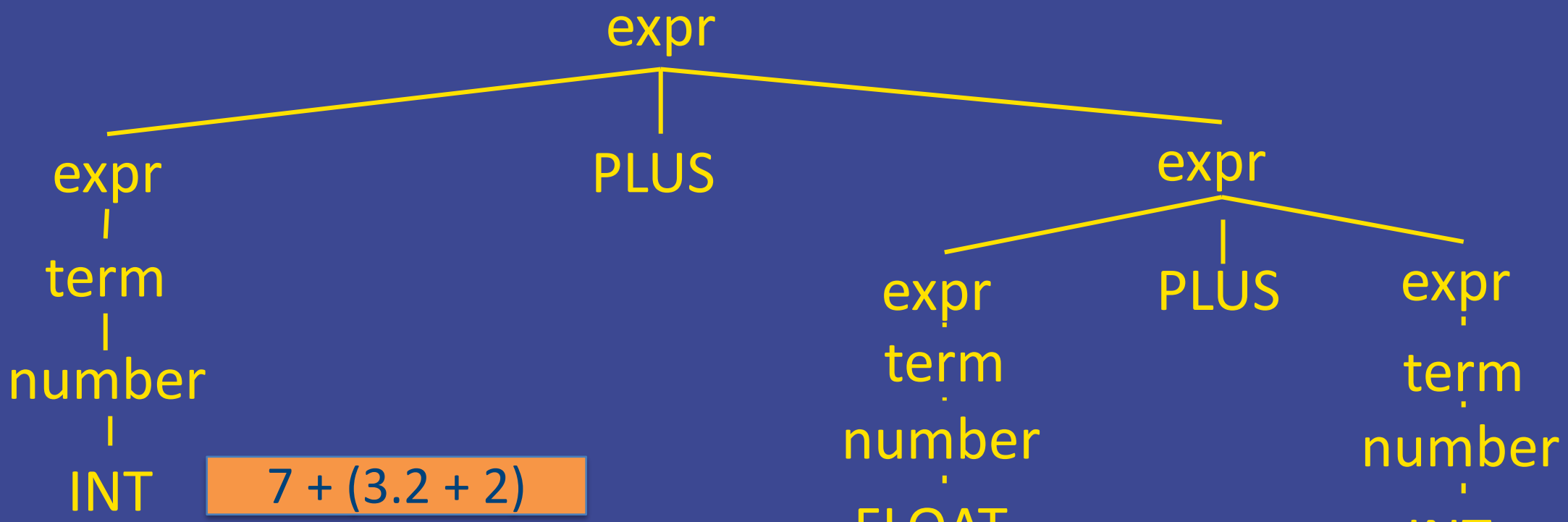
The grammar is still ambiguous because there are two leftmost derivations/parse trees for 7 + 3.2 + 2

First Parse Tree for 7 + 3.2 + 2

<expr> → <expr> PLUS <expr> | <term>
<term> → <term> TIMES <term> | LPAREN <expr> RPAREN | <number>
<number> → INT | FLOAT

7 + 3.2 + 2

INT PLUS FLOAT PLUS INT



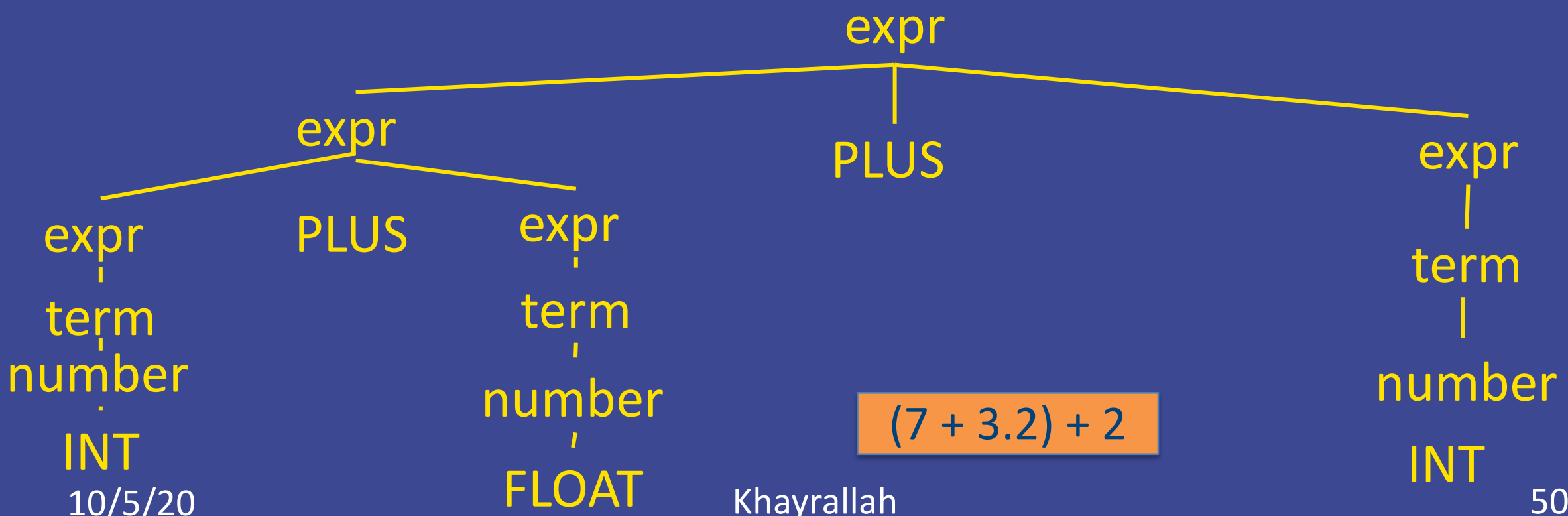
7 + (3.2 + 2)

Second Parse Tree for 7 + 3.2 + 2

<expr> → <expr> PLUS <expr> | <term>
<term> → <term> TIMES <term> | LPAREN <expr> RPAREN | <number>
<number> → INT | FLOAT

7 + 3.2 + 2

INT PLUS FLOAT PLUS INT



(7 + 3.2) + 2

Associativity

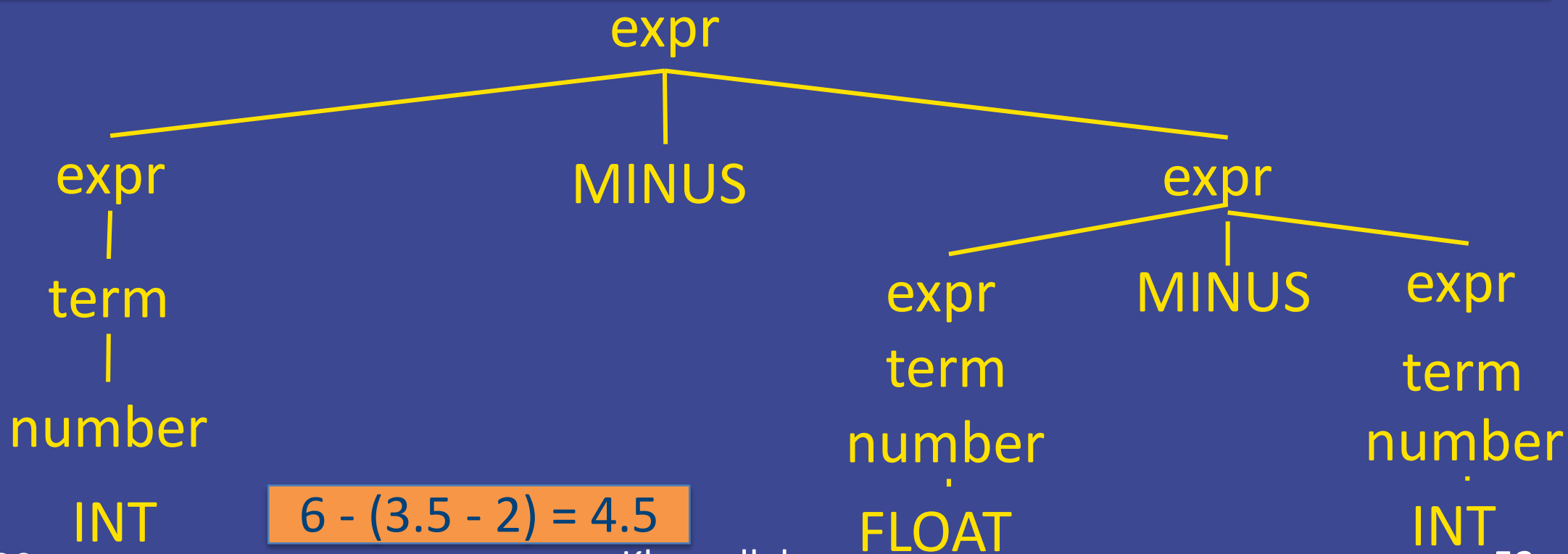
<expr> → <expr> PLUS <expr> | <term>
<term> → <term> TIMES <term> | LPAREN <expr> RPAREN | <number>
<number> → INT | FLOAT

- The problem is that the grammar above does not specify the associativity of the PLUS and TIMES operators

Subtraction?

<expr> → <expr> PLUS <expr> | <expr> MINUS <expr> | <term>
<term> → <term> TIMES <term> | LPAREN <expr> RPAREN | <number>
<number> → INT | FLOAT

INT MINUS FLOAT MINUS INT

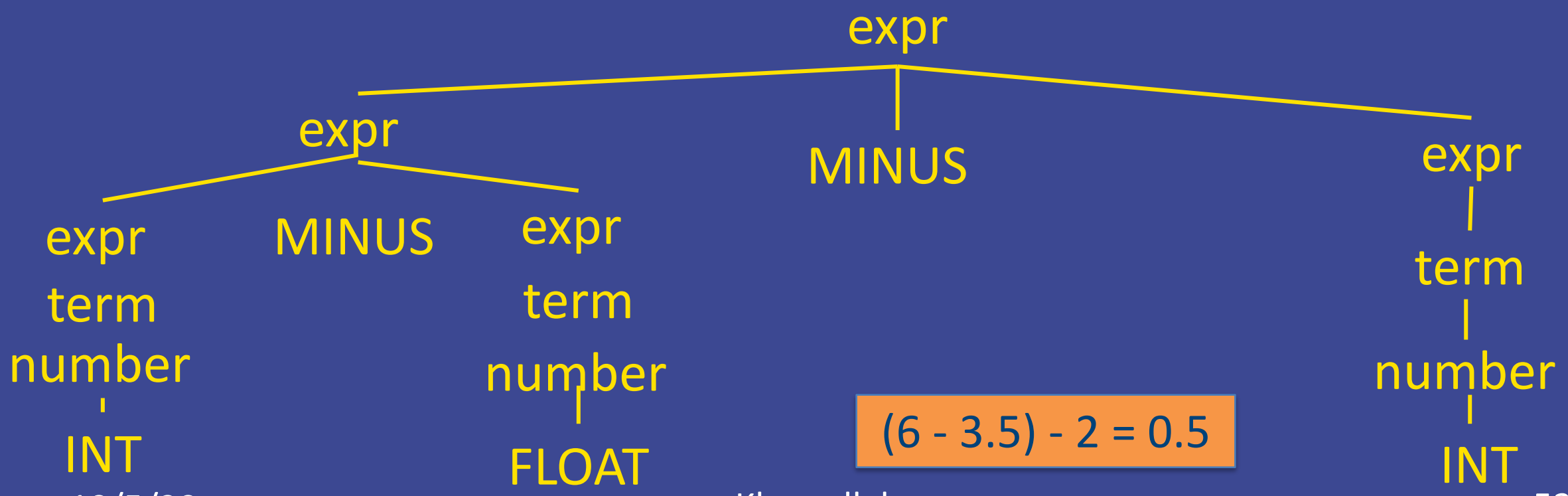


6 - (3.5 - 2) = 4.5

Subtraction?

<expr> → <expr> PLUS <expr> | <expr> MINUS <expr> | <term>
<term> → <term> TIMES <term> | LPAREN <expr> RPAREN | <number>
<number> → INT | FLOAT

INT MINUS FLOAT MINUS INT



(6 - 3.5) - 2 = 0.5

Associativity of + and -

<expr> → <expr> PLUS <expr> | <expr> MINUS <expr> | <term>
<term> → <term> TIMES <term> | LPAREN <expr> RPAREN |
<number>
<number> → INT | FLOAT

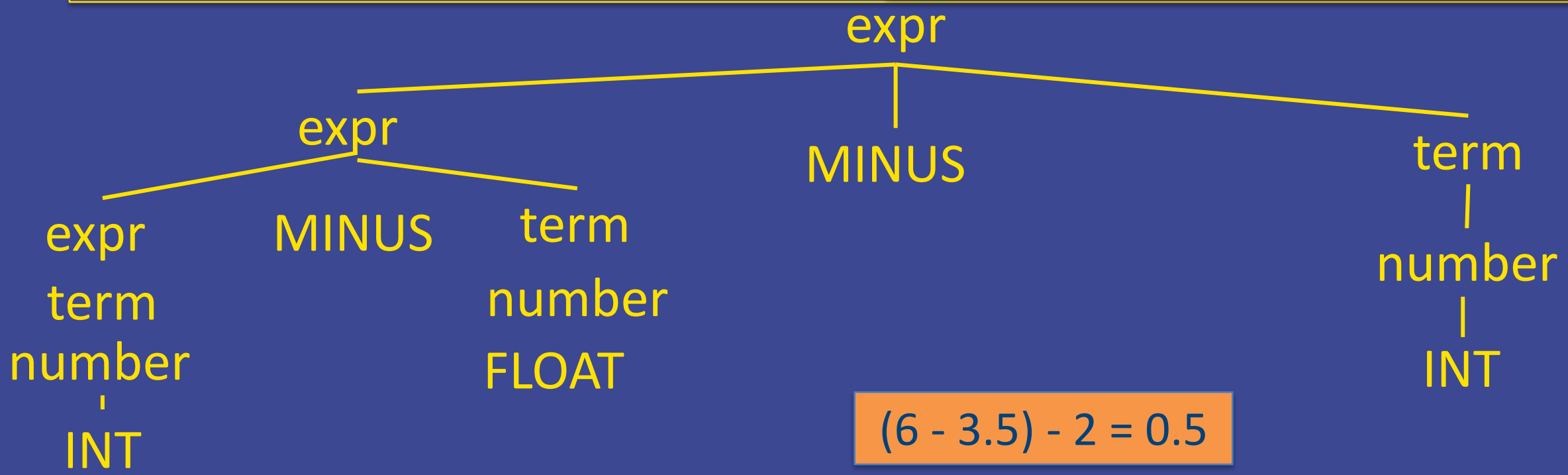
<expr> → <expr> PLUS <term> | <expr> MINUS <term> | <term>
<term> → <term> TIMES <term> | LPAREN <expr> RPAREN |
<number>
<number> → INT | FLOAT

- Left recursive => left associative

Left Associative Subtraction?

<expr> → <expr> PLUS <term> | <expr> MINUS <term> | <term>
<term> → <term> TIMES <term> | LPAREN <expr> RPAREN | <number>
<number> → INT | FLOAT

INT MINUS FLOAT MINUS INT



(6 - 3.5) - 2 = 0.5

Ambiguity?

<expr> → <expr> PLUS <term> | <expr> MINUS <term> | <term>
<term> → <term> TIMES <term> | LPAREN <expr> RPAREN |
<number>
<number> → INT | FLOAT

7 * 3 * 2

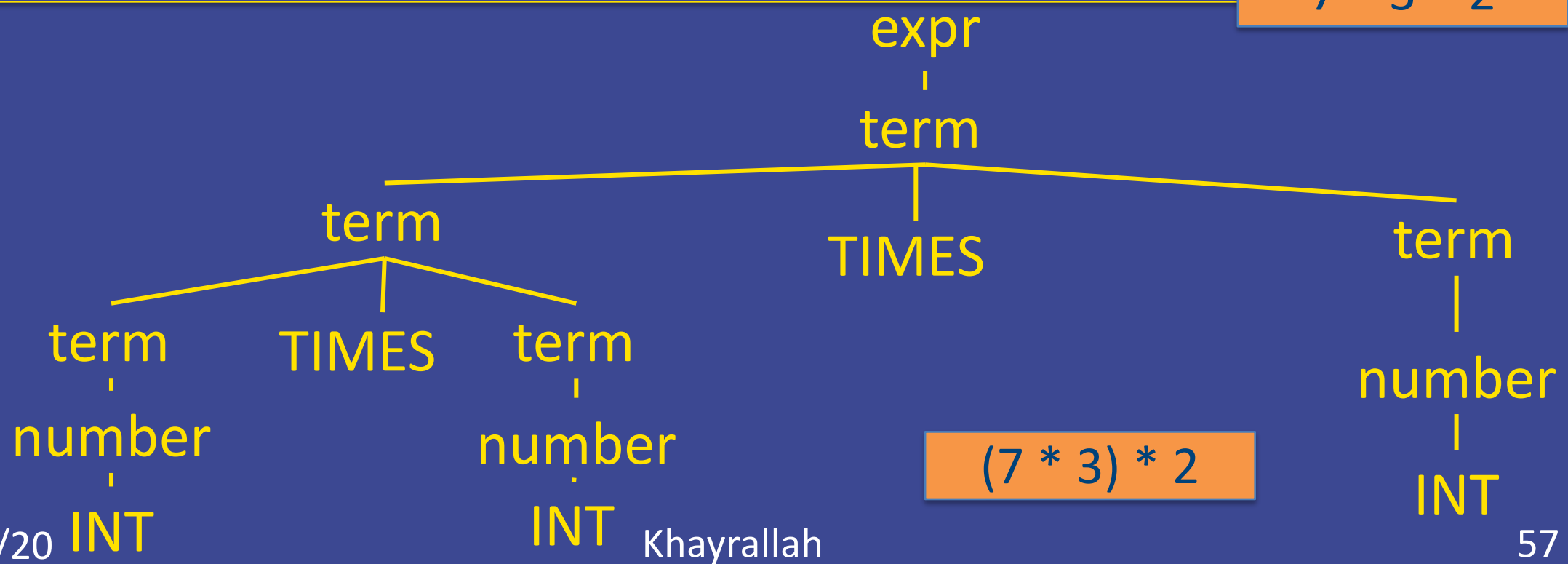
Is the grammar above ambiguous?

A. Yes
B. No
C. It depends

Ambiguity?

<expr> → <expr> PLUS <term> | <expr> MINUS <term> | <term>
<term> → <term> TIMES <term> | LPAREN <expr> RPAREN | <number>
<number> → INT | FLOAT

7 * 3 * 2

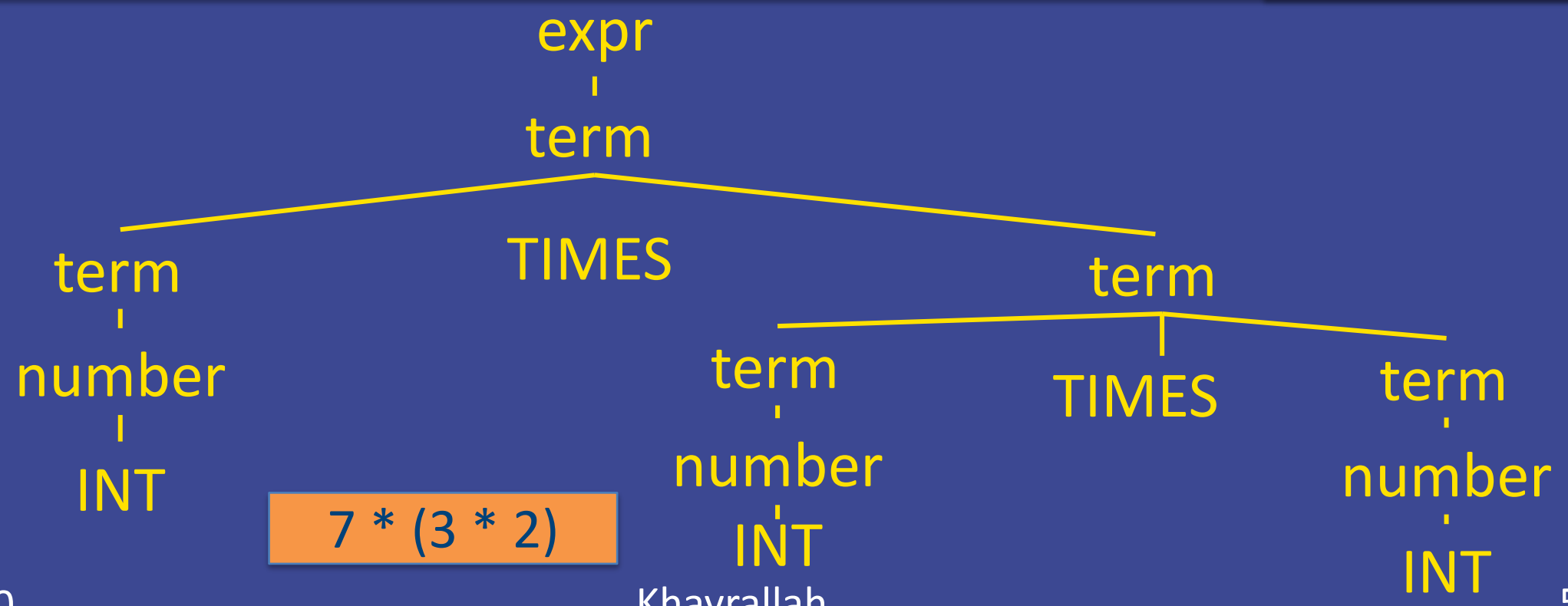


(7 * 3) * 2

Ambiguity?

<expr> → <expr> PLUS <term> | <expr> MINUS <term> | <term>
<term> → <term> TIMES <term> | LPAREN <expr> RPAREN | <number>
<number> → INT | FLOAT

7 * 3 * 2



7 * (3 * 2)

Ambiguity?

<expr> → <expr> PLUS <term> | <expr> MINUS <term> | <term>
<term> → <term> TIMES <term> | LPAREN <expr> RPAREN |
<number>
<number> → INT | FLOAT

7 * 3 * 2

The grammar is still ambiguous because there are two leftmost derivations for 7 * 3 * 2
(7 * 3) * 2 and 7 * (3 * 2)

Associativity of Multiplication

<expr> → <expr> PLUS <term> | <expr> MINUS <term> | <term>
<term> → <term> TIMES <term> | LPAREN <expr> RPAREN |
<number>
<number> → INT | FLOAT

<expr> → <expr> PLUS <term> | <expr> MINUS <term> | <term>
<term> → <term> TIMES <factor>
<factor> → LPAREN <expr> RPAREN | <number>
<number> → INT | FLOAT

- Left recursive => left associative

Unambiguous Grammar

<expr> → <expr> PLUS <term> | <expr> MINUS <term> | <term>
<term> → <term> TIMES <factor> | <factor>
<factor> → LPAREN <expr> RPAREN | <number>
<number> → INT | FLOAT

INT PLUS FLOAT TIMES INT

<expr>
⇒ <expr> PLUS <term>
⇒ <term> PLUS <term>
⇒ <factor> PLUS <term>
⇒ <number> PLUS <term>
⇒ INT PLUS <term>

Unambiguous Grammar

<expr> → <expr> PLUS <term> | <expr> MINUS <term> | <term>
<term> → <term> TIMES <factor> | <factor>
<factor> → LPAREN <expr> RPAREN | <number>
<number> → INT | FLOAT

INT PLUS FLOAT TIMES INT

⇒ INT PLUS <term>
⇒ INT PLUS <term> TIMES <factor>
⇒ INT PLUS <factor> TIMES <factor>
⇒ INT PLUS <number> TIMES <factor>
⇒ INT PLUS FLOAT TIMES <factor>
⇒ INT PLUS FLOAT TIMES <number>
⇒ INT PLUS FLOAT TIMES INT

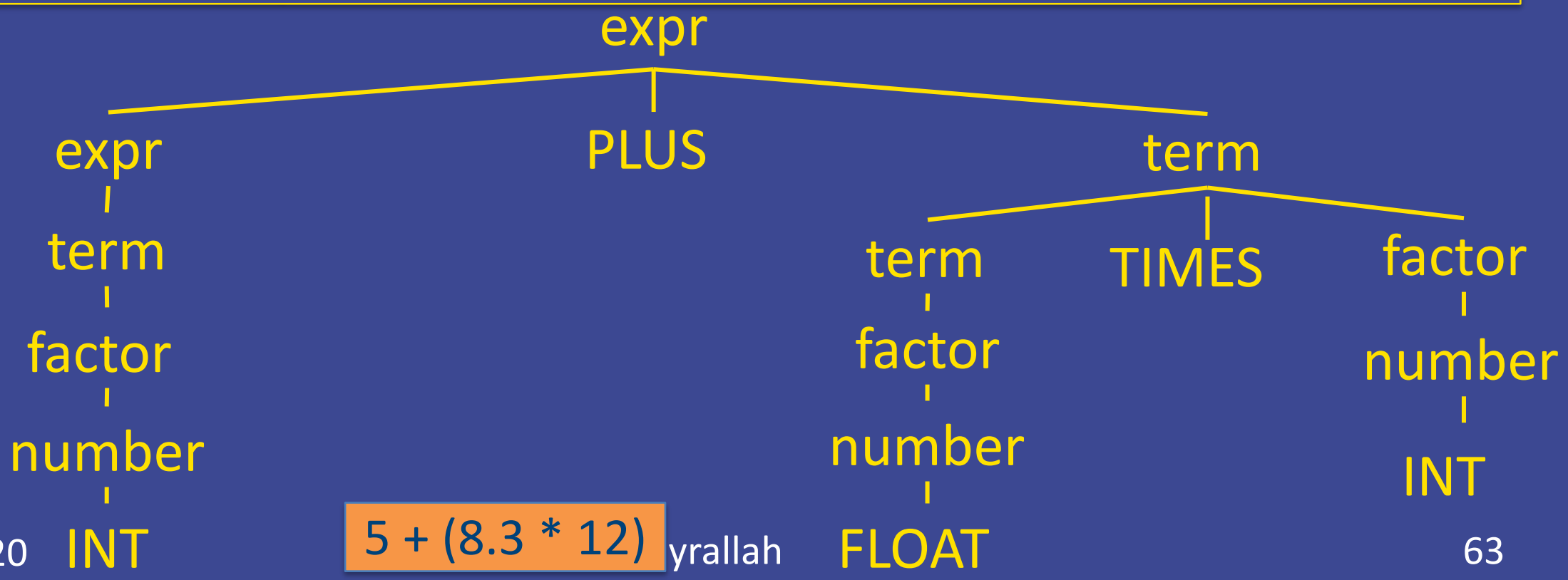
- More complex
- Longer derivations

Unambiguous Grammar

<expr> → <expr> PLUS <term> | <expr> MINUS <term> | <term>
<term> → <term> TIMES <factor> | <factor>
<factor> → LPAREN <expr> RPAREN | <number>
<number> → INT | FLOAT

5 + 8.3 * 12

INT PLUS FLOAT TIMES INT



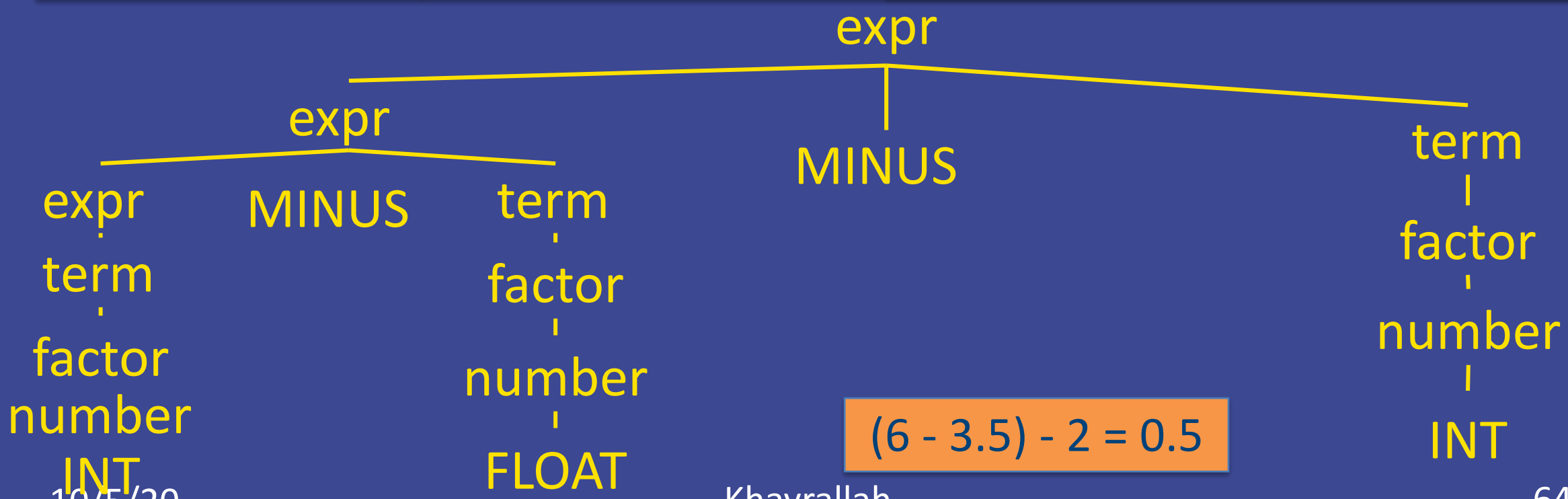
5 + (8.3 * 12)

Unambiguous Grammar

<expr> → <expr> PLUS <term> | <expr> MINUS <term> | <term>
<term> → <term> TIMES <factor> | <factor>
<factor> → LPAREN <expr> RPAREN | <number>
<number> → INT | FLOAT

6 - 3.5 - 2

INT MINUS FLOAT MINUS INT



iClicker

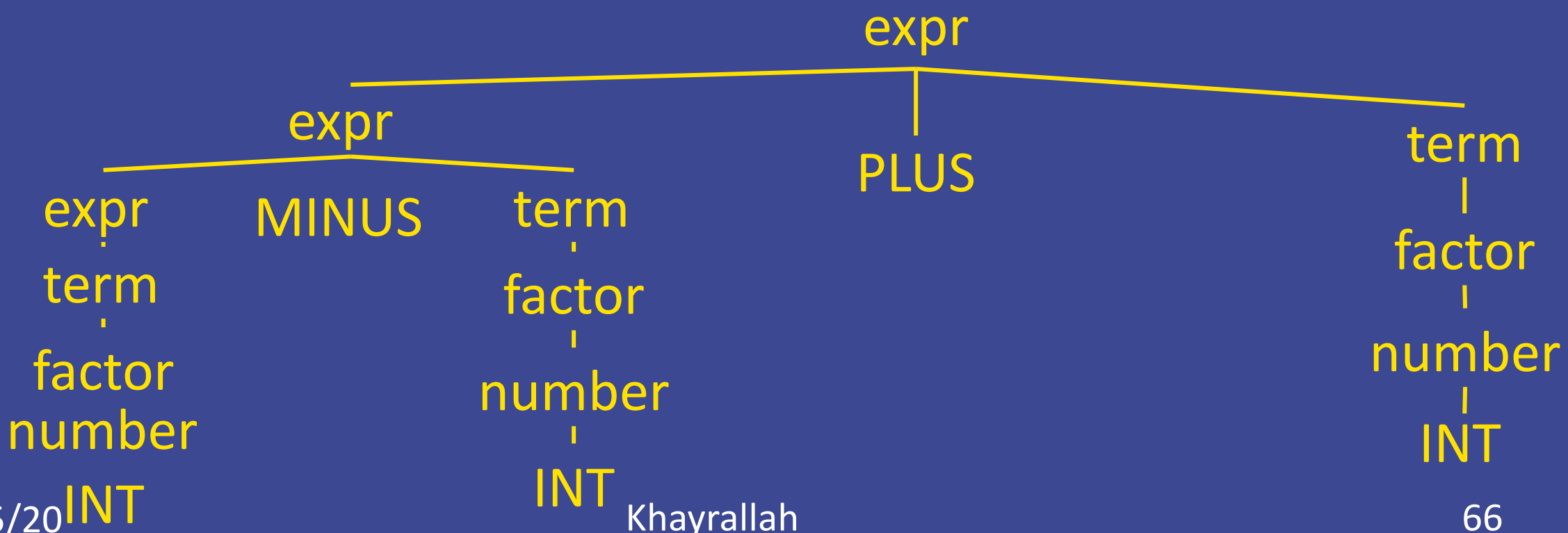
<expr> → <expr> PLUS <term> | <expr> MINUS <term> | <term>
<term> → <term> TIMES <factor> | <factor>
<factor> → LPAREN <expr> RPAREN | <number>
<number> → INT | FLOAT

Based on this grammar, what is the value of: 12 - 5 + 3?

12 - 5 + 3

<expr> → <expr> PLUS <term> | <expr> MINUS <term> | <term>
<term> → <term> TIMES <factor> | <factor>
<factor> → LPAREN <expr> RPAREN | <number>
<number> → INT | FLOAT

INT MINUS INT PLUS INT



MUFL 1.0

Valid Expressions:

- (+ 10)
- (- 10 1 2 3)
- (+ (* 2 3) 5 6 7 (- 8 2))
- 5

Invalid Expressions

- (+)
- 5 + 3

MUFL 1.0

<expr> -> OPENPAREN OPERATOR <operands> CLOSEPAREN |
NUMBER

<operands> -> <expr> | <expr> <operands>