Quiz-6-CFG-AMBIG-PREC

8 questions

Quiz

There are certain patterns that regular expressions can express that context-free grammars cannot express. But it is not an issue because those patterns do not show up in practice

- ○ True
- ○ False

Any RE can be expressed in BNF

- We just need to show fundamental operators
  - concat, choice, star

Any RE can be expressed in BNF

- We just need to show fundamental operators
  - concat, choice, star

```
add_expr ::= NUM '+' NUM
```

Any RE can be expressed in BNF

- We just need to show fundamental operators
  - concat, choice, star

```
simple_expr ::= NUM '+' NUM
              |   NUM '*' NUM
```

Any RE can be expressed in BNF

- We just need to show fundamental operators
  - concat, choice, star

How to express “a” in BNF?*

```
a_star ::= “”
         |  “a” a_star
```

Any RE can be expressed in BNF

- We just need to show fundamental operators
  - concat, choice, star

How to express "a" in BNF?*

```
a_star ::= ""
       |   "a"
       |   "a" a_star
```

Quiz

A production rule consists of:

- [ ] Terminals
- [ ] Regular Expressions
- [ ] Non-terminals
- [ ] function calls

Context-free grammar

We will use *Backus–Naur form* (BNF) form

- Production rules contain a sequence of either non-terminals or terminals
- In our class, terminals will either be string constants or tokens
- Traditionally tokens will be all caps.

```
Examples:

add_expr ::= NUM ‘+’ NUM

mult_expr ::= NUM ‘*’ NUM

joint_expr ::= add_expr ‘*’ add_expr

simple_expr ::= NUM ‘+’ NUM
              |   NUM ‘*’ NUM
```

Quiz

a left derivation will always produce the same parse tree as a right derivation

- ○ True
- ○ False

Not discussed yet. This is TRUE if the Grammar is unambiguous. The trees will create different sentential forms at different steps, but the tree will always be uniqud if the grammar is unambiguous. To be discussed in class.

Quiz

Different programming languages make structure more or less explicit, e.g. using ()s and {}s.

Write a few sentences on any programming language experience you have w.r.t. structure and how you use it. For example do you use {}s when you write if statements, even if they contain a single statement? Why or Why not? Do you think Python's use of whitespace is a good construct for structure? Have you ever used S expressions in a Lisp language?

Programming language structure

```
if (x) {
  my_var++;
 }
```

vs.

```
if (x)
  my_var++;
```

Should conditionals require braces?

```
5 + 6 * 3
```

vs.

```
5 + (6 * 3)
```

should expressions require parenthesis?

```
(+ 5 (* 6 3))
```

vs.

```
(+ 5 (* 6 3))
```

S expressions (lisp) require explicit structure

What are pros and cons of each?

Quiz: Top Down Parsing

Quiz

What is an example of input recognized by the following grammar?

$a \rightarrow a\ X$

$a \rightarrow Y$

- ○ XXXXXXXXY
- ○ XYYYYYYYY
- ○ YXXXXXXXX
- ○ YYYYYYYYX

Quiz

What is an example of input recognized by the following grammar?

1. $a \rightarrow a\ X$
2. $a \rightarrow Y$

How about this one? `XXXXXXXY`

RULE	Sentential Form
start	a

Quiz

What is an example of input recognized by the following grammar?

1 $a \rightarrow a\ X$

2 $a \rightarrow Y$

How about this one? `XXXXXXXY`

RULE	Sentential Form
start	a

Applying either rule gives us a sentential form that won't create the string

Quiz

What is an example of input recognized by the following grammar?

1. $a \rightarrow a\ X$
2. $a \rightarrow Y$

How about this one? `XYYYY`

RULE	Sentential Form
start	A

Quiz

What is an example of input recognized by the following grammar?

1 a → a X

2 a → Y

How about this one? `XYYYY`

RULE	Sentential Form
start	A

Similar reason:
strings that are
longer than 1 character
cannot end in y

Quiz

What is an example of input recognized by the following grammar?

1. $a \rightarrow a\ X$
2. $a \rightarrow Y$

How about this one? `YXXXXXX`

RULE	Sentential Form
start	A

Quiz

What is an example of input recognized by the following grammar?

1 $a \rightarrow a\ X$

2 $a \rightarrow Y$

How about this one? `YXXXXXX`

RULE	Sentential Form
start	A
1	aX
1	aXX
...	....
2	YXXXXX

Quiz

What is an example of input recognized by the following grammar?

1. $a \rightarrow a\ X$
2. $a \rightarrow Y$

How about this one? `YYYYYYX`

RULE	Sentential Form
start	A

Quiz

What is an example of input recognized by the following grammar?

1 $a \rightarrow a\ X$

2 $a \rightarrow Y$

How about this one? `YYYYYYX`

We can only produce 1 y, so we cannot derive this string

RULE	Sentential Form
start	A

Quiz

Which grammar is ambiguous?

(a)
e → e PLUS e
e → ID

(b)
e → e PLUS ID
e → ID

(c)
e → ID PLUS e
e → ID

(d)
e → ID PLUS ID
e → ID

Quiz

Which grammar is ambiguous?

(a)
e → e PLUS e
e → ID

(b)
e → e PLUS ID
e → ID

(c)
e → ID PLUS e
e → ID

(d)
e → ID PLUS ID
e → ID

Let’s look at some examples.

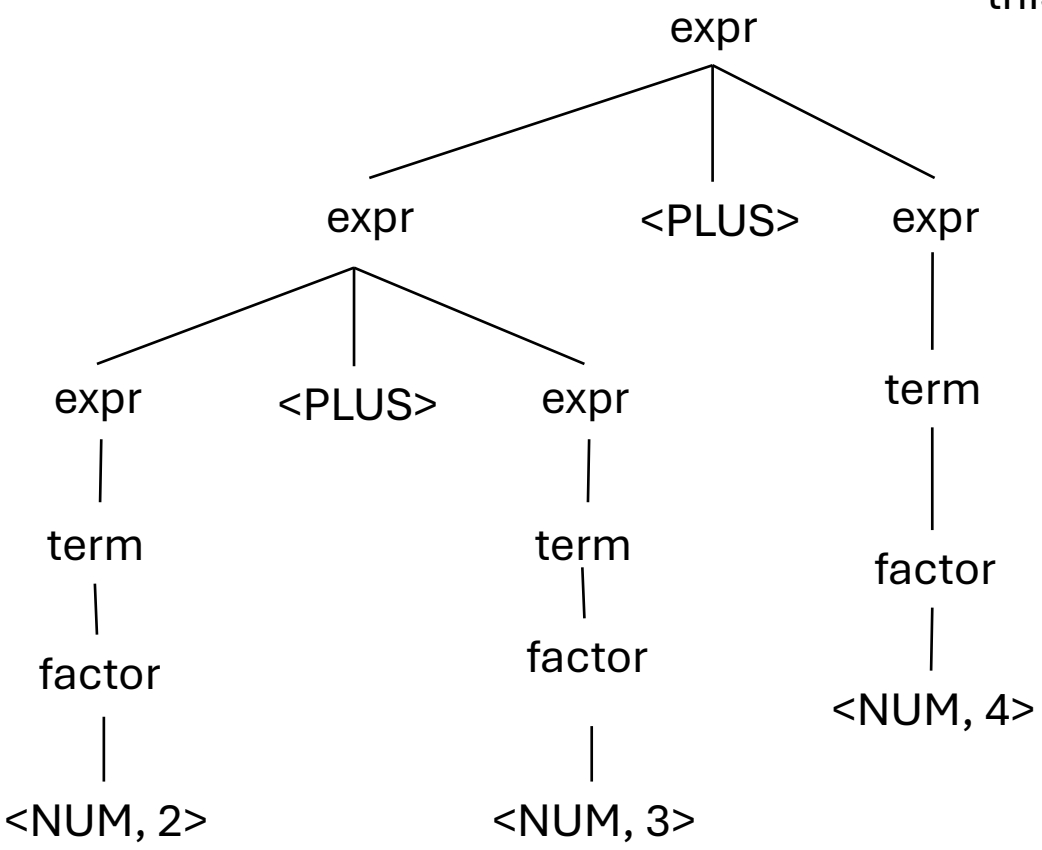
Let’s assume that E is an ”expr” and x is a number

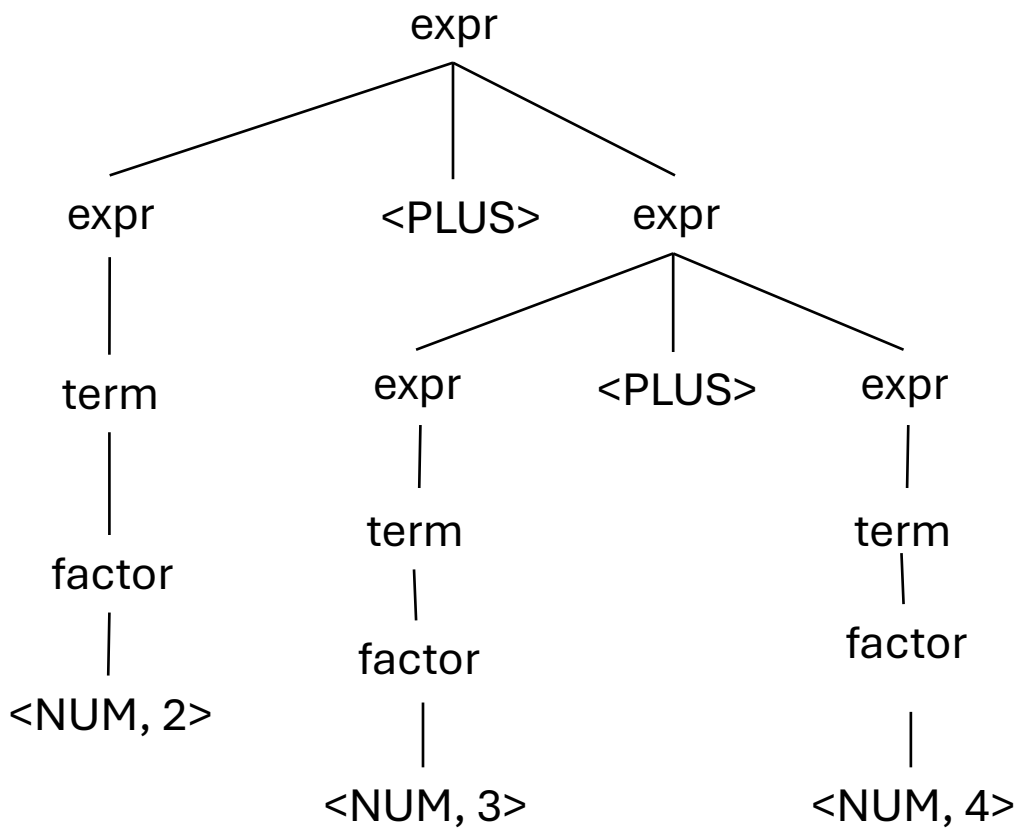
e → e PLUS e

e → ID

```
input: 2+3+4
```

Both parse trees are valid,
this grammar is ambiguous





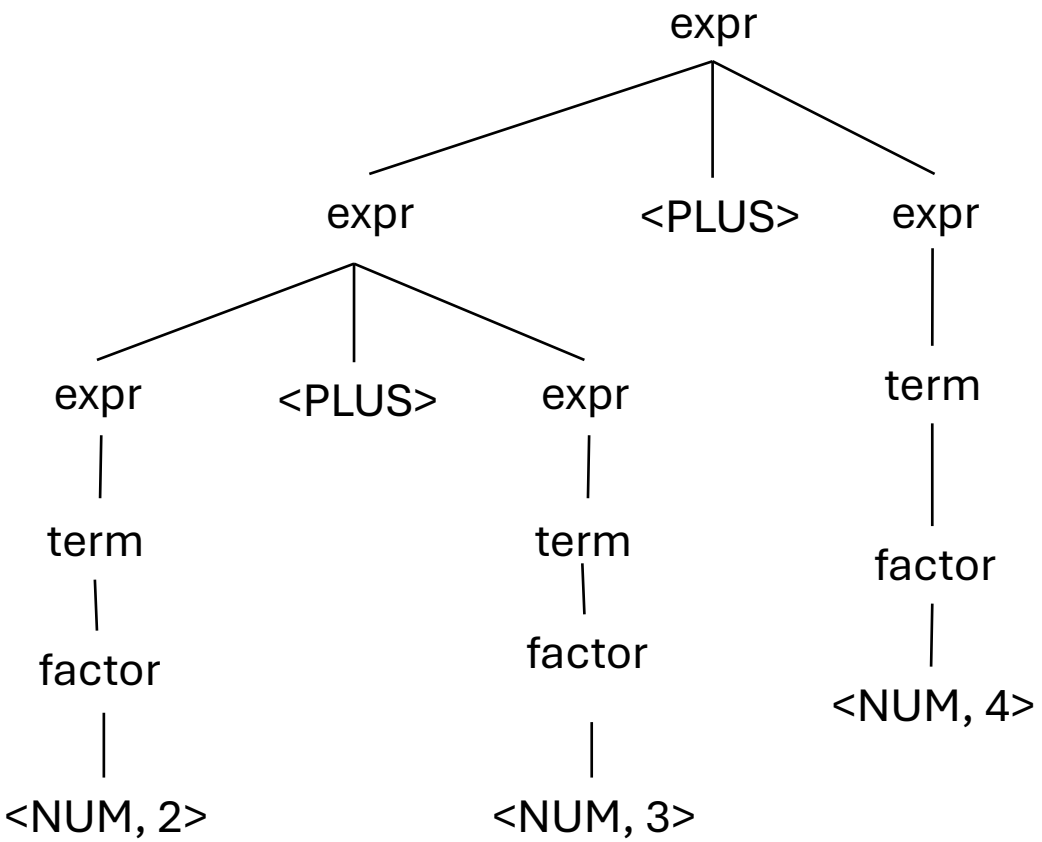
e → e PLUS ID

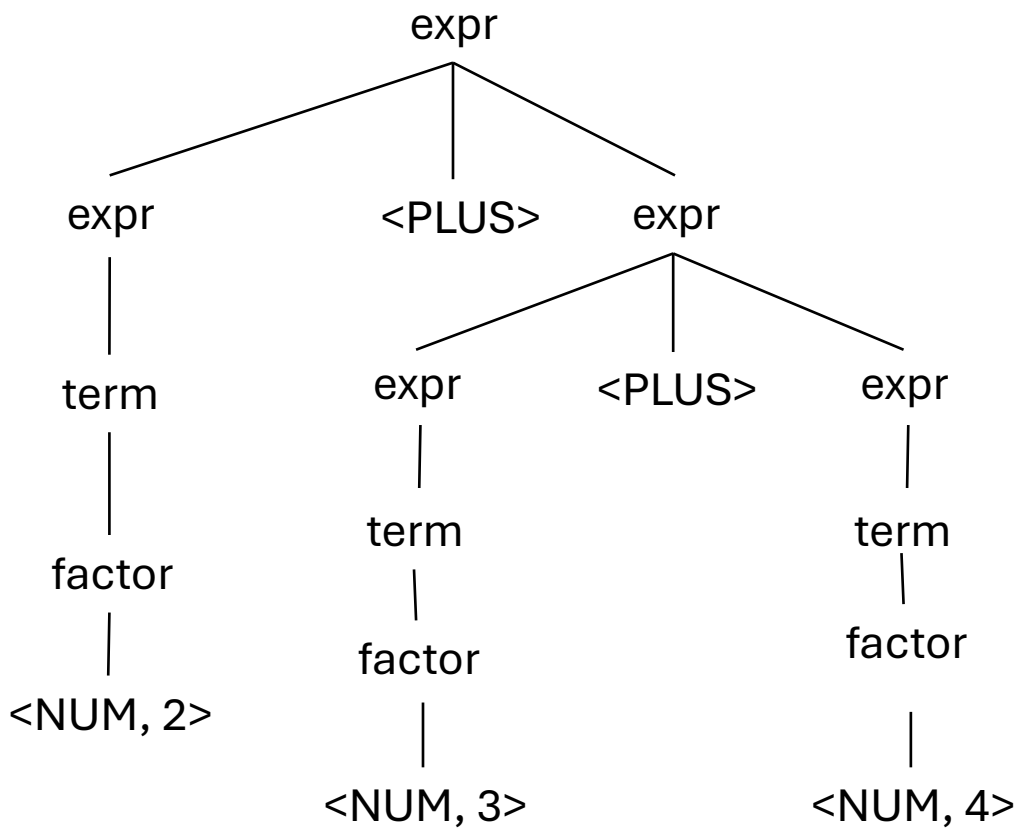
e → ID

`input: 2+3+4`

See next slide

What about this one?





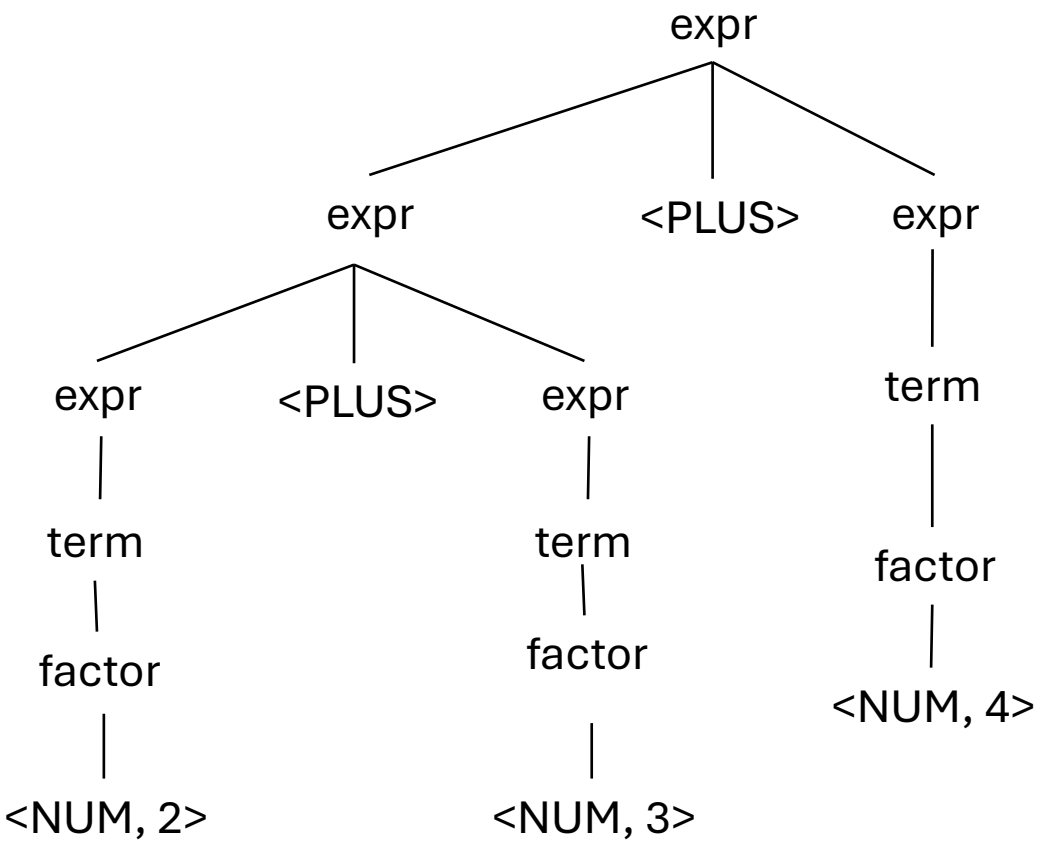
e → e PLUS ID

e → ID

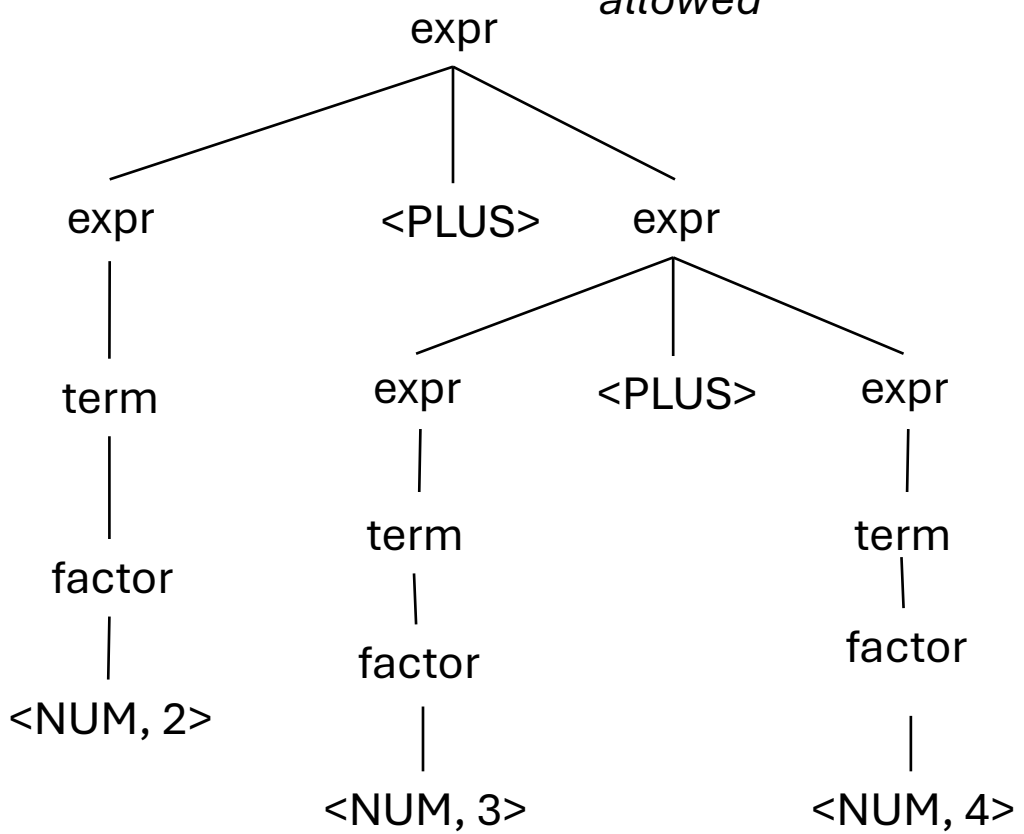
`input: 2+3+4`

UNAMBIGUOUS

What about this one?



Doesn't allow an expression on the RHS. This parse tree is not allowed



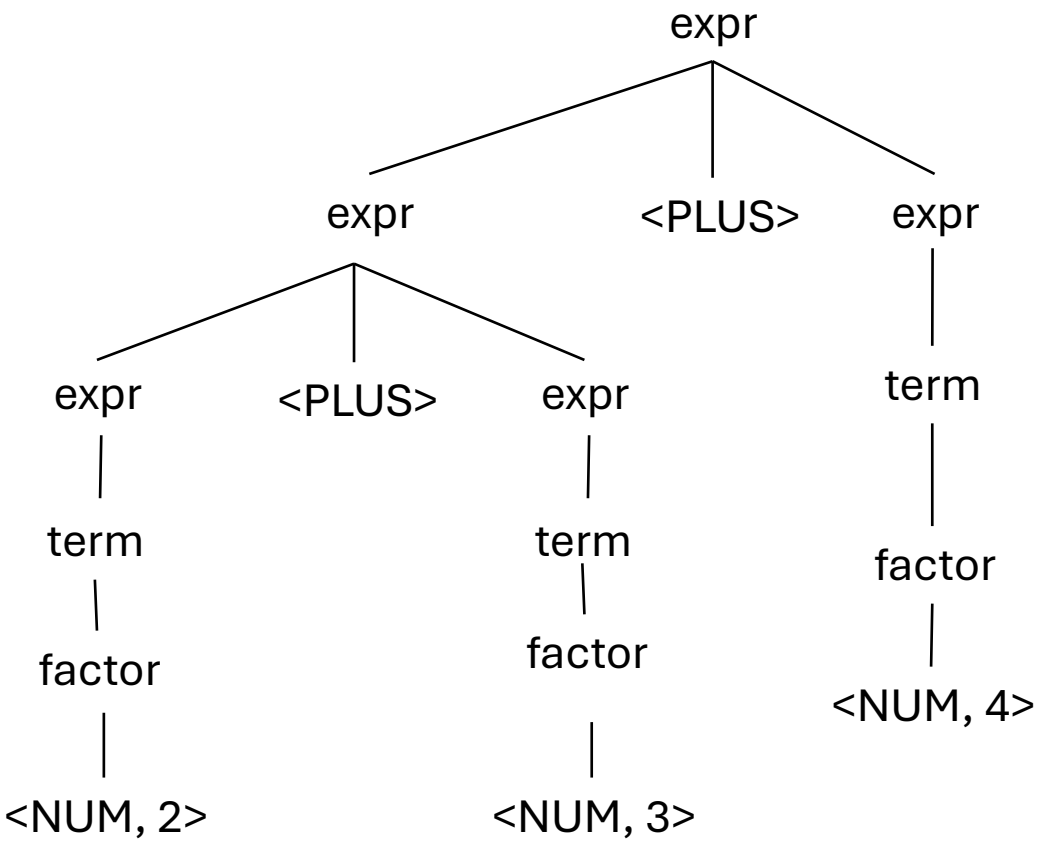
e → ID PLUS e

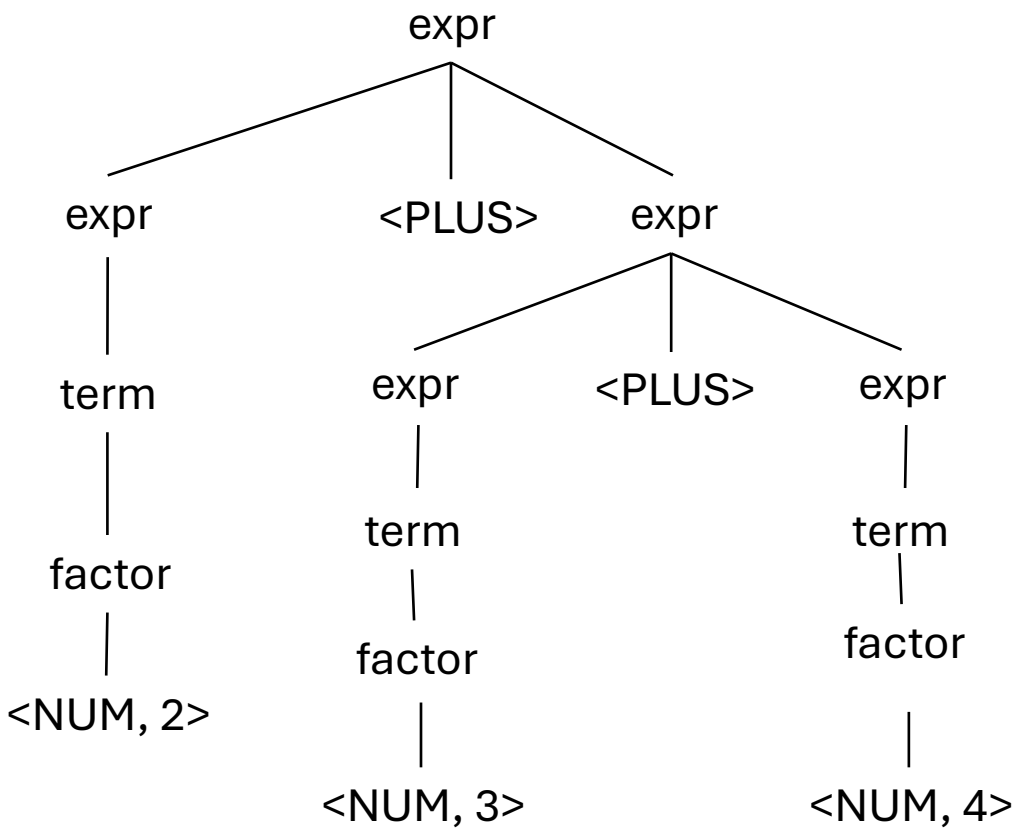
e → ID

`input: 2+3+4`

See next slide

What about this one?





e → ID PLUS e

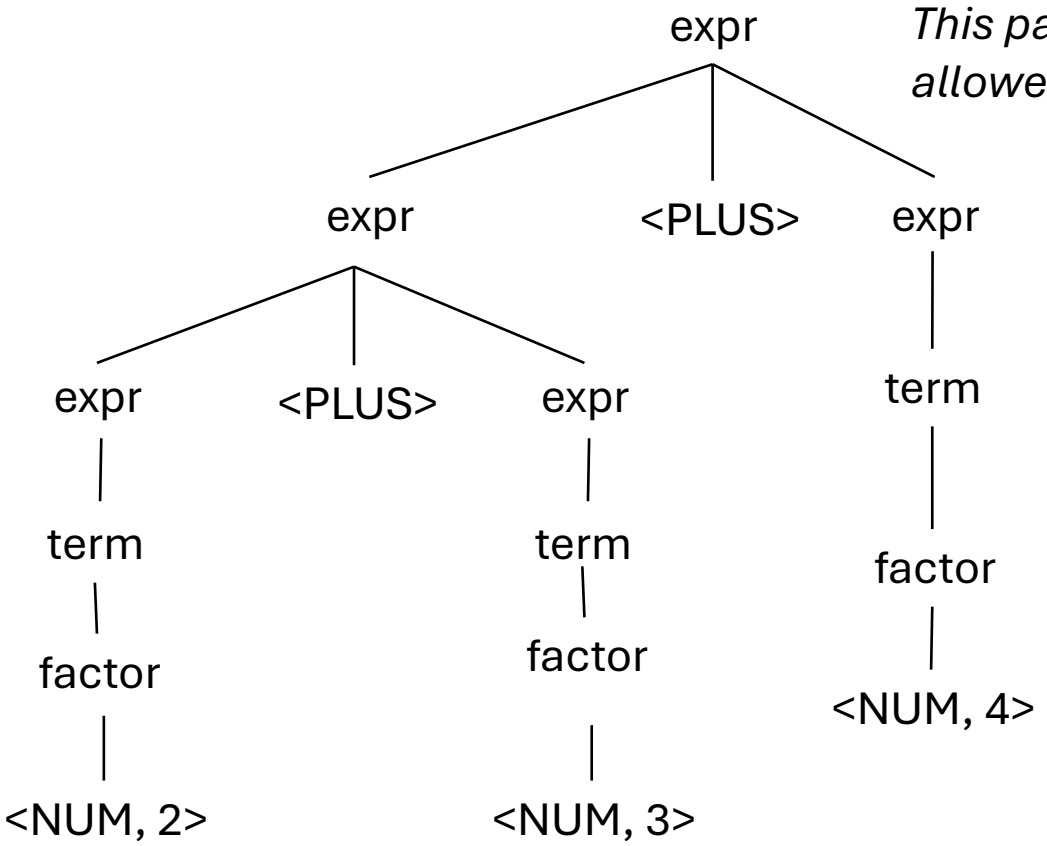
e → ID

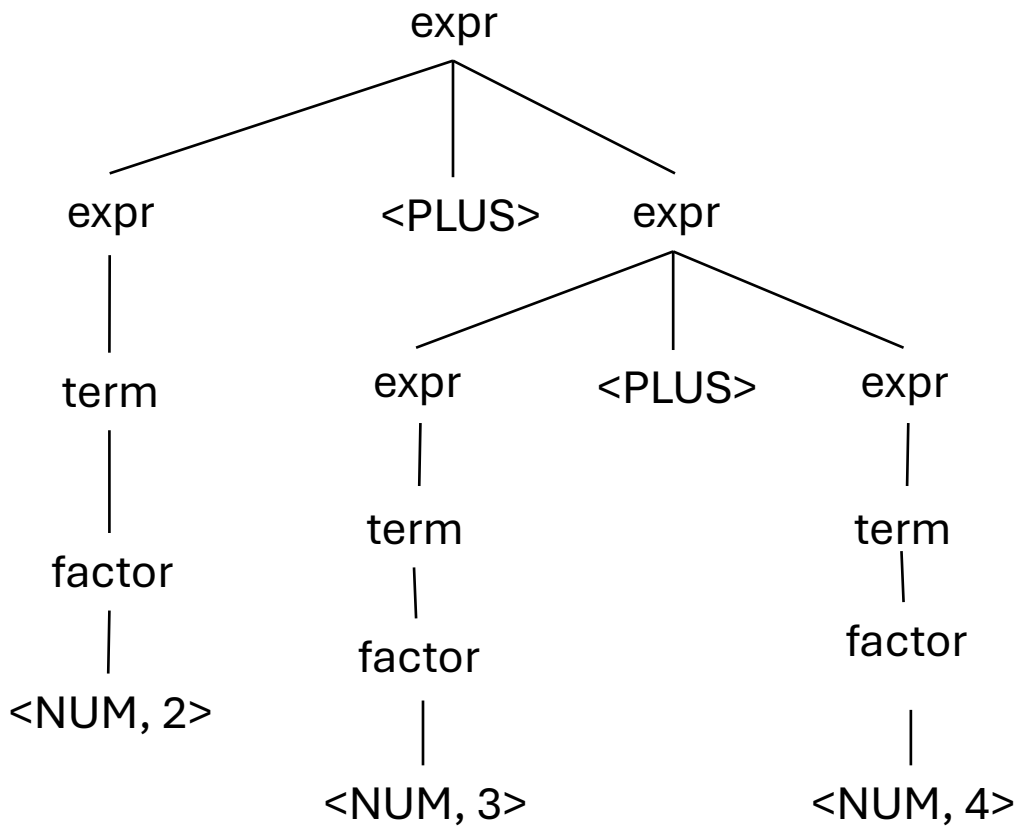
`input: 2+3+4`

UNAMBIGUOUS

What about this one?

Doesn't allow an expression on the LHS. This parse tree is not allowed





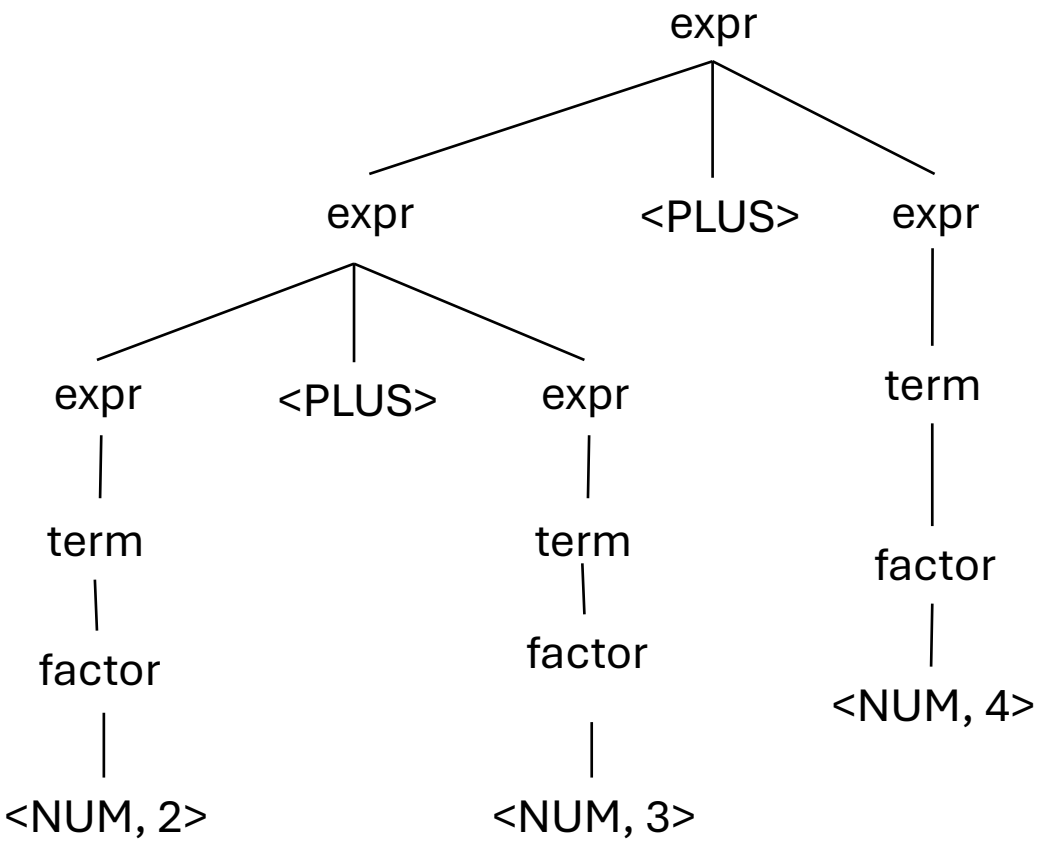
e → ID PLUS ID

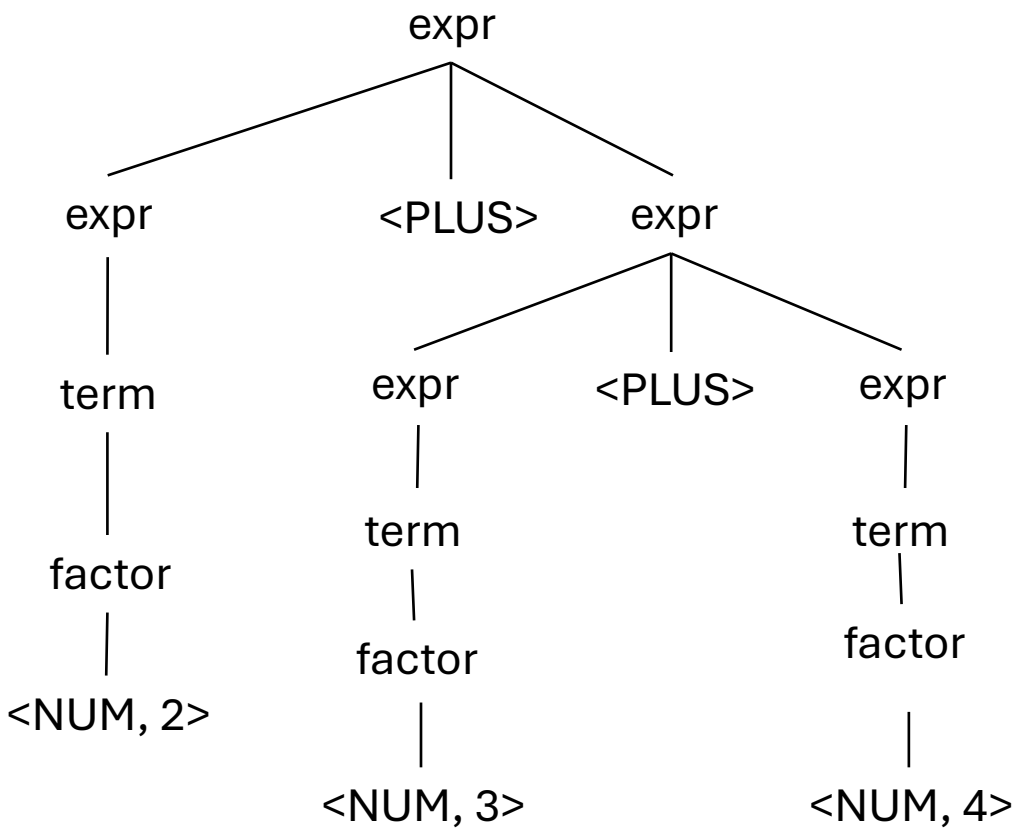
e → ID

`input: 2+3+4`

See next slide

What about this one?





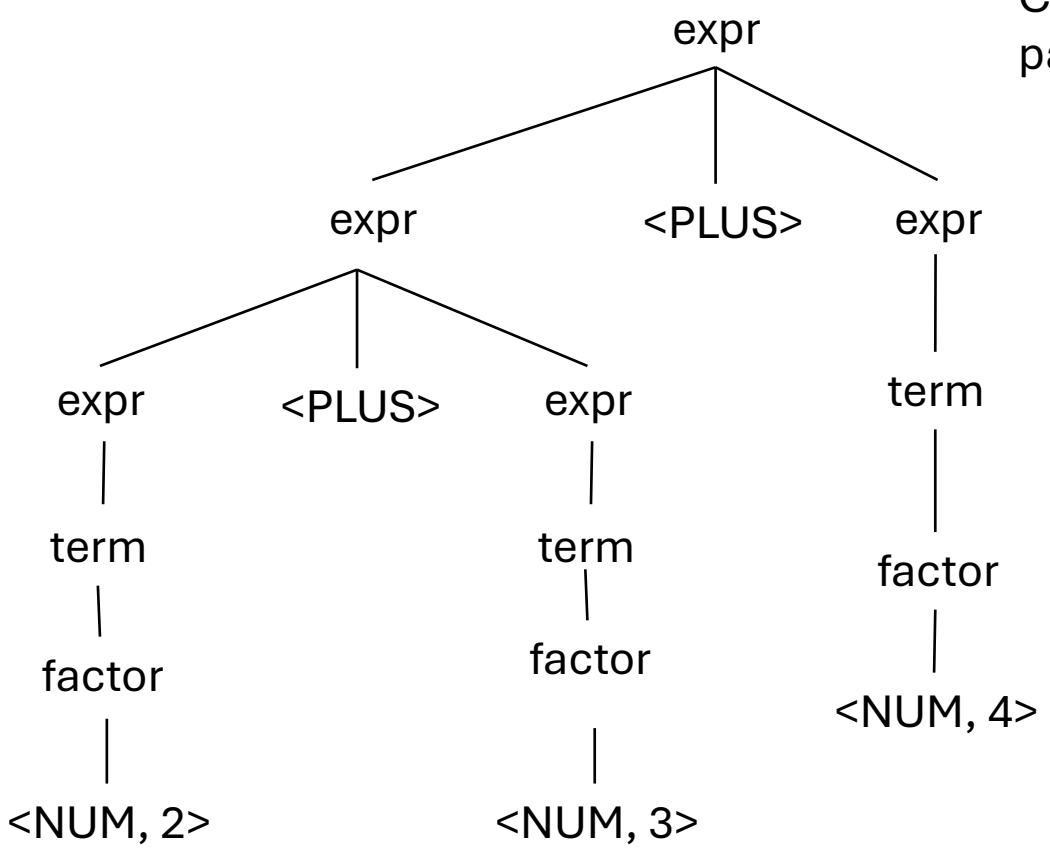
e → ID PLUS ID

e → ID

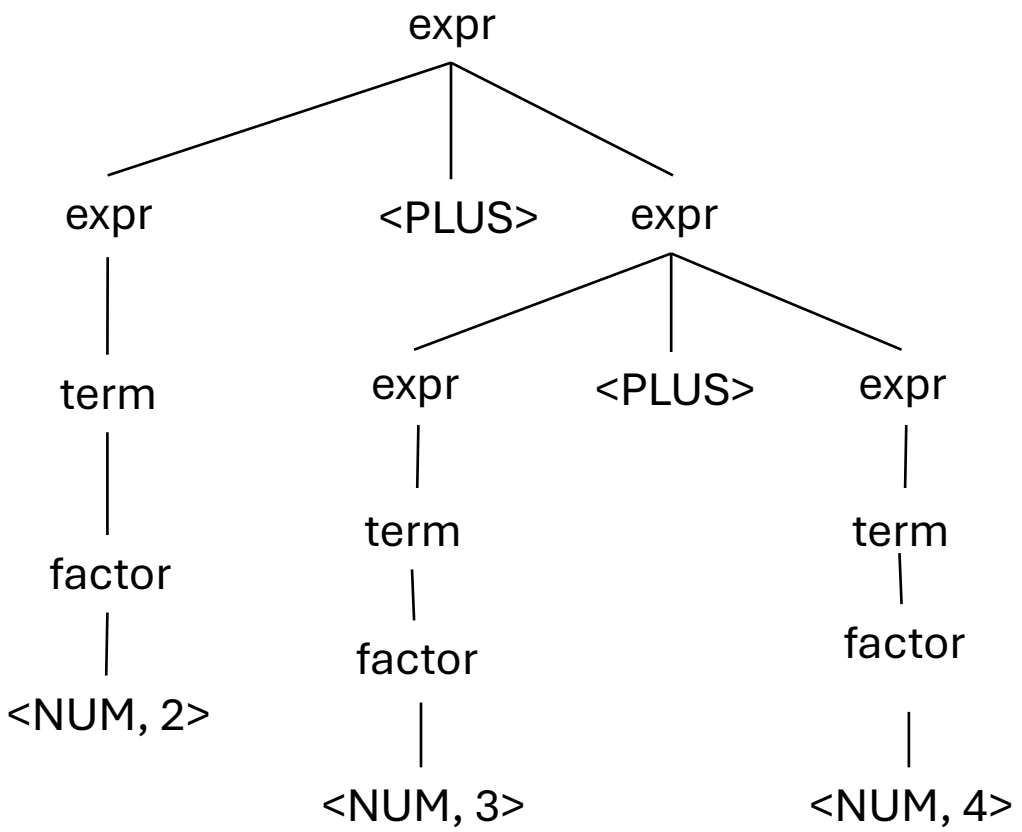
`input: 2+3+4`

UNAMBIGUOUS
But not expressive
No recursion

What about this one?



Cannot produce either parse tree!

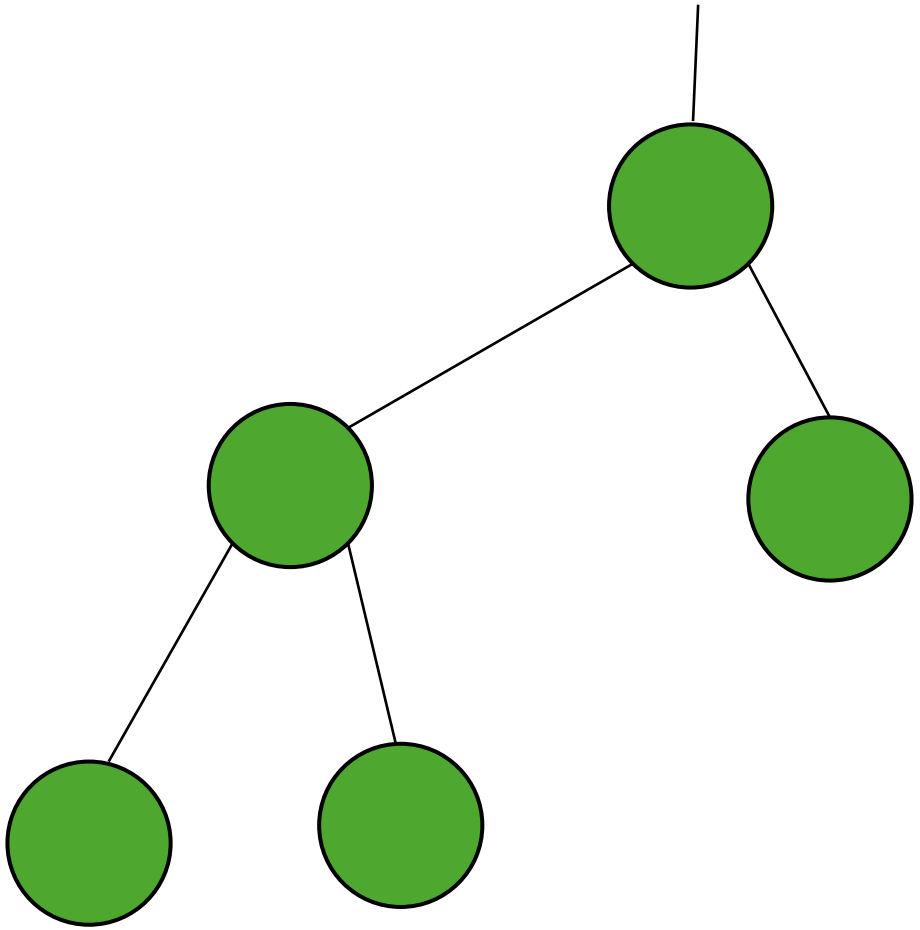


Quiz

operators with higher precedence should appear in production rules that appear higher in the parse tree

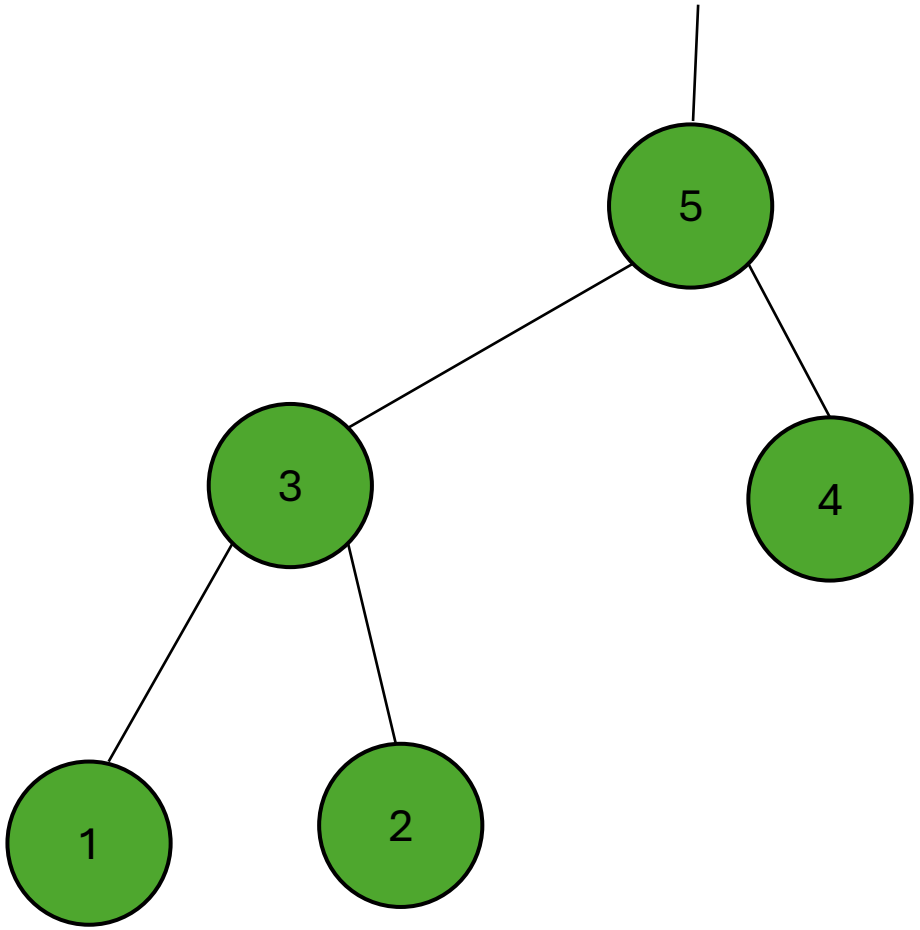
- ◯ True
- ◯ False

Post order traversal



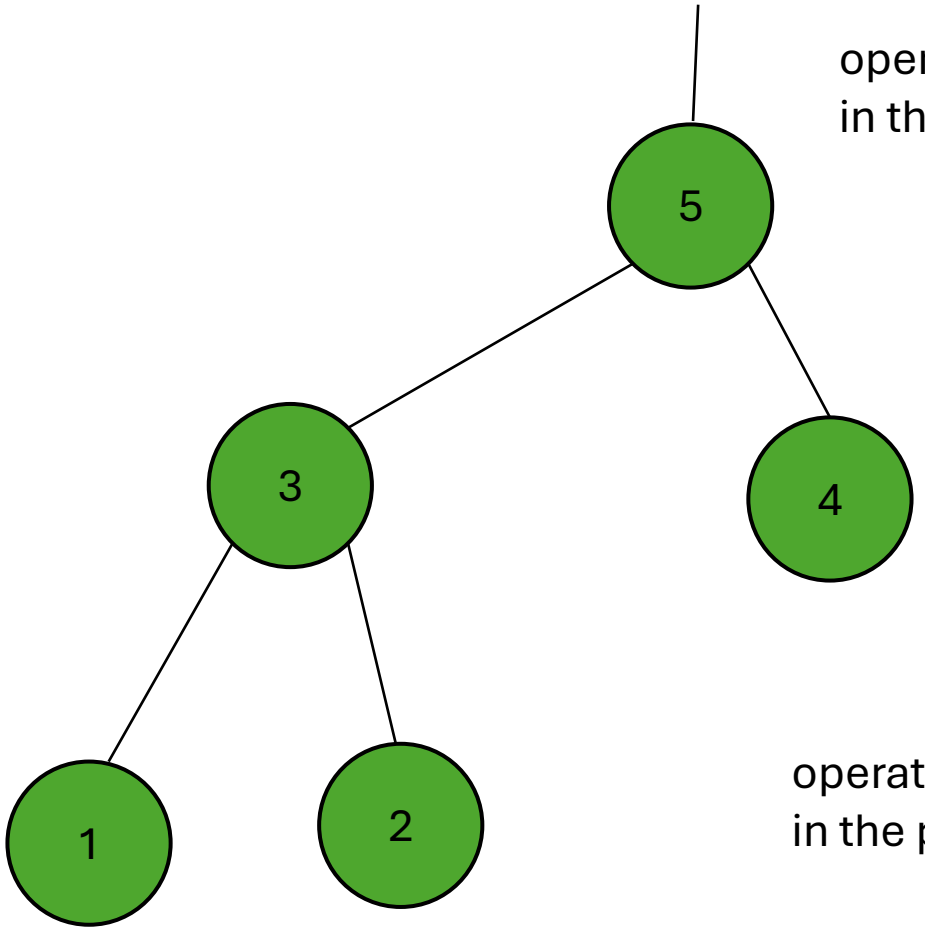
What is the post order traversal of this tree?

Post order traversal



What is the post order traversal of this tree?

Post order traversal



operators that are higher
in the parse tree get evaluated later

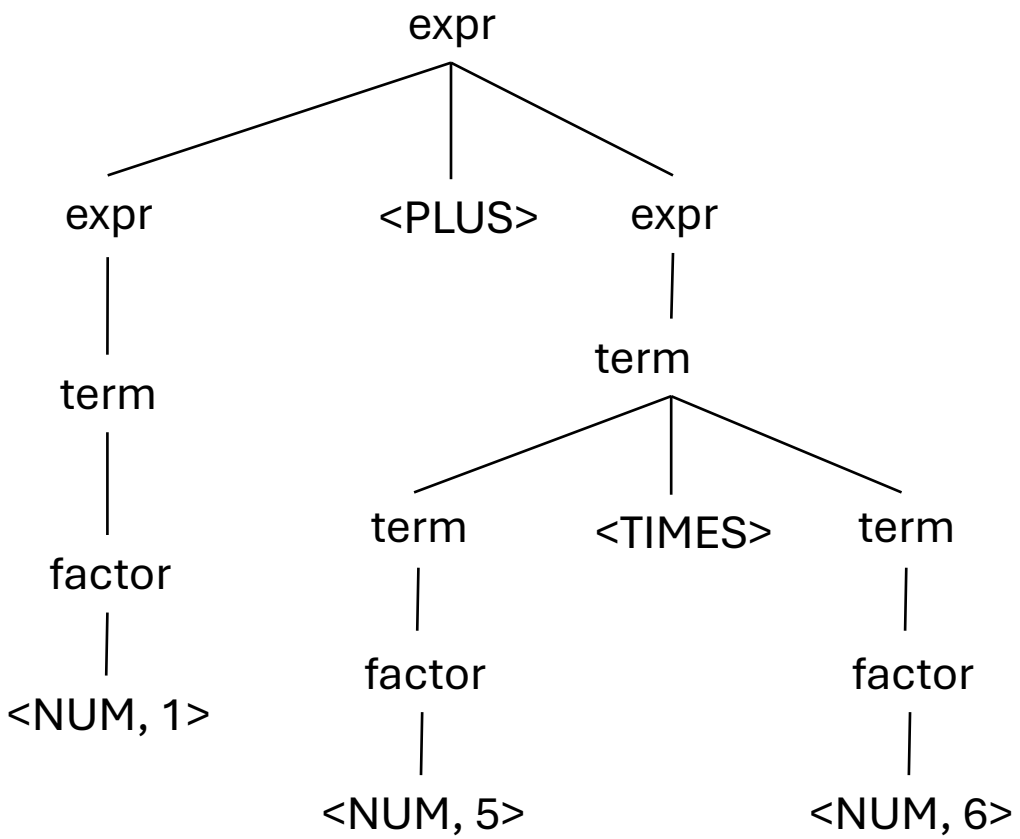
What is the post order traversal
of this tree?

operators that are lower
in the parse tree get evaluated earlier

Evaluating a parse tree

Operator	Name	Productions
+	`expr`	`: expr PLUS expr` `| term`
*	`term`	`: term TIMES term` `| factor`
()	`factor`	`: LPAREN expr RPAREN` `| NUM`

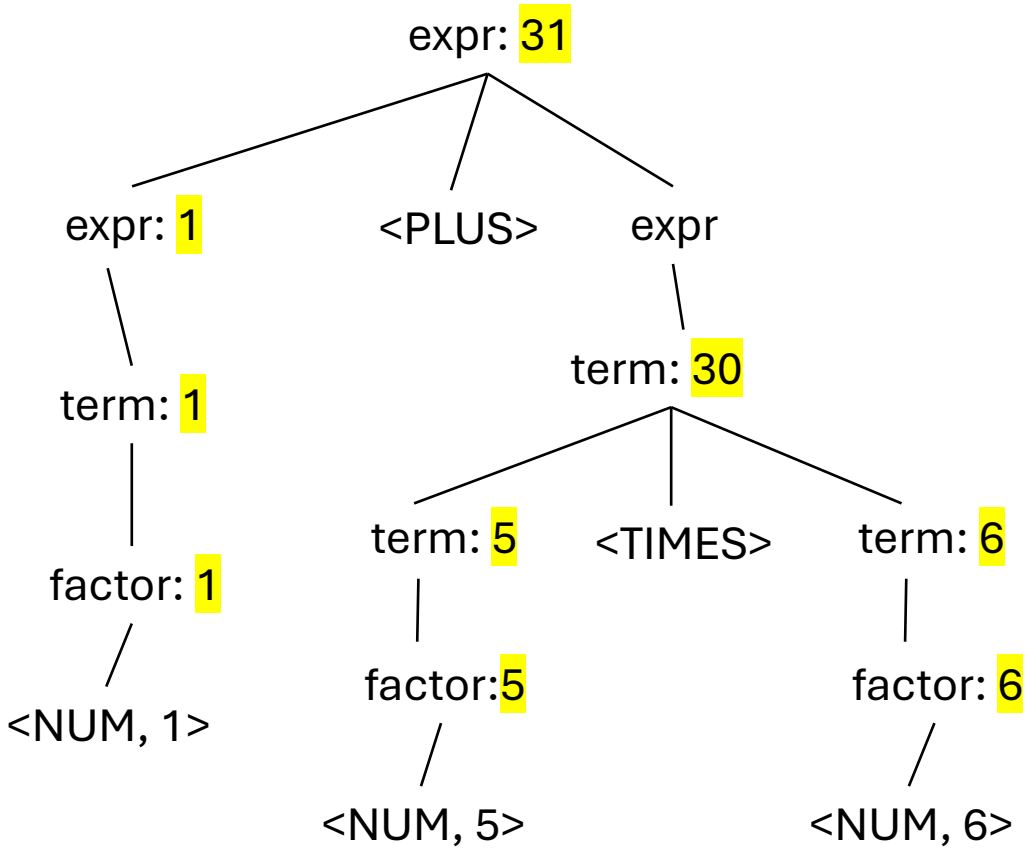
`input: 1+5*6`



Evaluating a parse tree

Operator	Name	Productions
+	`expr`	`: expr PLUS expr` `| term`
*	`term`	`: term TIMES term` `| factor`
()	`factor`	`: LPAREN expr RPAREN` `| NUM`

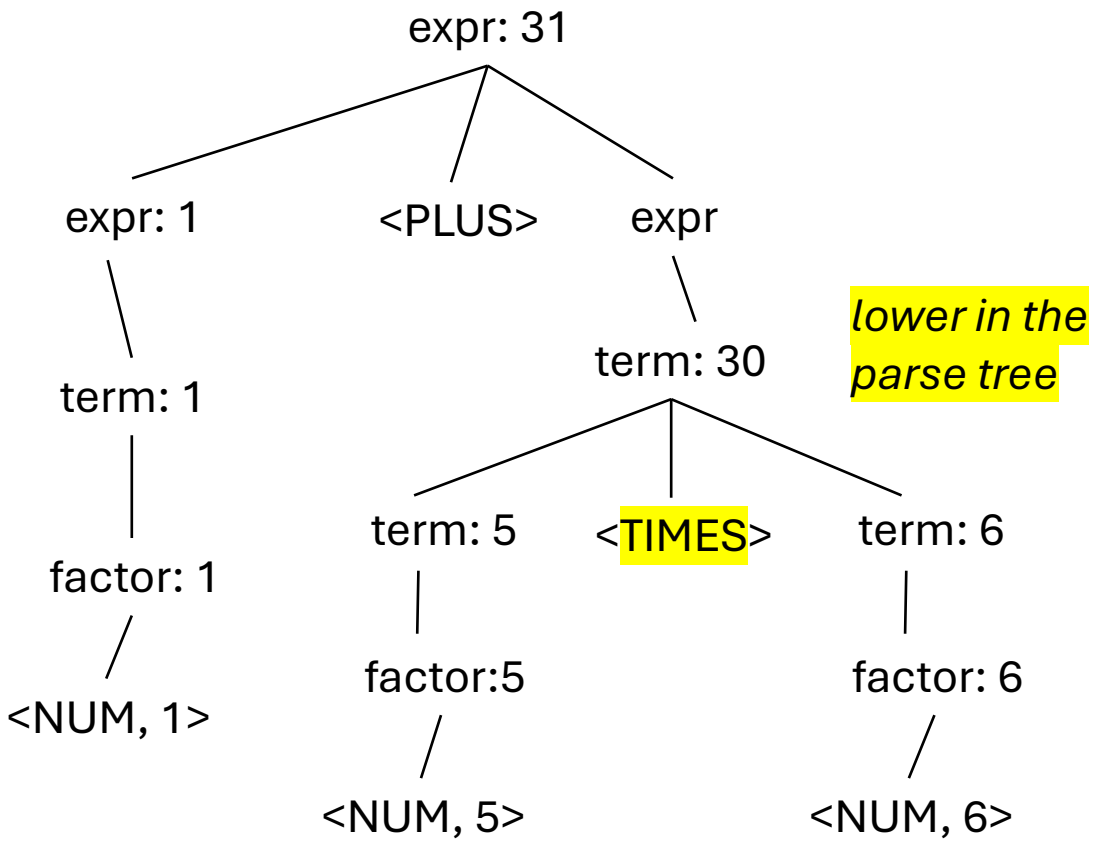
`input: 1+5*6`



Evaluating a parse tree

Operator	Name	Productions
+	expr	: expr PLUS expr | term
*	term	: term TIMES term | factor
()	factor	: LPAREN expr RPAREN | NUM

input: 1+5*6



Avoiding Ambiguity

- new production rules
  - One non-terminal for each level of precedence
  - lowest precedence at the top
  - highest precedence at the bottom
- How would we add power? ^

See this: See this

Precedence increases going down

Operator	Name	Productions
+	`expr`	`: expr PLUS expr` `| term`
*	`term`	`: term TIMES term` `| factor`
()	`factor`	`: LPAREN expr RPAREN` `| NUM`

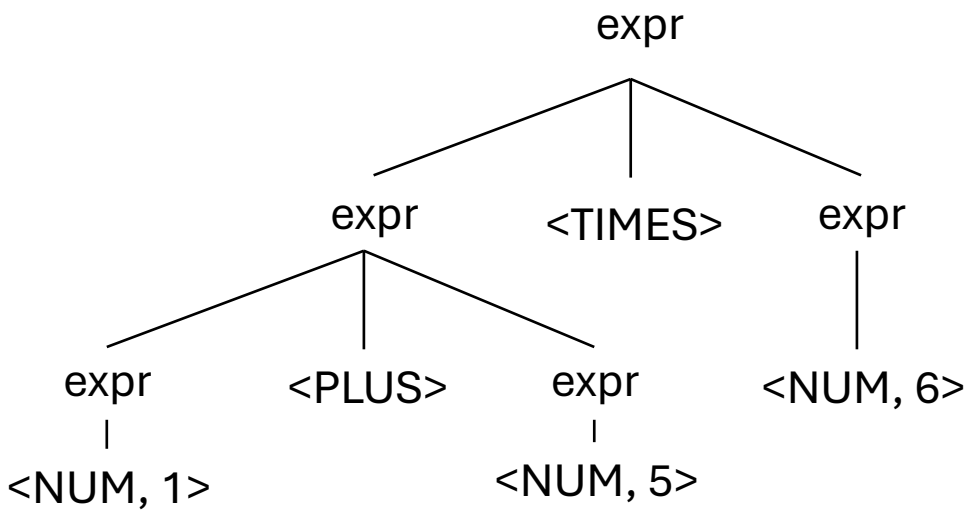
Quiz

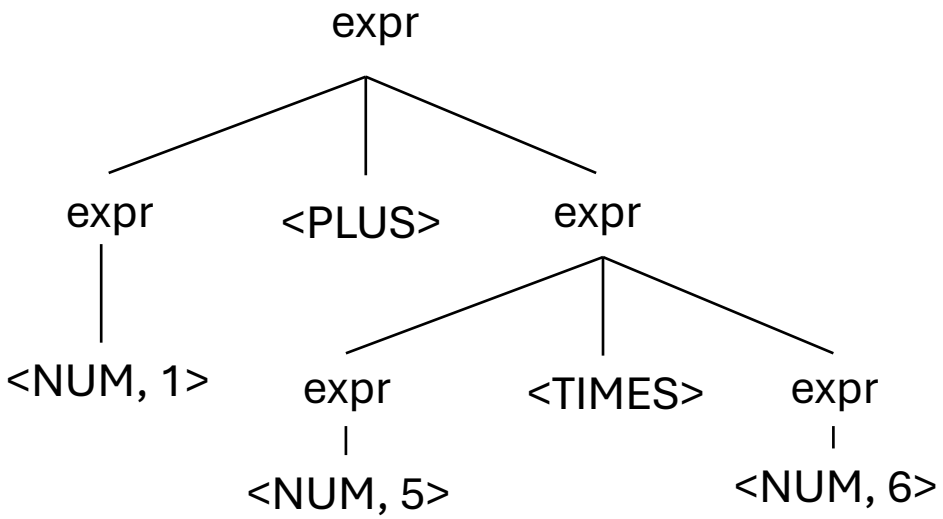
Write a few sentences about why it might be bad to have an ambiguous grammars

Ambiguous grammars

- `input: 1 + 5 * 6`

```
expr ::= NUM
       | expr PLUS expr
       | expr TIMES expr
       | LPAREN expr RPAREN
```





Ambiguous grammars

- `input: 1 + 5 * 6`

```
expr ::= NUM
       | expr PLUS expr
       | expr TIMES expr
       | LPAREN expr RPAREN
```

Evaluations are different!