Parser

Lecture Seven

Ambiguous Grammar

- Grammar G is unambiguous iff every w in L(G) has a unique leftmost (or rightmost) derivation
  - Fact: unique leftmost or unique rightmost implies the other
- A grammar lacking this property is ambiguous
  - Note: other grammars that generate the same language may be unambiguous
  - So, "ambiguous" applies to a grammar – not a language
- We need unambiguous grammars for parsing (well mostly: see later)

Example: Ambiguous Grammar

$$Exp \rightarrow Exp \; \texttt{Op} \; Exp \mid Dig$$

$$Op \rightarrow \texttt{+} \mid \texttt{-} \mid \texttt{*} \mid \texttt{/}$$

$$Dig \rightarrow \texttt{0} \mid \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9}$$

- Exercise: show that this is ambiguous
  - How? Show two different leftmost or rightmost derivations for the same string
  - Equivalently: show two different parse trees for the same string

2+3*4 – part 1

Exp → Exp + Exp | Exp – Exp | Exp * Exp
| Exp / Exp | Dig
Dig → [0-9]

Give a leftmost derivation of 2+3*4; show the parse tree

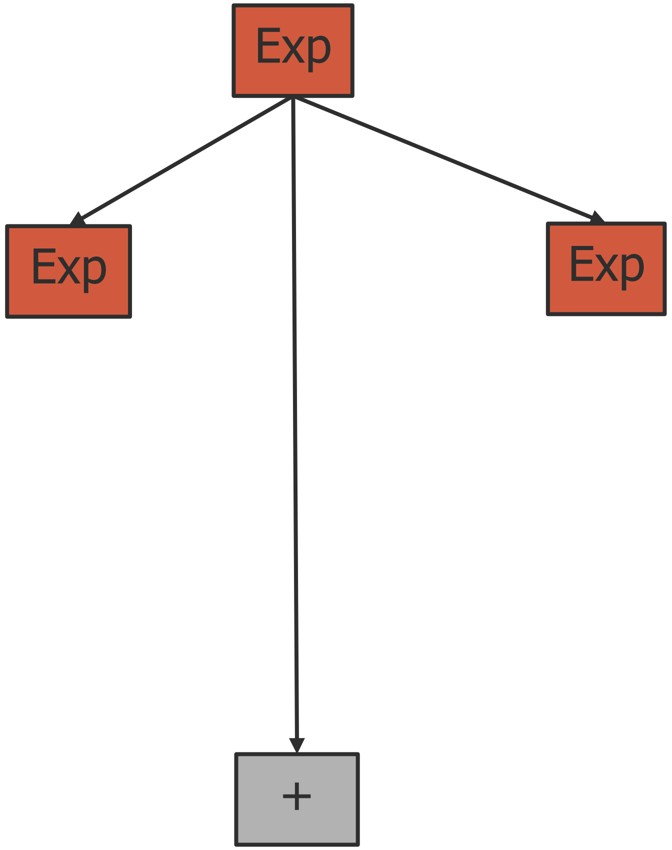
Exp

Exp

2+3*4 – part 2

Exp → Exp + Exp | Exp – Exp | Exp * Exp
| Exp / Exp | Dig
Dig → [0-9]

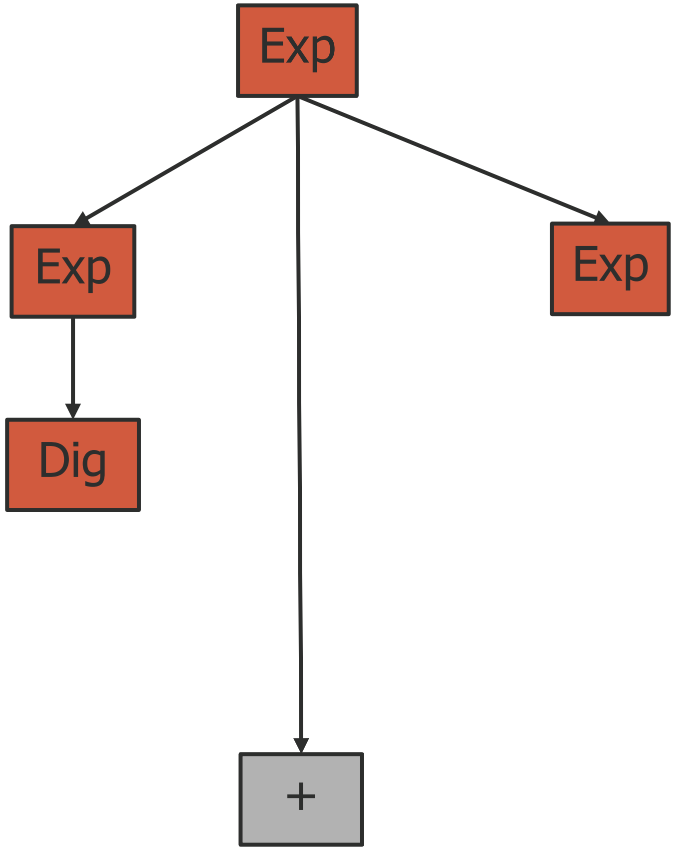
```
Exp
    => Exp + Exp
```



2+3*4

Exp → Exp + Exp | Exp – Exp | Exp * Exp
| Exp / Exp | Dig
Dig → [0-9]

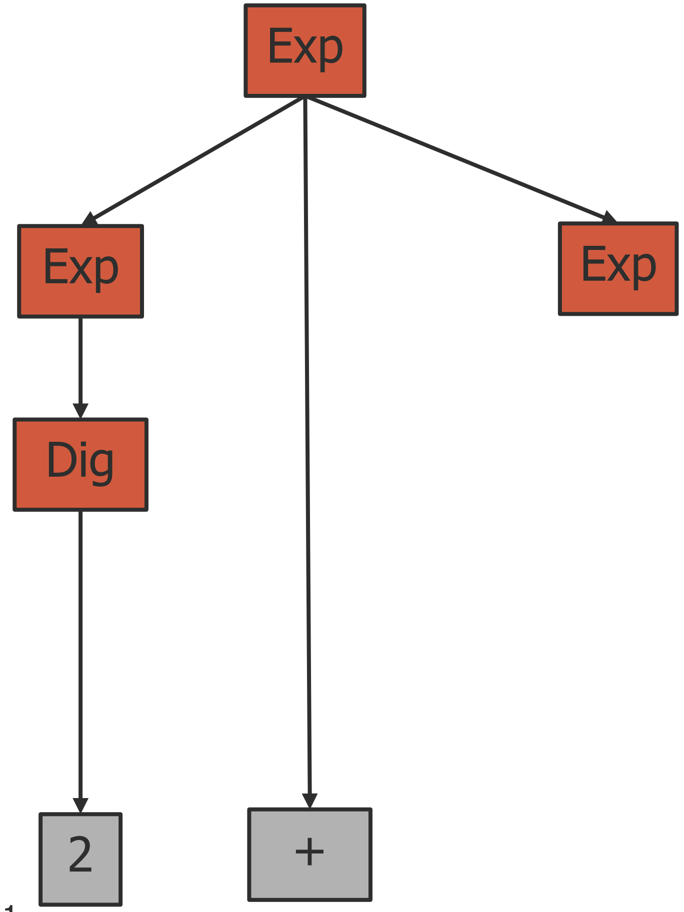
```
Exp
Exp => Exp + Exp
    => Dig + Exp
```



2+3*4 – part 4

Exp → Exp + Exp | Exp – Exp | Exp * Exp
    | Exp / Exp | Dig
Dig → [0-9]

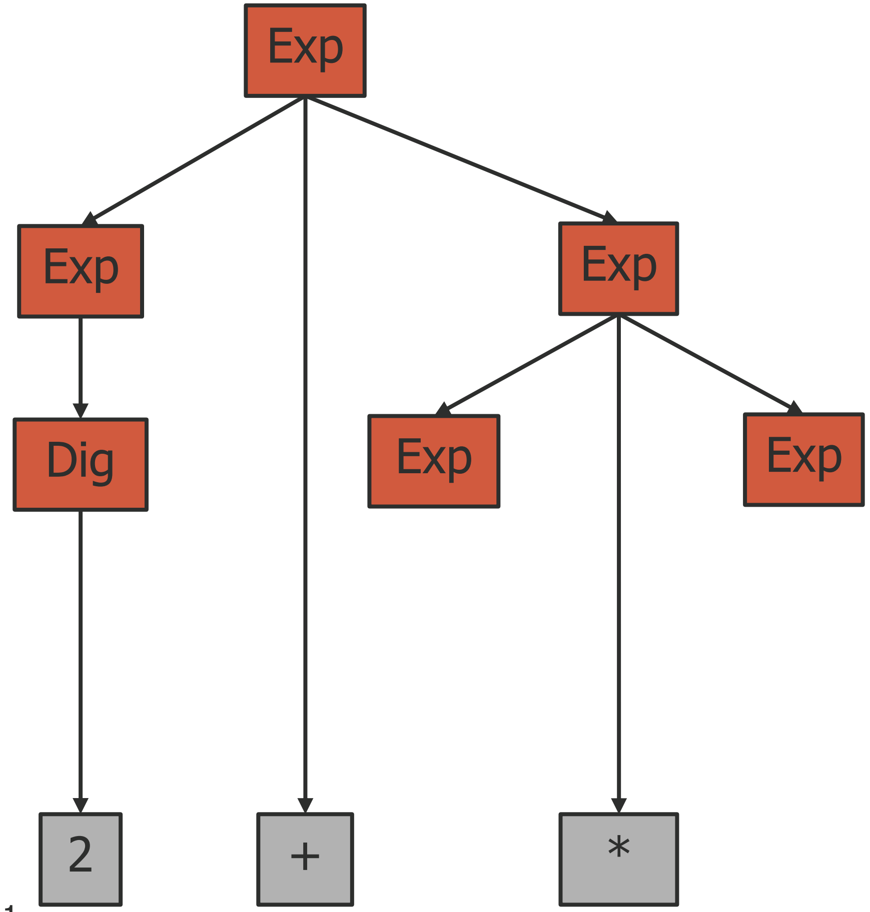
```
Exp
Exp => Exp + Exp
    => Dig + Exp
    =>  2  + Exp
```



2+3*4 – part 5

Exp ::= Exp + Exp | Exp – Exp | Exp * Exp
        | Exp / Exp | Dig
Dig ::= [0-9]

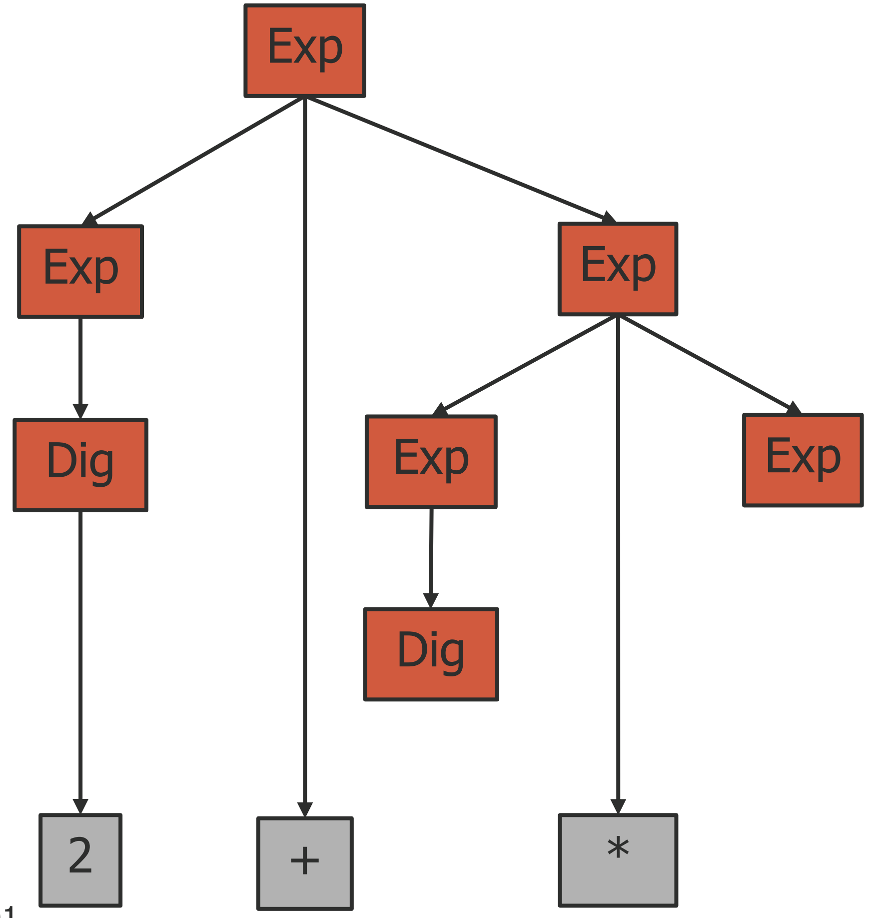
```
Exp
Exp => Exp + Exp
    => Dig + Exp
    =>  2  + Exp
    =>  2  + Exp * Exp
```



2+3*4 – part 6

Exp → Exp + Exp | Exp – Exp | Exp * Exp
        | Exp / Exp | Dig
Dig → [0-9]

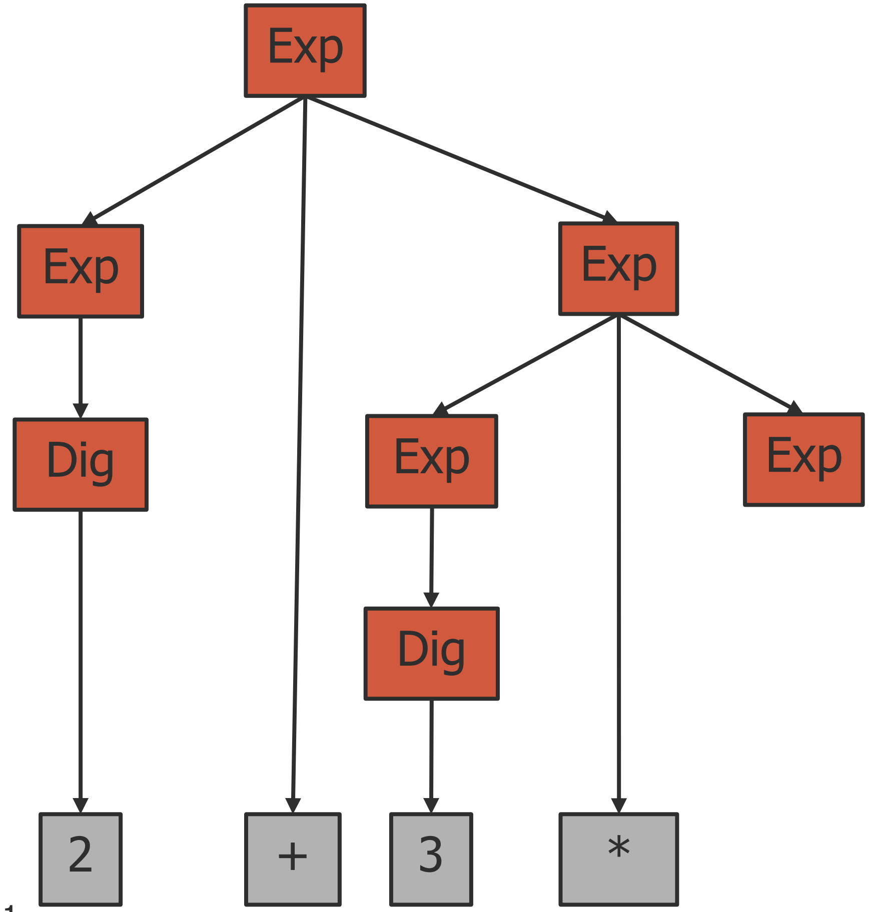
```
Exp
Exp => Exp + Exp
    => Dig + Exp
    =>  2  + Exp
    =>  2  + Exp * Exp
    =>  2  + Dig * Exp
```



2+3*4 – part 7

Exp → Exp + Exp | Exp – Exp | Exp * Exp
 | Exp / Exp | Dig
Dig → [0-9]

```
Exp
Exp => Exp + Exp
    => Dig + Exp
    =>  2  + Exp
    =>  2  + Exp * Exp
    =>  2  + Dig * Exp
    =>  2  +  3  * Exp
```



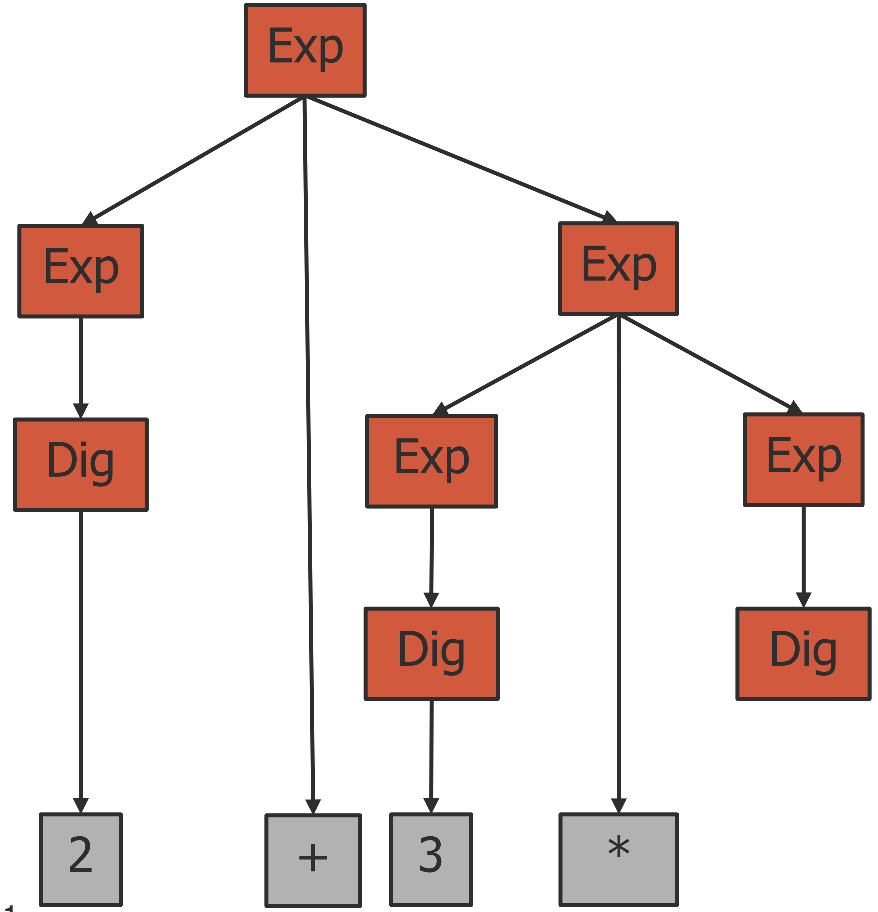
2+3*4 – part 8

Exp $\rightarrow$ Exp + Exp | Exp – Exp | Exp * Exp

| Exp / Exp | Dig

Dig $\rightarrow$ [0-9]

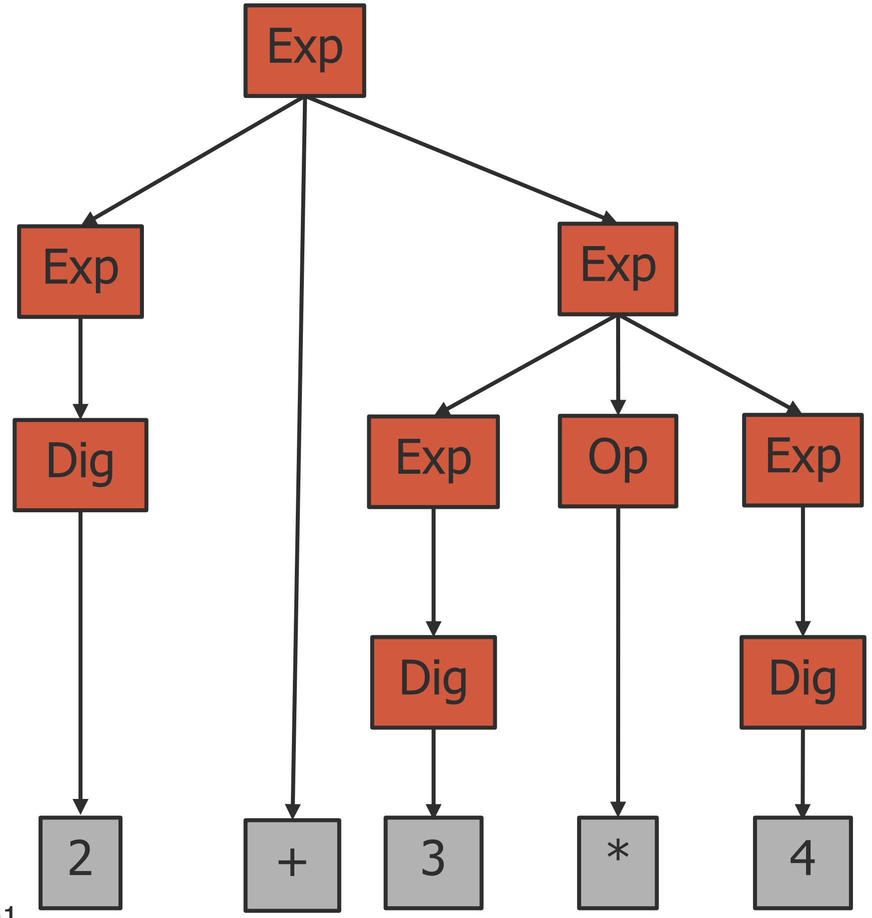
```
Exp
Exp => Exp + Exp
    => Dig + Exp
    =>  2  + Exp
    =>  2  + Exp * Exp
    =>  2  + Dig * Exp
    =>  2  +  3  * Exp
    =>  2  +  3  * Dig
```



2+3*4 – part 9

Exp → Exp + Exp | Exp – Exp | Exp * Exp
| Exp / Exp | Dig
Dig → [0-9]

```
Exp
Exp => Exp Op Exp
    => Dig Op Exp
    =>  2  Op Exp
    =>  2  +  Exp
    =>  2  +  Exp Op Exp
    =>  2  +  Dig Op Exp
    =>  2  +   3  Op Exp
    =>  2  +   3  *  Exp
    =>  2  +   3  *  Dig
    =>  2  +   3  *   4
```

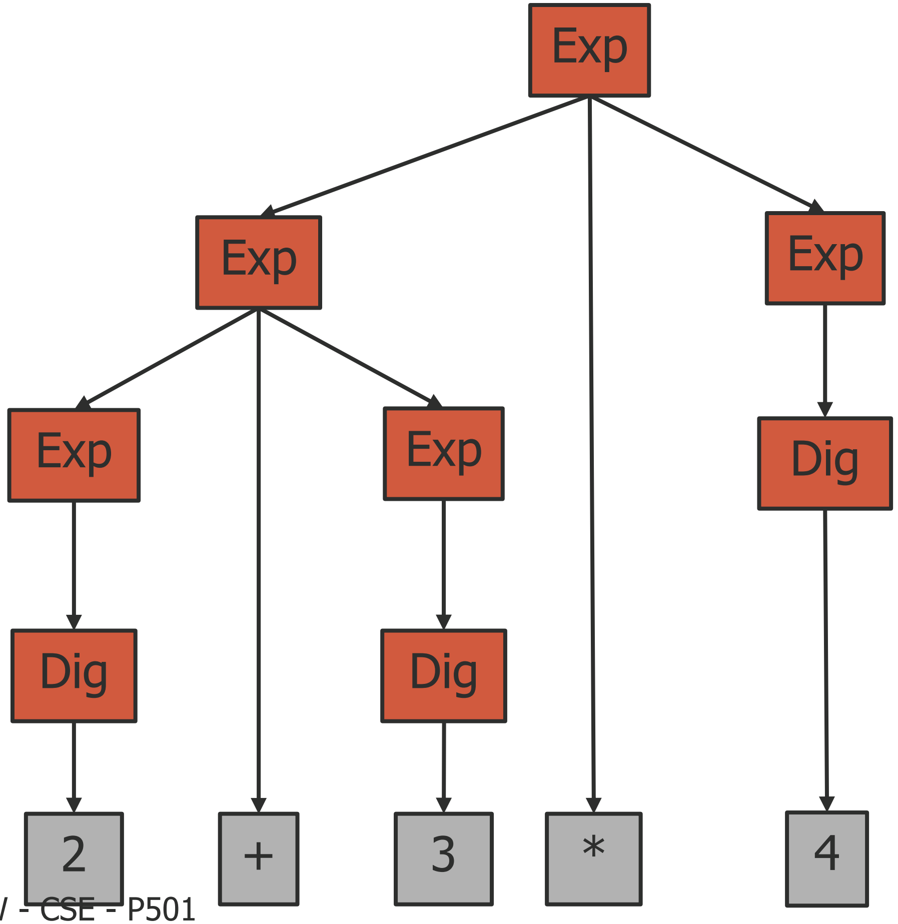


2+3*4 – part 10

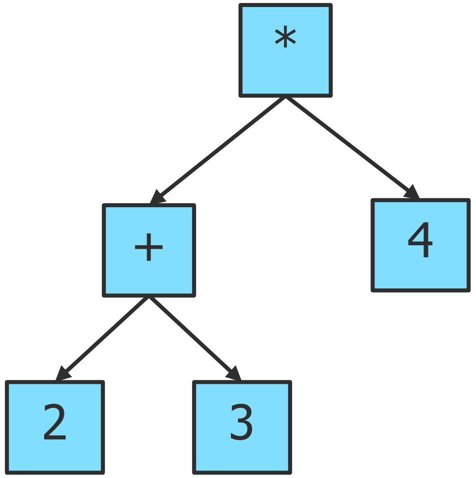
Exp → Exp + Exp | Exp – Exp | Exp * Exp
| Exp / Exp | Dig
Dig → [0-9]

Give a *different* leftmost derivation of 2+3*4

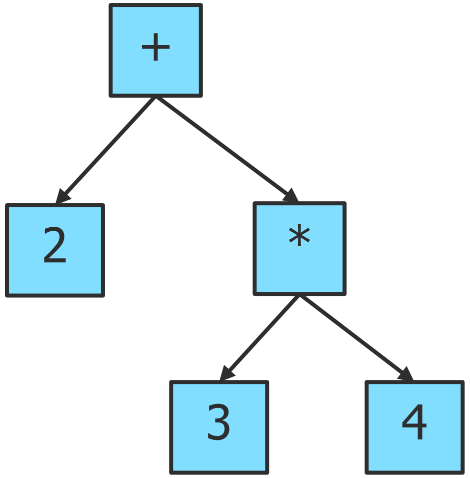
```
Exp
     => Exp * Exp
     => Exp + Exp * Exp
     =>  2  + Exp * Exp
     =>  2  +  3  * Exp
     =>  2  +  3  *  4
```



Are derivations *equivalent*?



Result = (2 + 3) * 4 = 20



Result = 2 + (3 * 4) = 14

Another example

Exp → Exp + Exp | Exp – Exp | Exp * Exp
| Exp / Exp | Dig
Dig → [0-9]

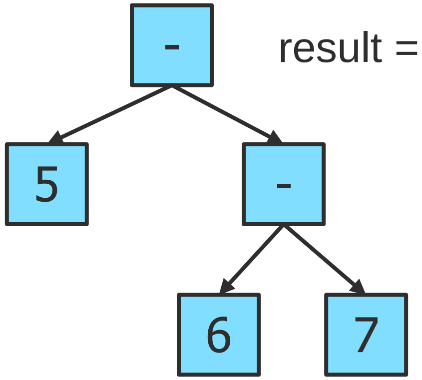
- Give two different derivations of 5 – 6 – 7

Another example

Exp → Exp + Exp | Exp – Exp | Exp * Exp
| Exp / Exp | Dig
Dig → [0-9]

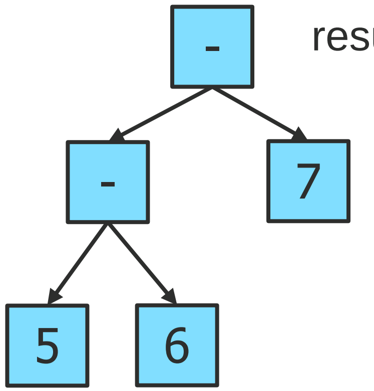
Give two different *rightmost* derivations of 5 – 6 – 7

```
Exp => Exp - Exp
    => Exp - Exp - Exp
    => Exp - Exp -  7
    => Exp -  6  -  7
    =>  5  -  6  -  7
```



result = 6

```
Exp => Exp - Exp
    => Exp -  7
    => Exp - Exp -  7
    => Exp -  6  -  7
    =>  5  -  6  -  7
```



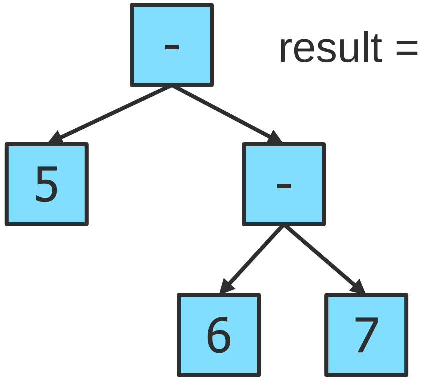
result = -8

Another example

Exp → Exp + Exp | Exp – Exp | Exp * Exp
| Exp / Exp | Dig
Dig → [0-9]

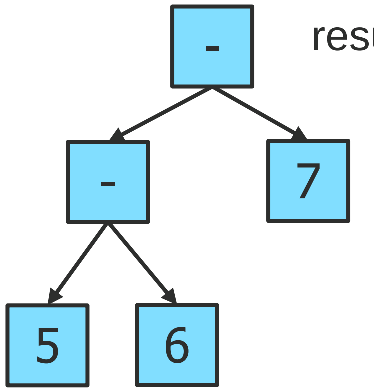
Give two different *leftmost* derivations of 5 – 6 – 7

```
Exp => Exp - Exp
    =>  5  - Exp
    =>  5  - Exp - Exp
    =>  5  -  6  - Exp
    =>  5  -  6  -  7
```



result = 6

```
Exp => Exp - Exp
    => Exp - Exp - Exp
    =>  5  - Exp - Exp
    =>  5  -  6  - Exp
    =>  5  -  6  -  7
```



result = -8

What went wrong?

- Grammar did not capture precedence or associativity
  - Eg: 2 + (3 * 4) = 14 versus (2 + 3) * 4 = 20
  - Eg: 5 - (6 - 7) = 6 versus (5 - 6) - 7 = -8
- Solution
  - Create a non-terminal for each level of precedence
  - Isolate the corresponding part of the grammar
  - Force the parser to recognize higher precedence sub-expressions first

Classic Expression Grammar

exp $\rightarrow$ exp + term | exp – term | term

term $\rightarrow$ term * factor | term / factor | factor

factor $\rightarrow$ int | (exp)

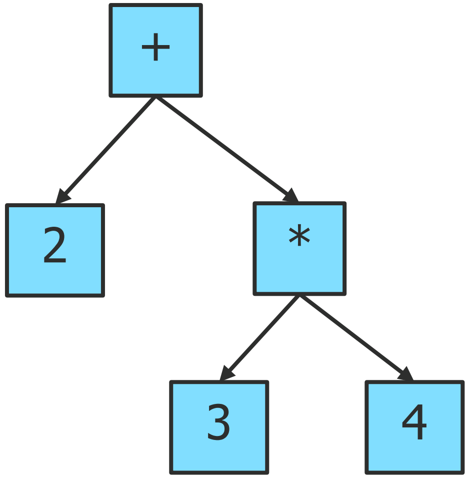
int $\rightarrow$ 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7

```
E → E + T | E - T | T
T → T * F | T / F | F
F → ( E ) | D
D → [0-9]
```

Derive 2 + 3 * 4

```
E → E + T | E - T | T
T → T * F | T / F | F
F → ( E ) | D
D → [0-9]
```

```
E => E + T
  => E + T * F
  => E + T * D
  => E + T * 4
  => E + F * 4
  => E + D * 4
  => E + 3 * 4
  => T + 3 * 4
  => F + 3 * 4
  => D + 3 * 4
  => 2 + 3 * 4
```



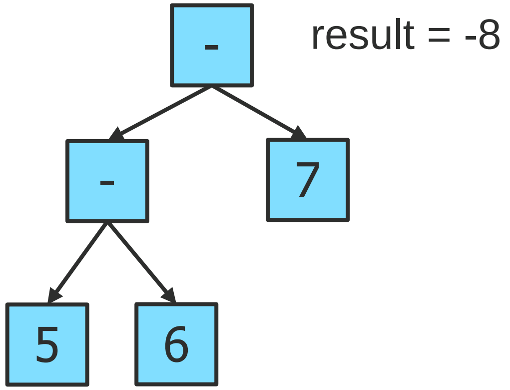
Result = 2 + (3 * 4) = 14

This grammar yields the correct, expected (school algebra) result

Derive 5 - 6 - 7

```
E → E + T | E – T | T
T → T * F | T / F | F
F → ( E ) | D
D → [0-9]
```

```
E => E - T
  => E - F
  => E - D
  => E - 7
  => E - T - 7
  => E - F - 7
  => E - D - 7
  => E - 6 - 7
  => F - 6 - 7
  => D - 6 - 7
  => 5 - 6 - 7
```



- This grammar yields the correct, expected (school algebra) result
- Note how left-recursive rules yield left-associativity

Classic Example of Ambiguous Grammar

- Grammar for conditional statements

$$
\begin{aligned}
\text{stm} \rightarrow\ & \text{if (cond) stm} \\
\mid\ & \text{if (cond) stm else stm}
\end{aligned}
$$

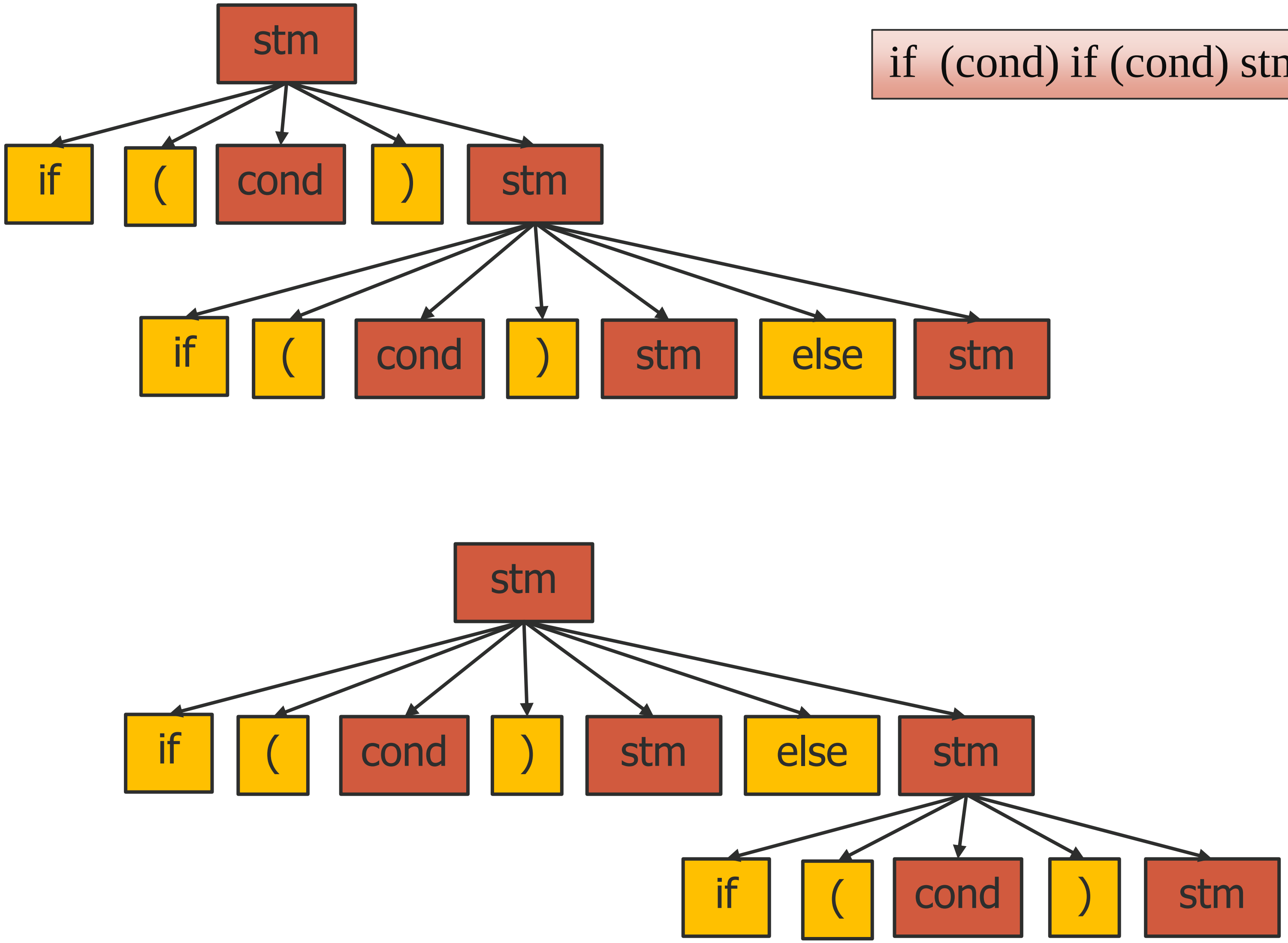
- Exercise: show that this is ambiguous
  - How?

"The Dangling Else" - a 'weakness' in C, Pascal, etc

Two Derivations

stm → if (cond) stm
| if (cond) stm else stm

if (cond) if (cond) stm else stm



Solving the Dangling Else

- Fix the grammar to separate *if* statements with *else* clause from those without
  - Done in Java reference grammar
  - Adds lots of non-terminals
- Use some ad-hoc rule in parser
  - “else matches closest unpaired if”
- Change the language
  - Only possible if you 'own' the language

Resolving Ambiguity with Grammar (1)

```
Stm      → IfElse | IfNoElse
IfElse   → if ( Exp ) IfElse else IfElse
IfNoElse → if ( Exp ) Stm
          | if ( Exp ) IfElse else IfNoElse
```

- formal, no additional rules beyond syntax
- sometimes obscures original grammar

Resolving Ambiguity with Grammar (2)

- If you can (re-)design the language, avoid the problem entirely

IfStm $\rightarrow$ if Exp then Stm end

| if Exp then Stm else Stm end

  - formal, clear, elegant
  - allows sequence of *Stms* in *then* and *else* branches, no { } needed
  - extra *end* required for every *if*

Parser Tools and Operators

- Most parser tools cope with ambiguous grammars
  - Earlier productions chosen before later ones
  - Longest match used if there is a choice
  - Makes life simpler if used with discipline
  - But be sure the tool does what you really want

- Specify operator precedence & associativity
  - Allows simpler, ambiguous grammar with fewer non-terminals
  - Used in CUP

Derivation

- Given a production:

  A ::= X1 X2 ... Xn

  aAb => aX1 X2 ... Xnb   is called a derivation

- A derivation is a sequence of replacements of structure names by choices on the right-hand sides of grammar rules.
  - It starts with a single structure name and ends with a string of token symbols
  - At each step in a derivation, a single replacement is made using one choice from a grammar rule.

- The set of all strings of token symbols obtained by derivations from the `exp` symbol is the *language defined by the grammar* of expressions. We can write this as:

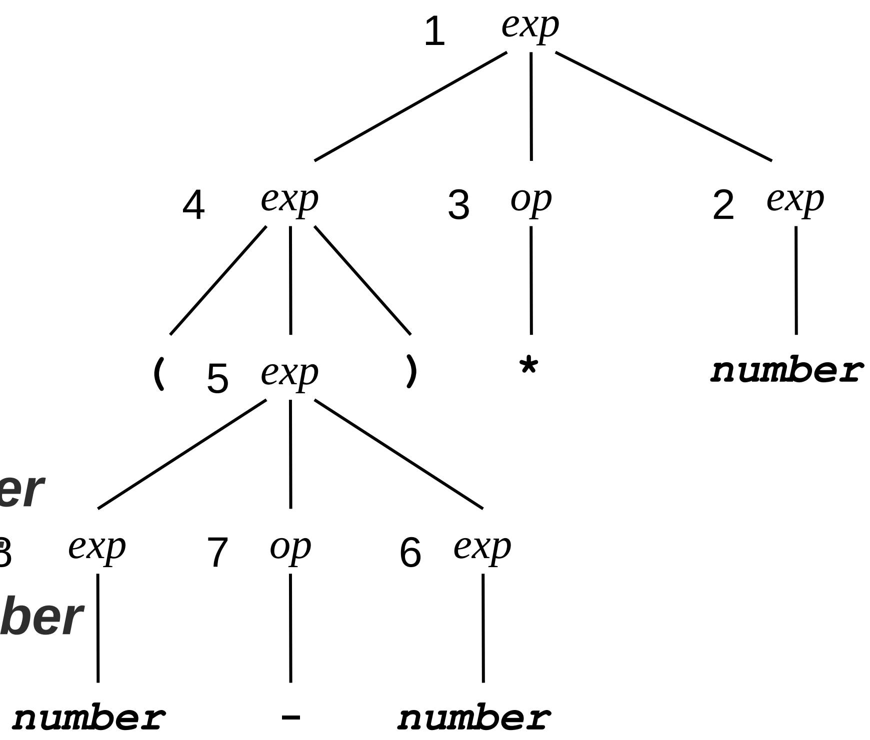
$$L(G) = \{ s: exp \Rightarrow^* s\}$$

Derivation (Example)

exp → *exp op exp* | **(** *exp* **)** | ***number***
op → **+** | **-** | *****

Derive (34-3)*42

(1) *exp* ⇒ *exp op exp*
(2) ⇒ *exp op* ***number***
(3) ⇒ *exp* **** number***
(4) ⇒ ***(*** *exp* ***) * number***
(5) ⇒ ***(*** *exp op exp* ***) * number***
(6) ⇒ ***(****exp op* ***number) * number***
(7) ⇒ ***(****exp* ***- number) * number***
(8) ⇒ ***(number - number)*number***



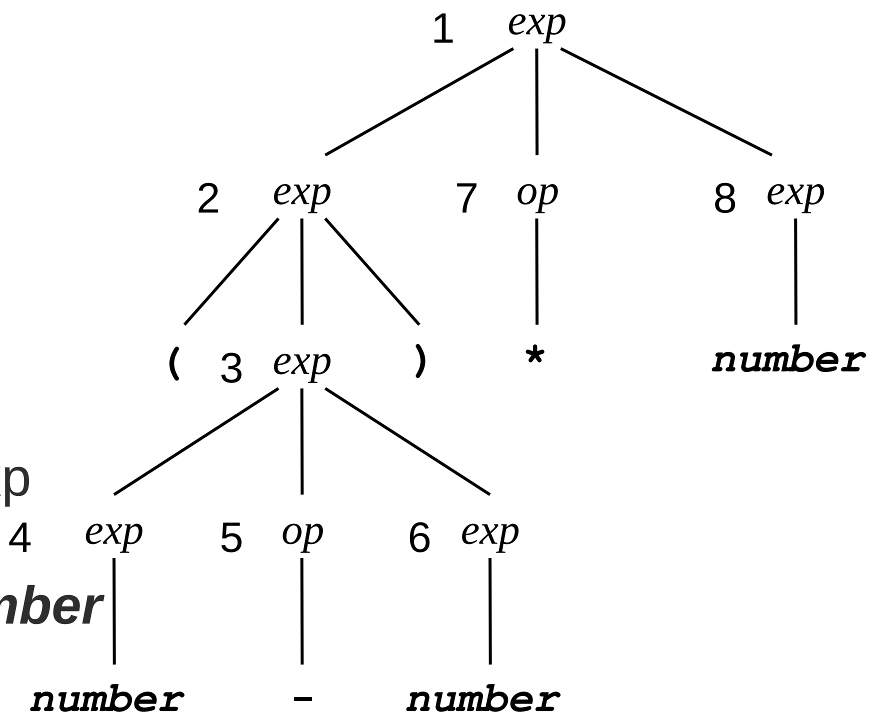
Right most derivation expand the Rightmost nonterminal in the production

Derivation (Example)

exp $\rightarrow$ *exp op exp* | **(** *exp* **)** | ***number***
op $\rightarrow$ **+** | **-** | *****

Derive `(34-3)*42`

(1) exp $\Rightarrow$ exp op exp
(2) $\Rightarrow$ ***(***exp***)*** op exp
(3) $\Rightarrow$ ***(***exp op exp***)*** op exp
(4) $\Rightarrow$ ***(number*** op exp***)*** op exp
(5) $\Rightarrow$ ***(number*** - exp***)*** op exp
(6) $\Rightarrow$ ***(number - number)*** op exp
(7) $\Rightarrow$ ***(number - number) **** exp
(8) $\Rightarrow$ ***(number - number) * number***

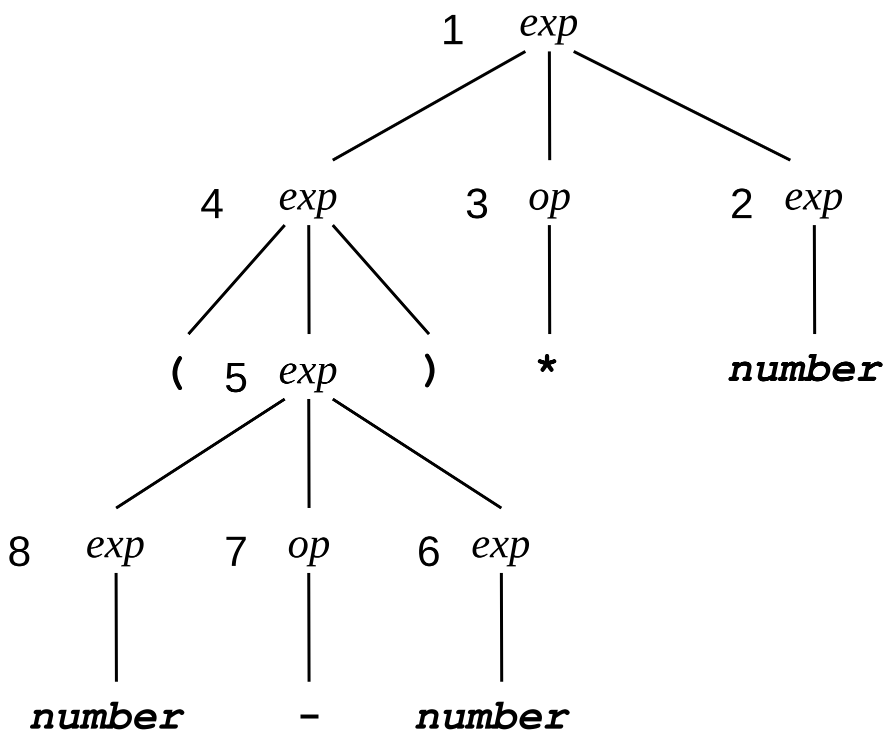


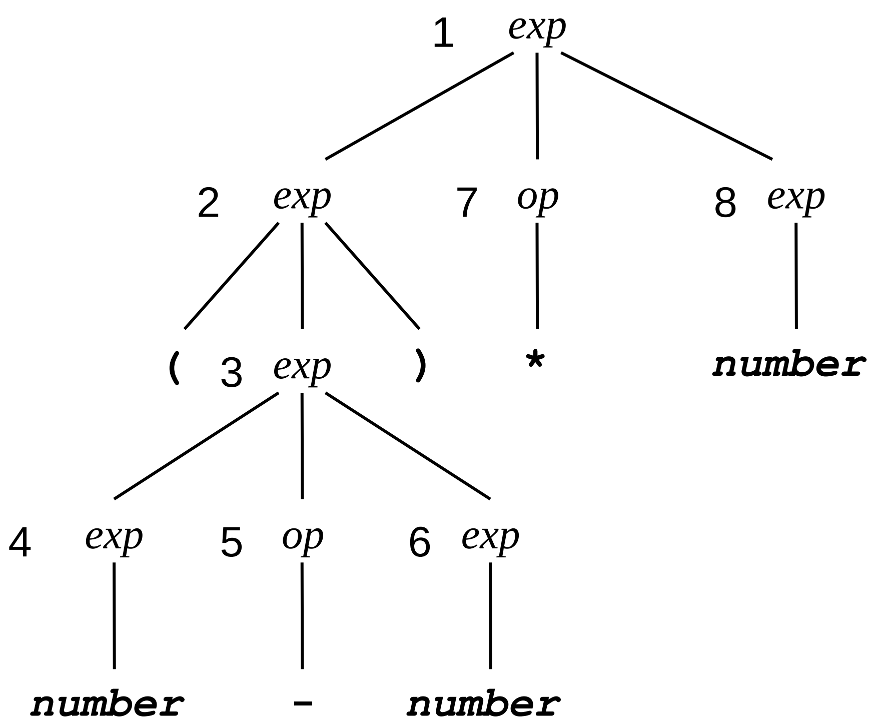
Left most derivation expand the leftmost nonterminal in the production

Parse Tree

Parse tree:

- the start symbol →the root
- the terminals of the input sequence → leafs
- for each production A →X1 X2 ... Xn used in a derivation, a node A with children X1 X2 ... Xn





Derivations

- Derivations allow us to replace any of the variables in a string. Leads to many different derivations of the same string.
- Leftmost Derivations: expand the leftmost nonterminal in the production
- Rightmost Derivations: expand the rightmost nonterminal in the production

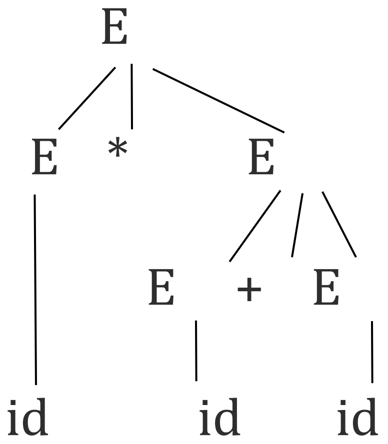
Ambiguous Grammars

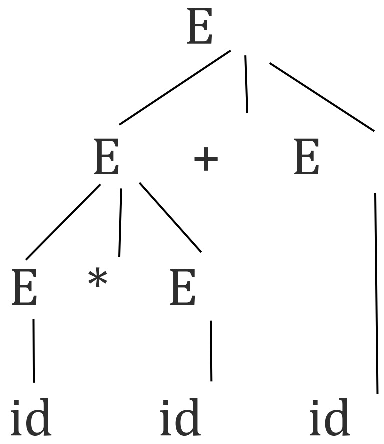
E ::= E + E

| E * E

| id

String id*id+id





It is ***ambiguous***: more than one parse tree for the same input depend on derivation used.

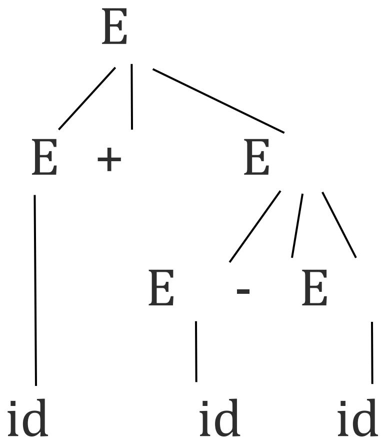
Ambiguous Grammars

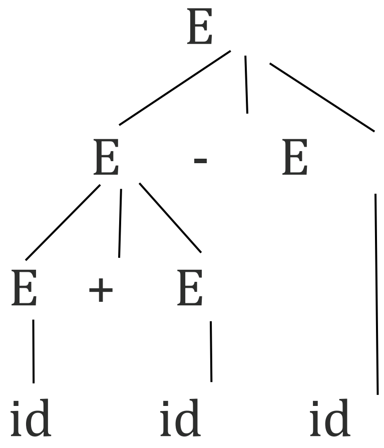
E ::= E + E

| E - E

| id

String id*id+id





It is ***ambiguous***: more than one parse tree for the same input depend on derivation used.

Ambiguous Grammars

Remove Ambiguity:

1. Make the grammar left-recursive

   $$\mathbf{E \rightarrow E + E' \mid E'}$$

2. Make the grammar right-recursive

   $$\mathbf{E \rightarrow E' + E \mid E'}$$

3. Make the grammar non-recursive

   $$E ::= E' + E' \mid E'$$

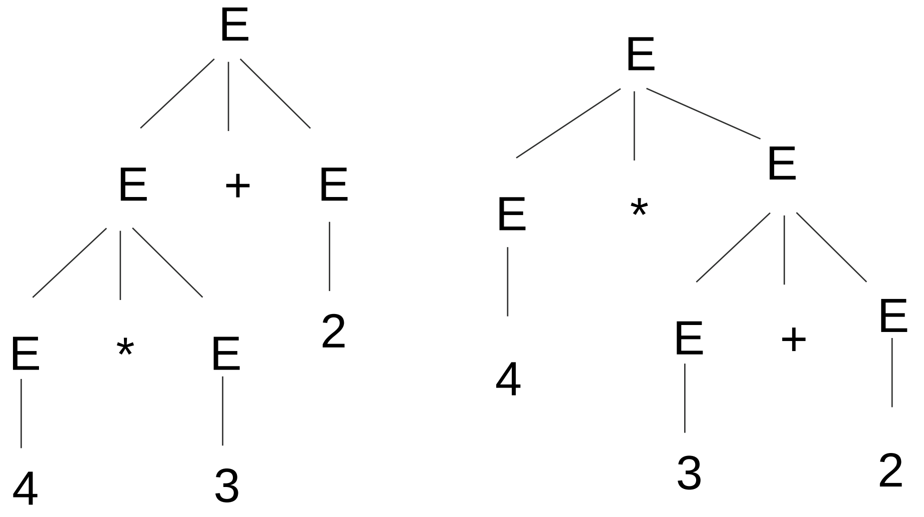
Parsing

- A *leftmost* derivation corresponds to a (*top-down*) pre-order traversal of the parse tree.
- A *rightmost* derivation corresponds to a (*bottom-up*) post-order traversal, but in reverse.
- Top-down parsers construct leftmost derivations.
  - (LL = <u>L</u>eft-to-right traversal of input, constructing a <u>L</u>eftmost derivation)
- Bottom-up parsers construct rightmost derivations in reverse order.
  - (LR = <u>L</u>eft-to-right traversal of input, constructing a <u>R</u>ightmost derivation)

Ambiguous Grammar(priority)

E→E+E|E*E| E^E| num

String:4*3+2



<u>Remove Ambiguity</u>

E→E+T|T

T→T*F|F

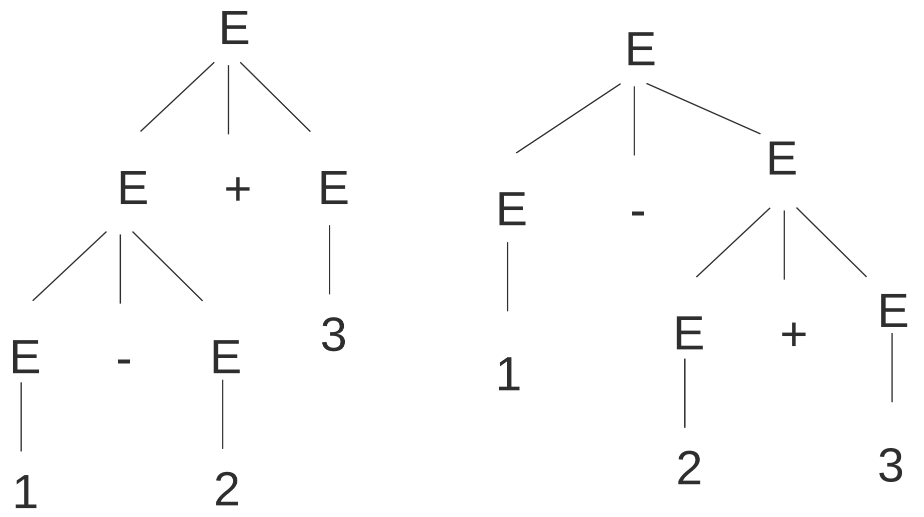
F→G^F|G

G→num

Ambiguous Grammar (associativity)

E→E+E|E-E|num

String:1-2+3



Remove Ambiguity

E→E+T|E-T|T

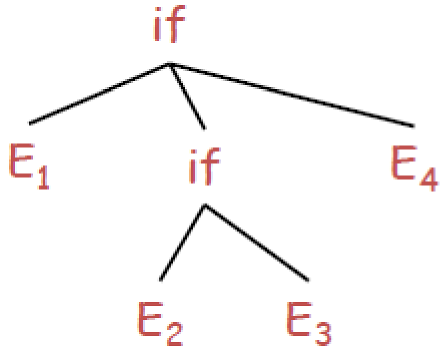
T→num

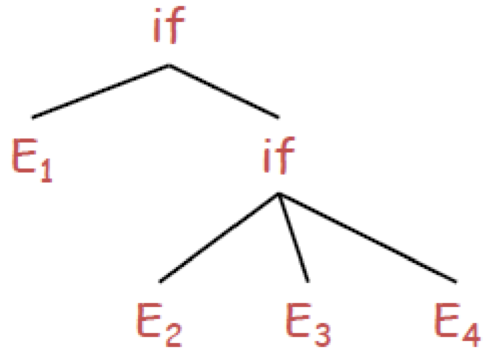
Ambiguous Grammars

E → if E then E

| if E then E else E

String : if E1 then if E2 then E3 else E4





Unambiguous Grammar

E → MIF | UIF

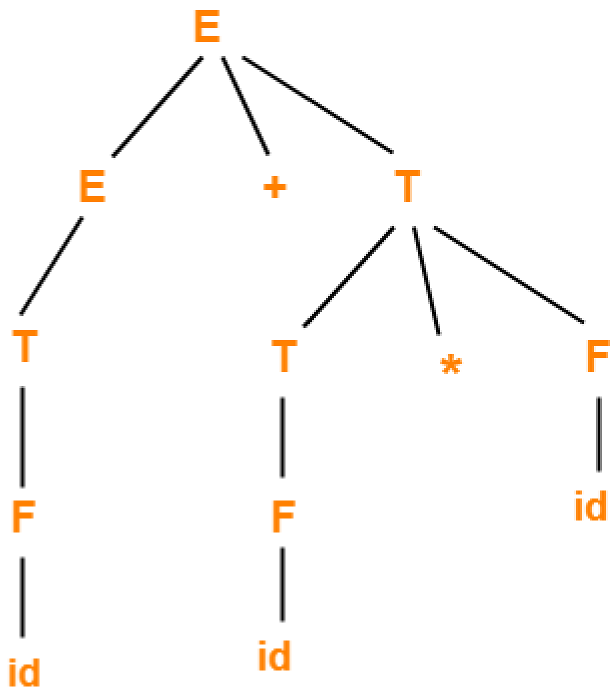
MIF → if E then MIF else MIF

UIF → if E then E

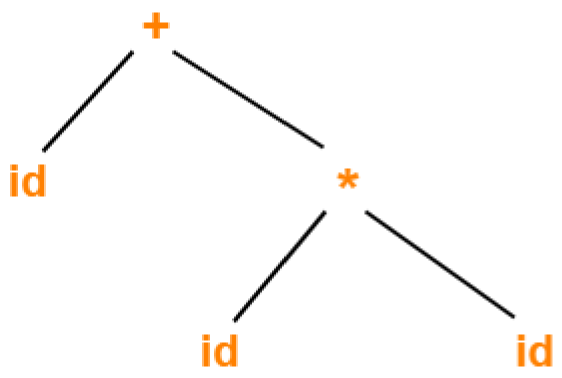
| if E then MIF else UIF

Syntax tree

- Syntax tree is the compact form of a parse tree.

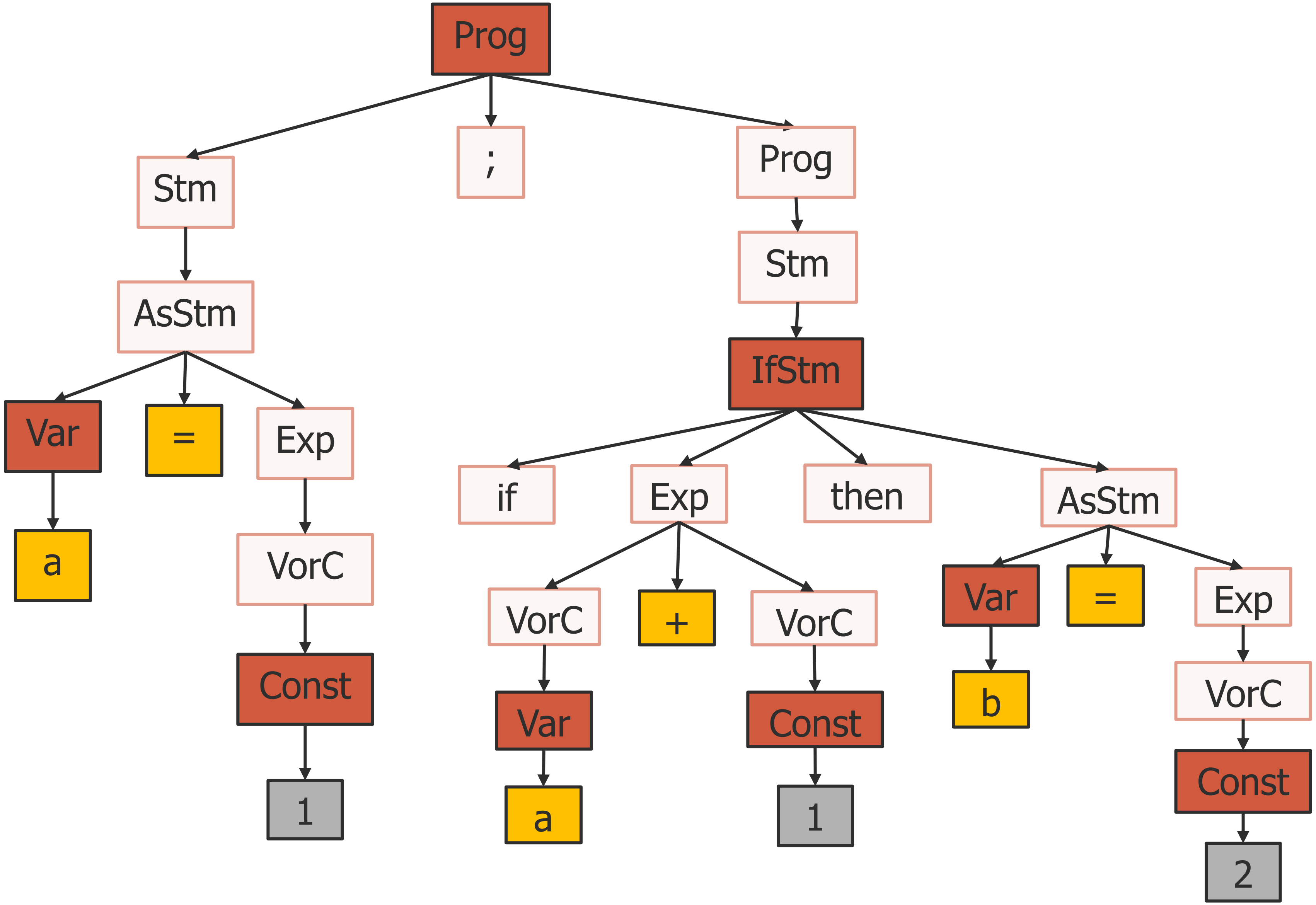


Parse Tree



Syntax Tree

Junk Nodes in the Parse Tree



AST (Abstract Syntax Tree)