02/24/2025

ch3. parsers

1. specify syntax. CFG.

$G = (S, N, T, P)$

S: start symbol.

N: a set of non-terminal symbols.

T: terminals / a finite set of word / tokens — leaf of parse tre

* P: a set of productions / rewrite rules

a rule of a CFG: $N \xrightarrow{::=} (N \cup T)^{+}$

ex1: Grammer production rule: G1

P:
① $S \rightarrow [\ ]$
② $S \rightarrow (\ )$
③ $S \rightarrow [S]$
④ $S \rightarrow (S)$

$T = \{[,], (,)\}$.

$N = \{S\}$.

Start: S. the nonterm on the LHS of the first production rule

2. derivation

Given a grammar $G = \{ S, N, T, P)$.

start from the string S, rewrite it. by applying a production, repeating it ~~by~~ until we get a string of terminals only.

$S \Rightarrow (S)$ // ④ sentential form

$\Rightarrow ([S])$ // ③ $(N \cup T)^{+}$

$\Rightarrow ([[ ]])$ // ① sentence

. sentences derived by G | ?

$S \overset{?}{\underset{*}{\Rightarrow}} [ ] [ ]$

$([ ]) \quad [( )]$

$[[ ]] \quad (( ))$

How many: infinite

$L(G) = \{$ all sentences can ~~be~~ ~~derived~~ by $G \}$.

eX2: $G_2 = \{S_2, N_2, T_2, P_2\}$

0 $E \rightarrow E \ op \ E$
1 $E \rightarrow num$
2 $E \rightarrow id$
3 $op \rightarrow +$
4 $op \rightarrow -$

$E \rightarrow E \ op \ E$
$\quad | \ num$
$\quad | \ id$

$op \rightarrow + \ | \ -$

G2: $S_2 = \{E\}$
$N_2 = \{E, op\}$
$T_2 = \{num, id, +, -\}$

id − num + id
x − 2 + y

<id, x>
<op + >
<num, 2>

$\underline{E} \Rightarrow \underline{E} \ op \ E$	0
$\Rightarrow \underline{E} \ op \ E \ op \ E$	0
$\Rightarrow id \ \underline{op} \ E \ op \ E$	2
$\Rightarrow id - \underline{E} \ op \ E$	3
$\Rightarrow id - num \ \underline{op} \ E$	1
$\Rightarrow id - num + E$	3
$\Rightarrow id - num + id$	2

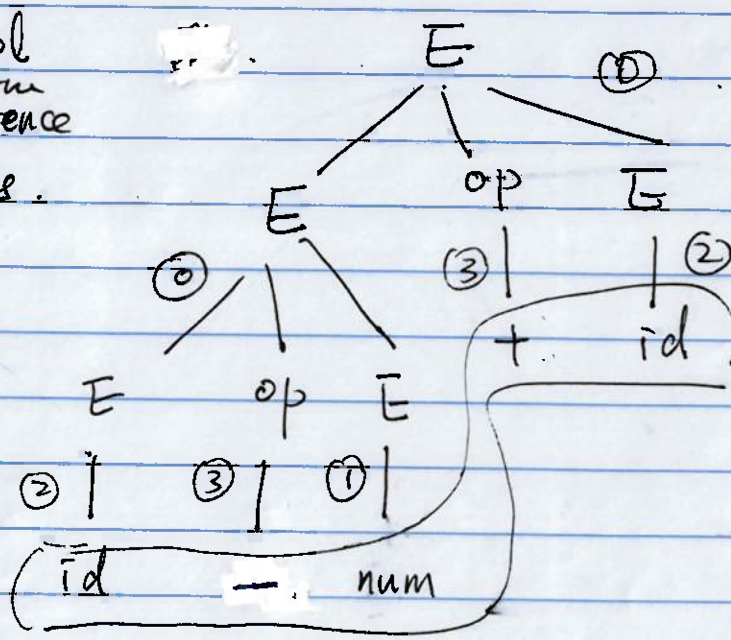
left most derivation: always rewrite the left most N. symobol.

right most derivation: always pick righmost N.

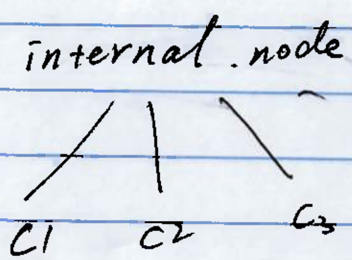
arbitrary derivation.

derivation / parse tree:

Start symbol as root, T in the sentence are the leaves.



application production rule



* same tree.

* different derivation. LMD 0 → 0 → 2 → 3 → 1 → 3 → 2.

0 → 3 → 2 → 0 → 2 → 1 → 3

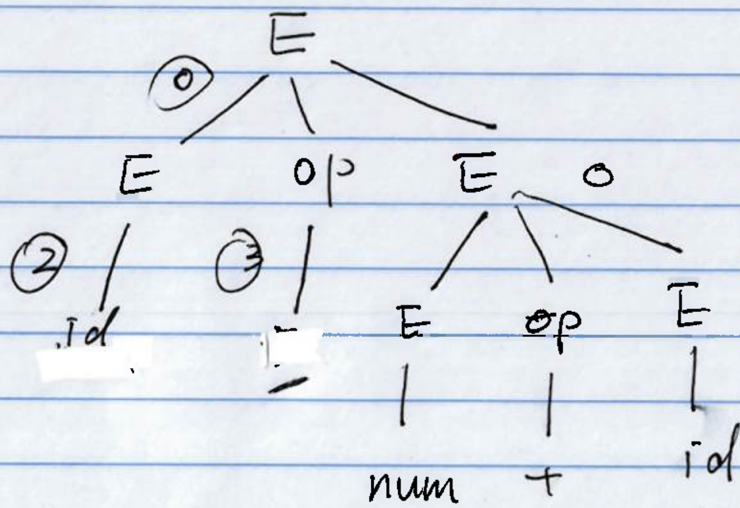
RMD 0 → 2 → 3 → 0 → 1 → 3 → 2

meaning. x − 2 + y

(52)

x - 2 + y. LMD 0 → 2 → 3 → 0 → 1 → 3 → 2.

same. sentence. different trees.



$\left\{ \begin{array}{l} x - \underline{(2+y)} \\ \underline{\underline{x-2}} + y \end{array} \right.$

$\left\{ \begin{array}{l} \underline{\underline{x-2}} + y \\ x - \underline{\underline{2+y}} \end{array} \right.$

3. ambiguous grammar.

G is an ambiguous grammar if there exists a string w that has at least 2 different LMD parse trees

G2 is ambiguous. x + $\underline{2+y}$. $\underline{\underline{x+2}}$ + y

$\underline{\underline{x+2}}$

ambiguous grammars are problematic. should be avoided.

How? try to rewrite the grammar.

EX: rewrite G2. by forcing left assocaity for "+", "−"

$\underline{x - 2 + y} + z$.

$E \rightarrow E \text{ op } \underline{E}$.
$\quad | \text{ num}$
$\quad | \text{ id}$

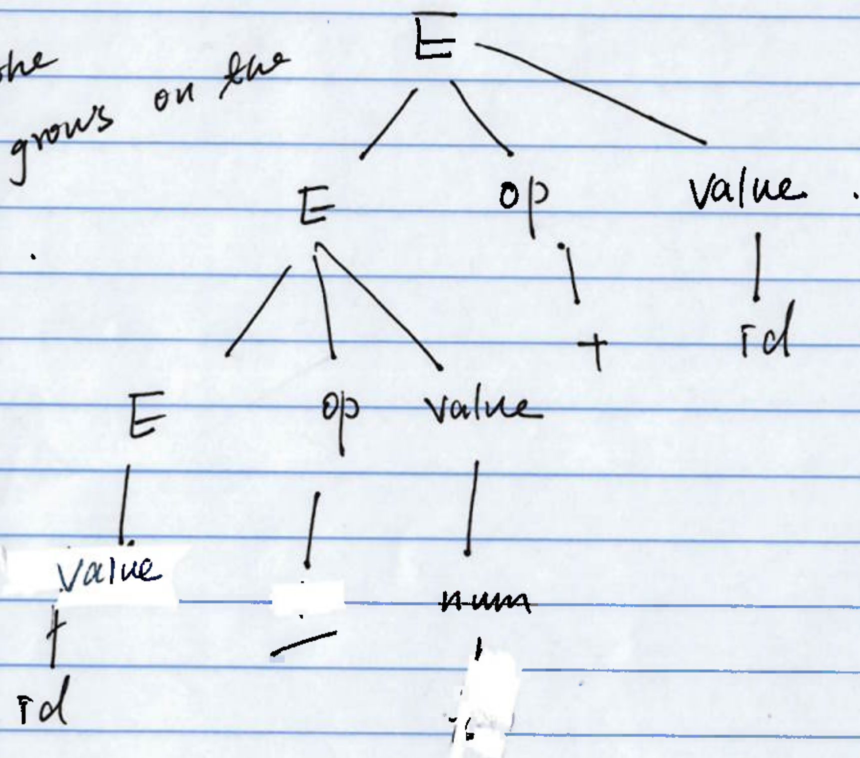
$\text{op} \rightarrow +$
$\quad | -$

$E \rightarrow E \text{ op Value}$
$\quad | \text{ value}$.

$\text{Val} \rightarrow \text{num}$
$\quad | \text{ id}$

$\text{op} \rightarrow +$
$\quad | -$

force the tree grows on the left.



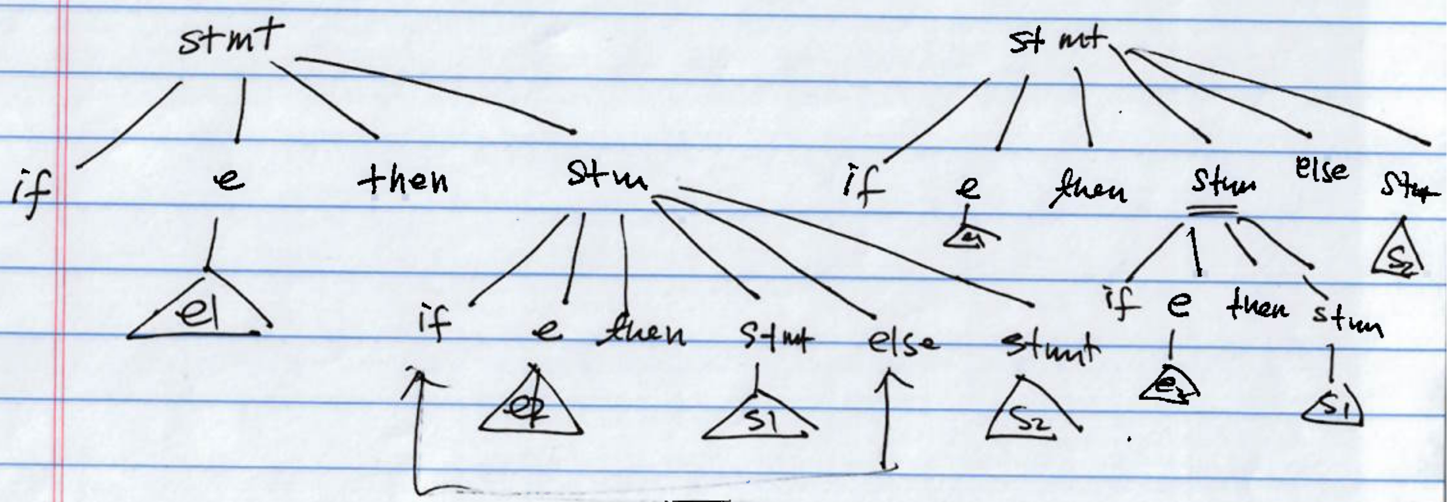
02/26/2025

ex: ① stmt → if e then stmt

| if e then stmt else stmt

| other-stmt ...

if E1 then if E2 then S1 else S2



ambiguous

rewrite: make sure else is matched

with the innermost if.

closest

then so

if e1 then if e2 else st1 else st2

①　　　　　　②　　　③　　　　④

if e1 then if e2 then st else st2

① stmt → if e then stmt

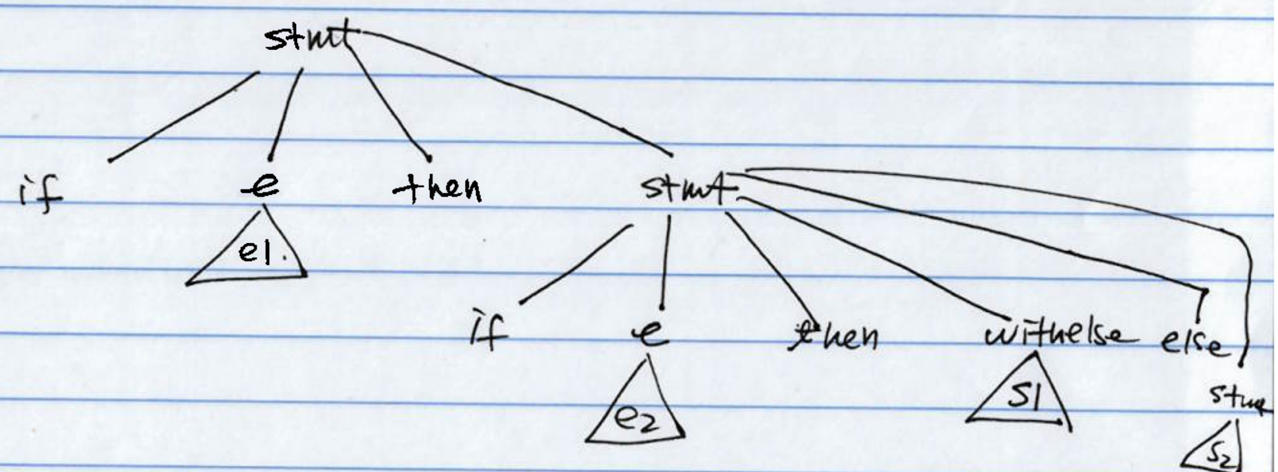
② | if e then withelse else stmt

③ | otherstmt.

④ withelse → if e then withelse else withelse

otherstmt.

if e1 then ~~e2~~ if e2 then s1 else s2



removing ambiguity is hard.

* actually it is hard to know if a grammar is ambiguous. (no efficient alg. to tell if an arbitrary grammar)

*. some CFL are inherently ambiguous.

i.e. no unambiguous grammar.

ex: $E \rightarrow E \text{ op } E$
$\quad | \text{ num}$
$\quad | \text{ id}$

$\text{op} \rightarrow +$
$\quad | -$
$\quad | *$
$\quad | /$

$E \rightarrow E \text{ op Value}$
$\quad | \text{ value}$
$\text{Value} \rightarrow \text{num}$
$\quad | \text{ id}$

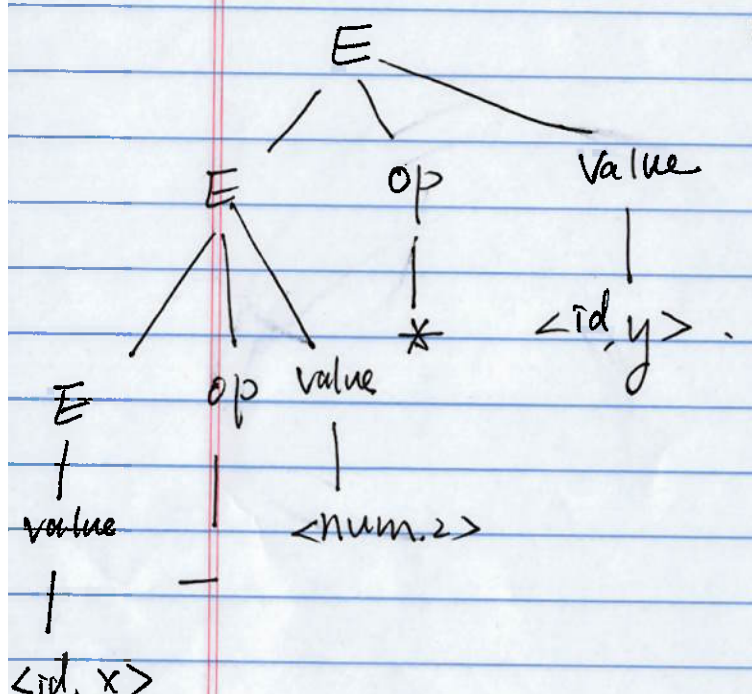
$\text{op} \rightarrow +$
$\quad | -$
$\quad | *$
$\quad | /$

$\underline{\underline{x - 2}} + y$ unique parse tree

$\underline{\underline{x - 2}} * y$

$x - (2 * y)$

$(x - 2) * y$



E

rewrite the grammar
force */ have higer
precedence than +/-

4. force. ~~per~~ precedence of different operators.

rewrite the grammar by introducing new non-terminals for each level of precedence. Structure the grammar s.t. the higher precedence op must go through the lower precedence op.

level 1 ( )

2 * /

3 + −

X − 2 * y

0 Goal → E

1 E → E + T

2 | E − T

3 | T

4 T → T × F

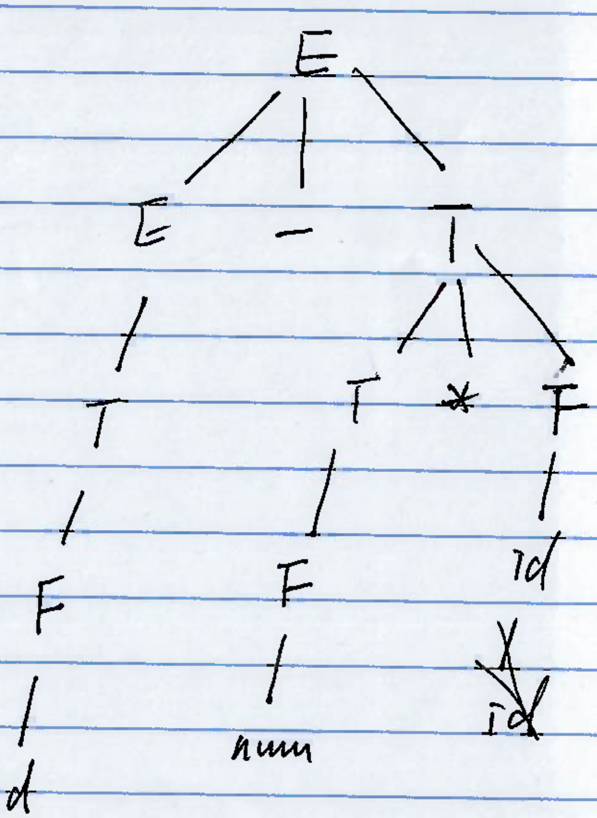
5 | T ✓ F

6 | F

7 F → (E)

8 | id

9 | num



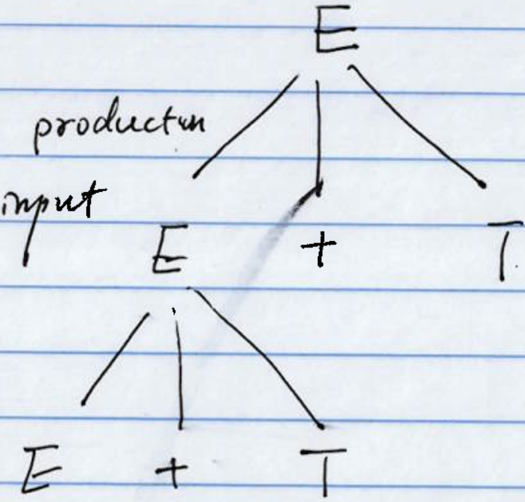
3.3. top-down parsing

build the tree (explicitly or implicitly), start with root. **build** the tree downward.

$x - 2 * y$

pick a production

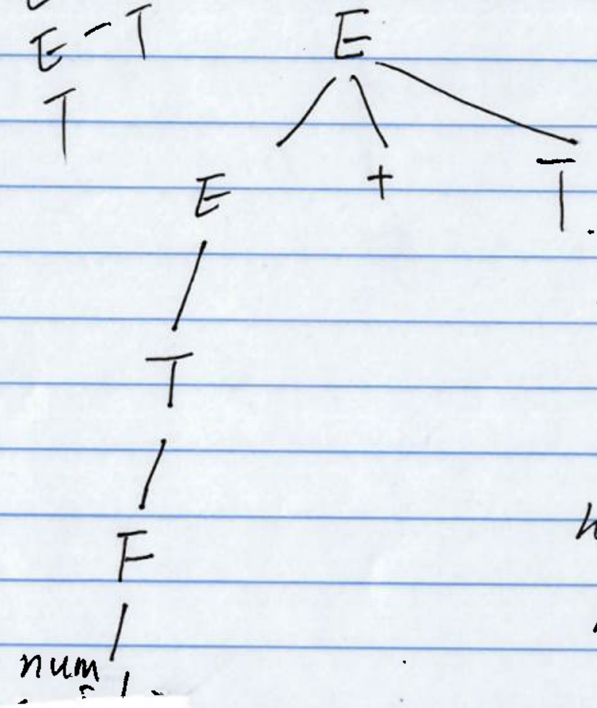
try to match input



avoid

"may never terminate"

$$E \to E+T \mid E-T \mid T$$



avoid / reduce.

∴ back track.

multiple rules that have the same left hand side.

$$F \to id \mid num \mid (E)$$

doesn't match

<x, id>

remove non-termination. / recursion.

For Leftmost derivation, we want to remove ~~left~~ recursion

$E \rightarrow \cancel{E} + E$

$E \rightarrow \underline{\underline{E}} + T$

$E \rightarrow T + E$

A grammar is left recursion if there exists $A \in N$ s.t. $A \overset{+}{\Longrightarrow} A\alpha$

one or more derivations.

for some string $\alpha \in (N \cup T)^+$

* indirect. left recursion

$$\begin{cases} A \rightarrow BC \\ B \rightarrow Ab \\ C \rightarrow d \end{cases}$$

$A \Rightarrow BC$
$\Rightarrow \underline{\underline{A}}bC$

not left recursion

$$\begin{cases} A \rightarrow BC \\ B \rightarrow bA \\ C \rightarrow d \end{cases}$$

$A \Rightarrow BC$.
$\Rightarrow bAC$

1. eliminating left recursion.

direct recursion.

$E \rightarrow E + T$
$\mid E - T$
$\mid T$

$\overset{?}{\Longrightarrow}$

$E \rightarrow T + E$
$\mid T - E$
$\mid T$

* right associative

direct recursion:

$$Fee \to Fee\,\alpha \mid \beta$$

$$A \to A\alpha \mid \beta$$

α. β. are strings not starting with Fee (A)

$$Fee \stackrel{*}{\Rightarrow} \beta(\alpha)^*$$

0 or more steps of derivation

$$A \stackrel{*}{\Rightarrow} \beta(\alpha)^*$$

$$B \stackrel{*}{\Rightarrow} \alpha^*$$

$$Fee \to \beta\ Fie$$

$$Fie \to \alpha\ Fie \mid \varepsilon$$

$$A \to \beta B$$

$$B \to \varepsilon \mid \alpha B \mid \cancel{B\alpha}$$

$a+b-c-d$

$$E \to E+T \mid E-T \mid T \quad \Longrightarrow \quad E \to TE'$$

$$E' \to +TE' \mid -TE' \mid \varepsilon$$

$$T \to T*F \mid T/F \mid F$$

$$T \to FT'$$

$$T' \to *FT' \mid /FT' \mid \varepsilon$$

0 $Goal \to E$

1 $E \to T E'$

2 $E' \to + T E'$

3 $\quad - T E'$

4 $\quad \varepsilon$

5 $T \to F T'$

6 $T' \to * F T'$

7 $\quad / F T'$

8 $\quad \varepsilon$

9 $F \to (E)$

10 $\quad | \text{ num}$

11 $\quad | \text{ id}$

$$\boxed{\begin{aligned} A &\to \underline{A \alpha} \\ &\mid \underline{\beta} \end{aligned}}$$

$$\left.\begin{aligned} A &\to ABC \\ &\mid a \\ &\mid b \end{aligned}\right\}$$

$$A \overset{*}{\Rightarrow} \begin{aligned} &a(BC)^* \\ &b(BC)^* \end{aligned}$$

$$\begin{aligned} A &\to aD \\ &\mid bD \end{aligned}$$

$$D \overset{*}{\Rightarrow} (BC)^*$$

$$\begin{aligned} D &\to \varepsilon \\ &\mid BCD \end{aligned}$$